RTPLATFORM MANUAL

TABLE OF CONTENTS

Revised October 22, 2004

CHAPTER 1

INTRODUCTION

RTPlatform is EBSnet's new cross-platform runtime environment. It defines an interface between the high-level platform-independent code in EBSnet's standalone products and the lower-level operating system/hardware environment.

RTPlatform is divided into a number of modules, each providing an interface to a specific service. For example, there is the rtpnet module, which defines a sockets-style interface to TCP/IP networking services, and rtpfile, which defines a roughly POSIX-style interface to file system services.

Some of the RTPlatform modules have platform-independent, or *generic*, implementations; others must rely on platform-specific code for their implementation (these are the *environment-specific* or *non-portable* modules). The release distribution of RTPlatform may include many different versions of the non-portable modules, each in a different directory that indicates the target environment. For example, the rtpnet module has an implementation for EBSnet's TCP/IP stack, RTIP, in "source/rtip/rtpnet.c", and an implementation for the Winsock library on 32-bit Microsoft Windows environments in "source/win32/rtpnet.c".

Although there may be many ".c" files (one for each target) corresponding to a particular module, there is usually only one ".h" file, located in the "include" directory. Therefore, because all of the header files are platform-independent, there are often no environment-specific header files included by the "core" source files of EBSnet's products. This greatly simplifies the porting process because it eliminates any potential symbol/namespace collisions.

Chapter 2

Generic (Platform-independent Modules)

GENERIC (PLATFORM-INDEPENDENT) MODULES

RTPDATE	-	Date Services
RTPDUTIL	-	Date/Timestamp Utilities
RTPMEMDB	-	Dynamic Memory Debugging Services
RTPPRINT	-	sprintf and related functions
RTPSTDUP	-	String Duplication Library Service
RTPBSEARCH	-	Binary Search Utility
RTPCHAR	-	Charcter Manipultation Services
RTPDEBUG	-	General Debugging Services
RTPDLLIST	-	Double Link List Utility
RTPFIO	-	High Level File Services
RTPHELPER	-	Helper Thread Services
RTPMTXDB	-	Mutex Debugging Services
RTPNETSM	-	Network State Machine Utility
RTPQSORT	-	Quick Sort Utility
RTPRAND	-	Pseudo Random Number Generator
RTPROT	-	Rotation Utility
RTPSCNV	-	String Conversion Utility
RTPSEMDB	-	Semaphore Debugging Services
RTPSTR	-	String Library Services
RTPTIMER	-	Timer Services
RTPWCHAR	-	Wide Character Manipulation Services
RTPWCS	-	Wide String Library Services
RTPWSCNV	-	Wide String Conversion Utility

RTPDATE

DATE SERVICES

rtp_set_date	- Set the current system date.
rtp_get_date	- Retrieve the current system date.

These functions sometimes need to be ported.

rtp_set_date

Function:

Set the current system date.

Summary:

int rtp_set_date (RTP_DATE* date)

Description:

Sets the system date to the new date passed in.

Parameters:

date - The new system date to be set.

Returns:

0 on succes; -1 otherwise.

rtp_get_date

Function:

Retrieve the current system date.

Summary:

int rtp_get_date (RTP_DATE* date)

Description:

Retrieves the current system date and returns it in the storage location.

Parameters:

date - Storage location for the current system date.

Returns:

0 on success; -1 otherwise.

RTPDUTIL

DATE/TIMESTAMP UTILITIES

rtp_get_utc_date	- Retrieve the current system deate in UTC.
rtp_date_utc_to_gmt	- Convert a UTC time to GMT.
rtp_date_to_timestamp	- Converts an RTP_DATE to an RTP_TIMESTAMP.
rtp_timestamp_to_date	- Converts an RTP_TIMESTAMP to an RTP_DATE.
rtp_date_get_timestamp	- Retrieves the current time.
rtp_date_print_time	- Print a date to a string.
rtp_date_parse_time	- Convert formatted string to an RTP_TIMESTAMP.
rtp_date_parse_time_uc	- Convert a formatted unicode string to an RTP_TIMESTAMP.
rtp_date_compare_time	- Compare two valid time stamps.
rtp_date_set_time_forever	- Set date to maximum value.
rtp_date_add_seconds	- Add a number of seconds to a date.
rtp_date_get_seconds_in_year	- Retrieve the number of seconds in a year.
rtp_date_time_difference	- Find the difference between two time stamps.
rtp_date_get_dayofweek	- Find day of the week.
rtp_date_get_seconds_by_date	- Find the number of seconds elapsed.

These functions **DO NOT** need to be ported.

Depends on:

RTPDATE

rtp_get_utc_date

Function:

Retrieve the current system date in UTC.

Summary:

unsigned long rtp_get_utc_date (void)

Description:

Retrieves the current system date in UTC. UTC is the number of seconds elapsed since midnight on January 1, 1970.

Returns

Returns the current UTC date as an unsigned long.

rtp_date_utc_to_gmt_

Function:

Convert a UTC time to GMT.

Summary:

int rtp_date_utc_to_gmt (RTP_DATE* gmDate, unsigned long utcSec)

Description:

Converts a UTC date obtained by calling rtp_get_utc_date to the GMT equivalent and stores the value in a RTP_DATE structure to be returned. UTC is the number of seconds elapsed since midnight on January 1, 1970. GMT is a structure equivalent of UTC time.

Parameters:

gmDate - Storage location for the converted GMT.

utcSec - UTC date to be converted.

Returns:

0 on success; -1 otherwise.

Preconditions:

The UTC date must be valid.

rtp_date_to_timestamp

Function:

Converts an RTP_DATE to an RTP_TIMESTAMP.

Summary:

void rtp_date_to_timestamp (RTP_TIMESTAMP* ptime,
    RTP_DATE* pdate)

Description:

Converts an RTP_DATE structure to an RTP_TIMESTAMP structure.

Parameters:

ptime - Storage location for the converted date.

pdate - Date to be converted.

Returns:

void

Preconditions:

The RTP_DATE * must refer to a valid RTP_DATE.

rtp_timestamp_to_date

Function:

Converts an RTP_TIMESTAMP to an RTP_DATE.

Summary:

void rtp_timestamp_to_date (RTP_DATE* pdate,
    RTP_TIMESTAMP* ptime)

Description:

Converts an RTP_TIMESTAMP structure to an RTP_DATE structure.

Parameters:

pdate - Storage location for the converted date.

ptime - Date to be converted.

Returns:

void

Preconditions:

The RTP_TIMESTAMP * must refer to a valid RTP_TIMESTAMP.

rtp_date_get_timestamp

Function:

Retrieves the current time.

Summary:

int rtp_date_get_timestamp (RTP_TIMESTAMP* ptime)

Description:

Retrieves the current time and returns it using the passed in RTP_TIMESTAMP structure pointer.

Parameters:

ptime	- Storage location for the converted date.

Returns:

0 on success; -1 otherwise.

rtp_date_print_time

Function:

Print a date to a string.

Summary:

int rtp_date_print_time (char* str, RTP_TIMESTAMP* ptime, int style)

Description:

Prints a date value contained in an RTP_TIMESTAMP structure to a formatted string. The string can have one of three formats depending on the value of [style].

1 Wednesday, 4-Feb-4 10:30:30 GMT

2 Wed Feb 4 10:30:30 2004

3 Wed, 4 Feb 2004 10:30:30 GMT

4 Feb 4 2004

Parameters:

str	- Storage location for converted time.
ptime	- Date to be formatted into a string.
style	- Style of string formatting to use.

Returns:

0 on success; -1 otherwise.

Preconditions:

The storage location must contain enough space to contain the formatted date value.

rtp_date_parse_time

Function:

Convert formatted string to an RTP_TIMESTAMP.

Summary:

int rtp_date_parse_time (RTP_TIMESTAMP* ptime, const char* str)

Description:

Extracts the time from a formatted string and returns it in an RTP_TIMESTAMP structure.

Parameters:

ptime	-	Storage location for extracted time.
str	-	Formatted string containing the date.

Returns:

0 on success, -1 otherwise.

rtp_date_parse_time_uc

Function:

Convert a formatted unicode string to an RTP_TIMESTAMP.

Summary:

int rtp_date_parse_time_uc (RTP_TIMESTAMP* ptime, const unsigned short* str)

Description:

Extracts the time from a formatted unicode string and returns it in an RTP_TIMESTAMP structure.

Parameters:

ptime	-	Storage location for extracted time.
str	-	Formatted string containing the date.

Returns:

0 on success, -1 otherwise.

rtp_date_compare_time

Function:

Compare two valid time stamps.

Summary:

int rtp_date_compare_time (RTP_TIMESTAMP* ptime1,
RTP_TIMESTAMP* ptime2)

Description:

Compare two valid time stamps.

Parameters:

ptime1	-	Time 1.
ptime2	-	Time 2.

Returns:

0 if equivalent; -1 if ptime1 < ptime2; 1 if ptime1 > ptime2.

rtp_date_set_time_forever

Function:

Set date to maximum value.

Summary:

void rtp_date_set_time_forever (RTP_TIMESTAMP* ptime)

Description:

Set date to maximum value.

Parameters:

ptime	-	Storage location for max time.

Returns:

void

rtp_date_add_seconds

Function:

Add a number of seconds to a date.

Summary:

void rtp_date_add_seconds (RTP_TIMESTAMP* ptime, long seconds)

Description:

Add seconds to an RTP_TIMESTAMP.

Parameters:

ptime	-	Valid timestamp.
seconds	-	Number of seconds to add to the timestamp.

Returns:

void

rtp_date_get_seconds_in_year

Function:

Retrieve the number of seconds in a year.

Summary:

long rtp_date_get_seconds_in_year (short year)

Description:

Retrieve the number of seconds elapsed in a given year.

Parameters:

year	-	The year in question.

Returns:

Number of seconds in a given year.

rtp_date_time_difference

Function:

Find the difference between two time stamps.

Summary:

long rtp_date_time_difference (RTP_TIMESTAMP* pearlier,
    RTP_TIMESTAMP* plater)

Description:

Retrieve the number of seconds between two time stamps.

Parameters:

pearlier	-	Time 1.
plater	-	Time 2.

Returns:

Delta number of seconds.

rtp_date_get_dayofweek

Function:

Find day of the week.

Summary:

int rtp_date_get_dayofweek (long year, long month, long day)

Description:

Retrieve the day of the week from the specified year, month, and day.

Parameters:

year	-	Four digit decimal value of the year.
month	-	1 to 12 identifying the month in the year.
day	-	Decimal value of the day of the month.

Returns:

0 to 6 inclusive, identifying the day of the week.

rtp_date_get_seconds_by_date

Function:

Find the number of seconds elapsed.

Summary:

long rtp_date_get_seconds_by_date (short year,   short month,
short day,   short hour, short minute, short second)

Description:

Find the number of seconds elapsed until the date passed in.

Parameters:

year	-	Four digit decimal value of the year.
month	-	1 to 12 identifying the month in the year.
day	-	Decimal value of the day of the month.
hour	-	Number of hours in the day.
minute	-	Number of minutes in the hour.
second	-	Number of seconds in the minute.

Returns:

Number of seconds elapsed.

RTPMEMDB

DYNAMIC MEMORY DEBUGGING SERVICES

_rtp_debug_malloc	- Allocate memory.
_rtp_debug_realloc	- Reallocate memory.
_rtp_debug_free	- Free memory.
_rtp_debug_SetHighWaterCallback	-
_rtp_debug_CheckMemBlockList	- Checks the memory block list.
_rtp_debug_PrintMemLeaks	- Identify any memory corruption.

These functions **DO NOT** need to be ported.

Depends on:

RTPMEM

_rtp_debug_malloc

Function:

Allocate memory.

Summary:

void* _rtp_debug_malloc (unsigned int size,
    RTP_MALLOC_FN allocFn, const char* file, long
    line_num)

Description:

Allocates, if possible, a portion of memory of 'size' bytes.

Parameters:

size	- Number of bytes to allocate.
allocFn	- Memory allocation function pointer.
file	- The preprocessor macro __FILE__.
line_num	- The preprocessor macro __LINE__.

Returns:

Pointer to the location of the allocated spcae, 0 otherwise. For debugging purposes; if the cause of the error is obtainable at the native Memory System layer, turn on RTP_DISPLAY_ERROR in rtpmem.c to display the native error value.

Preconditions:

Only use if dynamic memory is required. MUST not be called directly. Use the **rtp_malloc** macro defined in rtpmem.h and turn on RTP_TRACK_LOCAL_MEMORY. If the preprocessor macros used are unavailable in the target environment, define them at the compiler to a chosen dummy value.

_rtp_debug_realloc

Function:

Reallocate memory.

Summary:

void* _rtp_debug_realloc (void* ptr, unsigned int size, RTP_REALLOC_FN reallocFn, const char* file, long line_num)

Description:

Allocates, if possible, a portion of memory of 'size' bytes, then moves 'size' bytes, if possible, of the old region ptr was pointing to, to the newly allocated space.

Parameters:

ptr	-	Pointer to currently allocated space.
size	-	New number of bytes to allocate.
reallocFn	-	Memory reallocation function pointer.
file	-	The preprocessor macro __FILE__.
line_num	-	The preprocessor macro __LINE__.

Returns:

Pointer to the location of the newly allocated spcae, 0 otherwise. For debugging purposes; if the cause of the error is obtainable at the native Memory System layer, turn on RTP_DISPLAY_ERROR in rtpmem.c to display the native error value.

Preconditions:

Only use if dynamic memory is required. MUST not be called directly. Use the **rtp_realloc** macro defined in rtpmem.h and turn on RTP_TRACK_LOCAL_MEMORY. If the preprocessor macros used are unavailable in the target environment, define them at the compiler to a chosen dummy value.

_rtp_debug_free

Function:

Free memory.

Summary:

void _rtp_debug_free (void* ptr, RTP_FREE_FN _free, const char* file, long line_num)

Description:

Frees the portion of memory that ptr was allocated to by _rtp_debug_malloc or _rtp_debug_realloc.

Parameters:

ptr	-	Location that needs to be freed.
_free	-	Memory deallocation function pointer.
file	-	The preprocessor macro __FILE__.
line_num	-	The preprocessor macro __LINE__.

Returns:

void

Preconditions:

Only use if dynamic memory is required. MUST not be called directly. Use the **rtp_free** macro defined in rtpmem.h and turn on RTP_TRACK_LOCAL_MEMORY. If the preprocessor macros used are unavailable in the target environment, define them at the compiler to a chosen dummy value.

_rtp_debug_CheckMemBlockList

Function:

Checks the memory block list.

Summary:

void _rtp_debug_CheckMemBlockList (const char* logFile,
    const char* mode)

Description:

Checks the memory block list and print the memory debug session information to the indicated logFile.

Returns:

void

Preconditions:

Only use if dynamic memory is required. MUST not be called directly. Use the **rtp_debug_CheckMemBlockList** macro defined in rtpmem.h and turn on RTP_TRACK_LOCAL_MEMORY. The rtpstdio file system support and/or the rtpterm terminal layer must be present to make use of this debugging feature.

_rtp_debug_PrintMemLeaks

Function:

Identify any memory corruption.

Summary:

void _rtp_debug_PrintMemLeaks (const char* logFile, const
    char* mode)

Description:

Identify any memory corruptions. If RTP_MEM_DEBUG_FILE this prints the information to a file indicated by the logFile name passed in. If the logFile is NULL and RTP_MEM_DEBUG_PRINTF is defined, the information will be sent to rtp_printf.

Returns:

void

Preconditions:

Only use if dynamic memory is required. MUST not be called directly. Use the **rtp_debug_PrintMemLeaks** macro defined in rtpmem.h and turn on RTP_TRACK_LOCAL_MEMORY. The rtpfio file system support and/or the rtpterm terminal layer must be present to make use of this debugging feature.

RTPPRINT

SPRINTF AND RELATED FUNCTIONS

rtp_printf	- Print a formatted string to the terminal using rtp_term_cputs.
rtp_vprintf	- Print a formatted string to the terminal using rtp_term_cputs.
rtp_sprintf	- Print formatted data to a string.
rtp_snprintf	- Print formatted data to a string.
rtp_vsprintf	- Print formatted data to a string.
rtp_vsnprintf	- Print formatted data to a string.

These functions **DO NOT** need to be ported.

rtp_printf

Function:

Print a formatted string to the terminal using rtp_term_cputs.

Summary:

int rtp_printf (const char* f, .. Optional arguments for the formatted string.* /)

Description:

Print a formatted string to the terminal using rtp_term_cputs. Format the string using the variable length argument list.

Parameters:

f	- Formatted string.

Returns:

Number of characters printed.

Preconditions:

If not using the native system implementation of these functions via the macros in rtpprint.h and this funcitionality is desired, RTP_INCLUDE_PRINTF must be set to 1 in rtpprint.h.

rtp_vprintf

Function:

Print a formatted string to the terminal using rtp_term_cputs.

Summary:

int rtp_vprintf (const char* f, rtp_va_list ap)

Description:

Print a formatted string to the terminal using rtp_term_cputs. Format the string using the list of arguments passed in using rtp_va_list.

Parameters:

f	-	Formatted string.
ap	-	Arguments for the formatted string.

Returns:

Number of characters printed.

Preconditions:

If not using the native system implementation of these functions via the macros in rtpprint.h and this funcitionality is desired, RTP_INCLUDE_PRINTF must be set to 1 in rtpprint.h.

rtp_sprintf

Function:

Print formatted data to a string.

Summary:

int rtp_sprintf (char* wh, const char* f,... Optional arguments for the formatted data.* /)

Description:

Print formatted data to a string. Format the data using the variable length argument list.

Parameters:

wh	-	Storage location string for the formatted data.
f	-	Formatted data.

Returns:

Number of characters printed.

Preconditions:

If not using the native system implementation of these functions via the macros in rtpprint.h and this funcitionality is desired, RTP_INCLUDE_SPRINTF must be set to 1 in rtpprint.h.

rtp_snprintf

Function:

Print formatted data to a string.

Summary:

int rtp_snprintf (char* wh, int bufflen, const char* f, ...
 Optional arguments for the formatted data.* /)

Description:

Print formatted data to a string. Format the data using the variable length argument list. Limit the number of characters to be formatted to the bufflen.

Parameters:

wh	-	Storage location string for the formatted data.
bufflen	-	Number of characters to store.
f	-	Formatted data.

Returns:

Number of characters printed.

Preconditions:

If not using the native system implementation of these functions via the macros in rtpprint.h and this funcitionality is desired, RTP_INCLUDE_SPRINTF must be set to 1 in rtpprint.h.

rtp_vsprintf

Function:

Print formatted data to a string.

Summary:

int rtp_vsprintf (char* wh, const char* f, rtp_va_list ap)

Description:

Print formatted data to a string. Format the data using the argument list.

Parameters:

wh	-	Storage location string for the formatted data.
f	-	Formatted data.
ap	-	Arguments for the formatted string.

Returns:

Number of characters printed.

Preconditions:

If not using the native system implementation of these functions via the macros in rtpprint.h and this funcitionality is desired, RTP_INCLUDE_SPRINTF must be set to 1 in rtpprint.h.

rtp_vsnprintf

Function:

Print formatted data to a string.

Summary:

```
int rtp_vsnprintf ( char*  wh,   long bufflen,   const char*  f,
        rtp_va_list ap )
```

Description:

Print formatted data to a string. Format the data using the argument list. Limit the number of characters to be formatted to the bufflen.

Parameters:

wh	- Storage location string for the formatted data.
bufflen	- Number of characters to store.
f	- Formatted data.
ap	- Arguments for the formatted string.

Returns:

Number of characters printed.

Preconditions:

If not using the native system implementation of these functions via the macros in rtpprint.h and this funcitionality is desired, RTP_INCLUDE_SPRINTF must be set to 1 in rtpprint.h.

RTPFIO

HIGH LEVEL FILE SERVICES

rtp_stdio_fopen	- Opens a file using the supplied name.
rtp_stdio_fclose	- Close a file.
rtp_stdio_fread	- Read from a file to buffer.
rtp_stdio_fwrite	- Write a buffer to a file.
rtp_stdio_fprintf	- Write a formatted buffer to a file.
rtp_stdio_fgets	- Read lines from a file to a buffer.
rtp_stdio_fseek	- Moves the file pointer to the offset from the origin.
rtp_stdio_fileno	- Extract the RTP_HANDLE from the RTP_FILE returned by rtp_stdio_fopen.
rtp_stdio_feof	- Determines if at the end of the file.

These functions **DO NOT** need to be ported.

rtp_stdio_fopen

Function:

Opens a file using the supplied name.

Summary:

int rtp_stdio_fopen (void** rtpfile, const char* fname, const char* mode)

Description:

Opens a file using the mode settings, and sets the file information.

Parameters:

rtpfile	- A pointer to the address of an RTP_FILE to store the file information.
fname	- The name of the file to open.
mode	- For the mode argument:

"r"	Open a file for reading only	RTP_FILE_O_RDONLY
"r+"	Open a file for reading and writing	RTP_FILE_O_RDWR
"w"	Open a file for writing only	RTP_FILE_O_WRONLY
	Create a file if it does not exist	RTP_FILE_O_CREAT
	Truncate a file to 0 bytes after opening	RTP_FILE_O_TRUNC
"w+"	Open a file for reading and writing	RTP_FILE_O_RDWR
	Create a file if it does not exist	RTP_FILE_O_CREAT
	Truncate a file to 0 bytes after opening	RTP_FILE_O_TRUNC
"a"	All writes will be appended to the file	RTP_FILE_O_APPEND
	Create a file if it does not exist	RTP_FILE_O_CREAT
"a+"	Open a file for reading and writing	RTP_FILE_O_RDWR
	All writes will be appended to the file	RTP_FILE_O_APPEND
	Create a file if it does not exist	RTP_FILE_O_CREAT
"b..."	b prepended to the mode in the open call causes the file to be opened in binary mode	RTP_FILE_O_BINARY
"t..."	t prepended to the mode in the open call causes the file to be opened in text mode	RTP_FILE_O_TEXT
Note:	**If neither the b or t are prepended, text mode is the default implementation.**	

Returns:

0 if successful, -1 on failure, and -2 if a non-fatal error occurred.

rtp_stdio_fclose

Function:

Close a file.

Summary:

int rtp_stdio_fclose (void* rtpfile)

Description:

Closes a file that was opened with rtp_stdio_fopen.

Parameters:

rtpfile	-	File handle used to close the file.

Returns:

0 if successful, -1 otherwise.

rtp_stdio_fread

Function:

Read from a file to buffer.

Summary:

long rtp_stdio_fread (void* buffer,   unsigned long len, unsigned long n,   void* rtpfile)

Description:

Reads n items of size len from the current position in the rtpfile and stores them in buffer.

Parameters:

buffer	-	Buffer to store the read items.
len	-	Size of items to read.
n	-	Number of items to read.
rtpfile	-	Handle of file to be read.

Returns:

Number of items read, 0 if end of file, -1 on error, and -2 on non-fatal error.

Preconditions:

Should not mix rtp_stdio_fread and rtp_stdio_fgets calls.

rtp_stdio_fwrite

Function:

Write a buffer to a file.

Summary:

long rtp_stdio_fwrite (void* buffer, unsigned long len,
unsigned long n, void* rtpfile)

Description:

Write n items of size len from the buffer to the current position in the file.

Parameters:

buffer	-	Buffer to write.
len	-	Size of items to write.
n	-	Number of items to write.
rtpfile	-	Handle of the file to be written to.

Returns:

Number of items written, -1 on error, and -2 on non-fatal error.

Preconditions:

Should not mix rtp_stdio_fwrite() and rtp_stdio_fgets() calls.

rtp_stdio_fprintf

Function:

Write a formatted buffer to a file.

Summary:

long rtp_stdio_fprintf (void* rtpfile, const char* f, ...
(optional) Arguments to be formatted.* /)

Description:

Write n formatted bytes from the buffer to the current position in the file.

Parameters:

rtpfile	-	Handle of the file to be written to.
f	-	Formatted buffer.

Returns:

Number of bytes written, -1 on error, and -2 on non-fatal error.

rtp_stdio_fgets

Function:

Read lines from a file to a buffer.

Summary:

int rtp_stdio_fgets (char* buf, int buflen, void* rtpfile)

Description:

Read lines from a file to buf or if the end of the file or the line has not been reached yet, reads (buflen-1) of data.

Parameters:

buf	- Buffer to store the read.
buflen	- Number of bytes to read.
rtpfile	- Handle of file to be read.

Returns:

Number of bytes written, -1 otherwise.

Preconditions:

Should not mix rtp_stdio_fread and rtp_stdio_fgets calls.

rtp_stdio_fseek

Function:

Moves the file pointer to the offset from the origin.

Summary:

long rtp_stdio_fseek (void* rtpfile, long offset, int origin)

Description:

Move the file pointer to the offset in the file relative to the origin.

Parameters:

rtpfile	- Handle of file to move its pointer.
offset	- The offset to move the pointer.
origin	- The location that the offset is relative to:
	0 To seek from the beginning of the file.
	1 To seek from the current file pointer position.
	2 To seek from the end of the file.

Returns:

0 on success, -1 otherwise.

rtp_stdio_fileno

Function:

Extract the RTP_HANDLE from the RTP_FILE returned by rtp_stdio_fopen.

Summary:

int rtp_stdio_fileno (RTP_HANDLE* handle, void* rtpfile)

Description:

Extract the RTP_HANDLE from the RTP_FILE returned by rtp_stdio_fopen.

Parameters:

handle	-	Storage location for the extracted RTP_HANDLE.
rtpfile	-	Descriptor used to extract the underlying RTP_HANDLE.

Returns:

0 on success, -1 otherwise.

rtp_stdio_feof

Function:

Determines if at the end of the file.

Summary:

int rtp_stdio_feof (void* rtpfile)

Description:

Determines if the internal file pointer is at the end of the file.

Parameters:

rtpfile	-	Handle of file to check eof.

Returns:

0 if rtp_stdio_fread has not read to the end of the file, -1 in the case of an error (invalid RTP_FILE *, or bad position in the file), and 1 if reached the end of the file.

RTPSTDUP

STRING DUPLICATION LIBRARY SERVICE

rtp_strdup	-	Duplicate a string.

rtp_strdup

Function:

Duplicate a string.

Summary:

char* rtp_strdup (const char* str)

Description:

Use dynamic memory supplied by rtpmem to duplicate a string.

Parameters:

str	-	Constant string to be duplicated.

Returns:

The duplicated string. (char*) 0 on error.

Preconditions:

Depends on rtpmem and rtpmemdb in the case that RTP_TRACK_LOCAL_MEMORY is defined.

STANDARD UTILITY LIBRARY SERVICES

RTPCHAR

rtp_isalnum	- Check if a value is a character or digit.
rtp_iscntrl	- Check if a character is a control character.
rtp_isdigit	- Check if a character is a digit.
rtp_isprint	- Check if a character is printable.
rtp_isspace	- Check if a character is a space.
rtp_isupper	- Check if a character is an uppercase character.
rtp_isxdigit	- Check if a character is a hexidecimal digit.
rtp_tolower	- Convert character to lower case.
rtp_toupper	- Convert character to upper case.

RTPROT

rtp_lrotl	- Long rotate left.
rtp_lrotr	- Long rotate right.

RTPSCNV

rtp_hatoi	- Convert a hex string to a decimal integer.
rtp_atoi	- Convert a string to an integer.
rtp_atol	- Convert a string to a long.
rtp_strtol	- Find a string within a string.
rtp_strtoul	- Convert a string to an unsigned long.
rtp_itoa	- Convert an integer to a string.
rtp_ltoa	- Convert an long to a string.
rtp_ultoa	- Convert an unsigned long to a string.

RTPSTR

rtp_memcat	- Write a data into the middle of a buffer, return total length
rtp_stricmp	- Compare characters of one buffer with another.
rtp_stristr	- Find a string within a string.
rtp_strncmp	- Compare characters of one buffer with another.
rtp_strnicmp	- Compare characters of one buffer with another.
rtp_memchr	- Finds a character in a buffer.
rtp_memcmp	- Compare bytes of one buffer with another.
rtp_memcpy	- Copies bytes of one buffer to another.
rtp_memmove	- Copies bytes of one buffer to another.
rtp_memset	- Set a number of bytes in a buffer.
rtp_strcat	- Copies the source string to the end of the destination string.
rtp_strchr	- Find a character within a string.
rtp_strcmp	- Compare characters of one string with another.
rtp_strcpy	- Copies the source string to the destination.
rtp_strlen	- Finds the length of a string.
rtp_strncpy	- Copies the source string to the destination.
rtp_strrchr	- Find the last occurance of a character within a string.
rtp_strstr	- Find a string within a string.

RTPQSORT

rtp_qsort	- Quick sort using the indicated comparison function.

RTPBSEARCH

rtp_bsearch	- Binary search of an already sorted array.

rtp_rand	- Genearate pseudorandom numbers.
rtp_srand	- Seed the pseudorandom number generator.

These functions **DO NOT** need to be ported.

RTPCHAR

rtp_isalnum

Function:

Check if a value is a character or digit.

Summary:

int rtp_isalnum (int chr)

Description:

Determine if 'chr' is an alphanumeric character (between 0 and 9, a and z, or A and Z).

Parameters:

chr	-	Value to check.

Returns:

Non-zero if it is a hexidecimal digit, 0 if not a digit.

rtp_iscntrl

Function:

Check if a character is a control character.

Summary:

int rtp_iscntrl (int ch)

Description:

Determines if ch is a control character.

Parameters:

ch	-	Character to check.

Returns:

1 if a control character, 0 if not a control character.

rtp_isdigit

Function:

Check if a character is a digit.

Summary:

int rtp_isdigit (int ch)

Description:

Determines if ch is a digit between 0 and 9.

Parameters:

ch	- Character to check.

Returns:

Greater than 0 if it is a digit, 0 if not a digit.

rtp_isprint

Function:

Check if a character is printable.

Summary:

int rtp_isprint (int ch)

Description:

Determines if ch is a printable character.

Parameters:

ch	- Character to check.

Returns:

1 if printable, 0 if not printable.

rtp_isspace

Function:

Check if a character is a space.

Summary:

int rtp_isspace (int ch)

Description:

Determines if ch is a space.

Parameters:

ch	-	Character to check.

Returns:

1 if a space, 0 if not a space.

rtp_isupper

Function:

Check if a character is an uppercase character.

Summary:

int rtp_isupper (int ch)

Description:

Determines if ch is an uppercase character.

Parameters:

ch	-	Character to check.

Returns:

1 if an uppercase character, 0 if not an uppercase character.

rtp_isxdigit

Function:

Check if a character is a hexidecimal digit.

Summary:

int rtp_isxdigit (int chr)

Description:

Determines if ch is a hexidecimal digit between 0 and 9, a and f, or A and F.

Parameters:

chr	- Character to check.

Returns:

1 if a hexidecimal digit, 0 if not a hexidecimal digit.

rtp_tolower

Function:

Convert character to lower case.

Summary:

int rtp_tolower (int c)

Description:

Convert c to lower case.

Parameters:

c	- Character to convert.

Returns:

c in lower case if c is a character.

rtp_toupper

Function:

Convert character to upper case.

Summary:

int rtp_toupper (int c)

Description:

Convert c to upper case.

Parameters:

c	-	Character to convert.

Returns:

c in upper case if c is a character.

RTPROT

rtp_lrotl

Function:

Long rotate left.

Summary:

unsigned long rtp_lrotl (unsigned long rotate,   int nbits)

Description:

Rotate the value 'rotate' to the left 'nbits'.

Parameters:

rotate	-	The value to be rotated.
nbits	-	Number of bits to rotate the value.

Returns:

The rotated value.

RTPSCNV

rtp_atoi

Function:

Convert a string to an integer.

Summary:

int rtp_atoi (const char* s)

Description:

Convert s to an integer.

Parameters:

s	-	String to be converted.

Returns:

Integer value of s.

rtp_atol

Function:

Convert a string to a long.

Summary:

long rtp_atol (const char* s)

Description:

Convert s to a long.

Parameters:

s	- String to be converted.

Returns:

Long value of s.

rtp_strtol

Function:

Find a string within a string.

Summary:

long rtp_strtol (const char* str, char** delimiter, int base)

Description:

Find last occurance of find_chr in str.

Parameters:

str	- String to be converted.
delimiter	- Character that stops the convertion.
base	- Base of return value.

Returns:

The last occurance of find_chr found in str, 0 if not found.

rtp_strtoul

Function:

Convert a string to an unsigned long.

Summary:

unsigned long rtp_strtoul (const char* str, char** delimiter, int base)

Description:

Determine the long value of the passed string. If base is 0 then the base is determined by the format of the string; otherwise the base is assumed to be correct.

Parameters:

str	-	String to be converted.
delimiter	-	Character that stops the convertion.
base	-	Base of return value.

Returns:

Unsigned long representation of the passed string to the delimiter, 0 on error.

rtp_itoa

Function:

Convert an integer to a string.

Summary:

char* rtp_itoa (int num, char* dest, int base)

Description:

Convert num to a string using base and store it in dest.

Parameters:

num	-	Integer to convert.
dest	-	Destination string.
base	-	Base of conversion.

Returns:

Value of dest.

rtp_ltoa

Function:

Convert an long to a string.

Summary:

char* rtp_ltoa (long num,   char* dest,   int base)

Description:

Convert num to a string using base and store it in dest.

Parameters:

num	-	Long to convert.
dest	-	Destination string.
base	-	Base of conversion.

Returns:

Value of dest.

rtp_ultoa

Function:

Convert an unsigned long to a string.

Summary:

char* rtp_ultoa (unsigned long num,   char* dest,   int base)

Description:

Convert num to a string using base and store it in dest.

Parameters:

num	-	Unsigned long to convert.
dest	-	Destination string.
base	-	Base of conversion.

Returns:

Value of dest.

RTPSTR

rtp_memcat

Function:

Write a data into the middle of a buffer, return total length

Summary:

long rtp_memcat (char* dst, long offset, const char* src, long size)

Description:

copies size bytes from src to offset]

Returns:

offset + size

rtp_stricmp

Function:

Compare characters of one buffer with another.

Summary:

int rtp_stricmp (const char* s1,   const char* s2)

Description:

Compare characters of s1 to characters of s2 ignoring case.

Parameters:

s1	- Pointer to buffer to be compared.
s2	- Pointer to buffer to be compared.

Returns:

Less than 0 if s1 s2, and 0 if s1 == s2.

rtp_stristr

Function:

Find a string within a string.

Summary:

char* rtp_stristr (const char* str,   const char* find_str)

Description:

Find find_str in str ignoring case.

Parameters:

str	-	String to search.
find_str	-	String to find.

Returns:

The address of the first place that find_str is found in str, 0 if not found.

rtp_strncmp

Function:

Compare characters of one buffer with another.

Summary:

int rtp_strncmp (const char* s1,   const char* s2, unsigned int n)

Description:

Compare a number of characters of s1 to a number of characters of s2.

Parameters:

s1	-	Pointer to buffer to be compared.
s2	-	Pointer to buffer to be compared.
n	-	Number of characters to compare.

Returns:

Less than 0 if s1 s2, and 0 if s1 == s2.

rtp_strnicmp

Function:

Compare characters of one buffer with another.

Summary:

int rtp_strnicmp (const char* s1, const char* s2, unsigned int n)

Description:

Compare a number of characters of s1 to a number of characters of s2, ignoring case.

Parameters:

s1	-	Pointer to buffer to be compared.
s2	-	Pointer to buffer to be compared.
n	-	Number of characters to compare.

Returns:

Less than 0 if s1 s2, and 0 if s1 == s2.

rtp_memchr

Function:

Finds a character in a buffer.

Summary:

void* rtp_memchr (const void* str,   int chr,   unsigned int n)

Description:

Finds the first occurance of chr in str.

Parameters:

str	-	The buffer to search.
chr	-	The character to find.
n	-	Number of characters to search in the buffer.

Returns:

The location of chr in str, 0 if not found.

rtp_memcmp

Function:

Compare bytes of one buffer with another.

Summary:

int rtp_memcmp (const void* a,   const void* b, unsigned int n)

Description:

Compare n bytes of a to n bytes of b.

Parameters:

a	-	Pointer to a buffer to be compared.
b	-	Pointer to a buffer to be compared.
n	-	Number of bytes to compare.

Returns:

Less than 0 if a b, and 0 if a == b.

rtp_memcpy

Function:

Copies bytes of one buffer to another.

Summary:

void* rtp_memcpy (void* a, const void* b, unsigned int n)

Description:

Copies n bytes of b to a.

Parameters:

a	-	Pointer to destination buffer.
b	-	Pointer to source buffer.
n	-	Number of bytes to copy.

Returns:

Value of a.

rtp_memmove

Function:

Copies bytes of one buffer to another.

Summary:

void* rtp_memmove (void* a, const void* b, unsigned int n)

Description:

Copies n bytes of b to a, and if they overlap, b is copied over to a before it is lost.

Parameters:

a	- Pointer to destination buffer.
b	- Pointer to source buffer.
n	- Number of bytes to copy.

Returns:

Value of a.

rtp_memset

Function:

Set a number of bytes in a buffer.

Summary:

void* rtp_memset (void* p, int b, unsigned int n)

Description:

Sets n bytes of p to the value of b.

Parameters:

p	- Bbuffer to be set.
b	- Value to set the buffer's bytes to.
n	- Number of bytes to set.

Returns:

void

rtp_strcat

Function:

Copies the source string to the end of the destination string.

Summary:

char* rtp_strcat (char* a, const char* b)

Description:

Appends string b to string a.

Parameters:

a	-	Destination buffer.
b	-	Source buffer.

Returns:

Value of a.

rtp_strchr

Function:

Find a character within a string.

Summary:

char* rtp_strchr (const char* str, int find_chr)

Description:

Find a find_chr within a str.

Parameters:

str	-	String to search.
find_chr	-	Character to find.

Returns:

The first place find_chr is found in str, 0 if not found.

rtp_strcmp

Function:

Compare characters of one string with another.

Summary:

int rtp_strcmp (const char* s1, const char* s2)

Description:

Compare characters of s1 to characters of s2.

Parameters:

s1	-	String to be compared.
s2	-	String to be compared.

Returns:

Less than 0 if s1 s2, and 0 if s1 == s2.

rtp_strcpy

Function:

Copies the source string to the destination.

Summary:

char* rtp_strcpy (char* a, const char* b)

Description:

Copies string b to string a.

Parameters:

a	-	Destination buffer.
b	-	Source buffer.

Returns:

Value of a.

rtp_strlen

Function:

Finds the length of a string.

Summary:

unsigned int rtp_strlen (const char* string)

Description:

Determines the length of string.

Parameters:

string	- String to find the length of.

Returns:

Length of the string.

rtp_strncpy

Function:

Copies the source string to the destination.

Summary:

char* rtp_strncpy (char* a, const char* b, unsigned int n)

Description:

Copies n characters of string b to string a.

Parameters:

a	- Destination buffer.
b	- Source buffer.
n	- Number of characters to copy from.

Returns:

Value of a.

rtp_strrchr

Function:

Find the last occurance of a character within a string.

Summary:

char* rtp_strrchr (const char* str, int find_chr)

Description:

Find last occurance of find_chr in str.

Parameters:

str	-	String to search.
find_chr	-	Character to find.

Returns:

The last occurance of find_chr found in str, 0 if not found.

rtp_strstr

Function:

Find a string within a string.

Summary:

char* rtp_strstr (const char* str, const char* find_str)

Description:

Find last occurance of find_chr in str.

Parameters:

str	-	String to search.
find_str	-	String to find.

Returns:

The last occurance of find_chr found in str, 0 if not found.

RTPQSORT

rtp_qsort

Function:

Quick sort using the indicated comparison function.

Summary:

void rtp_qsort (void* head, unsigned long num, unsigned long
size, RTP_COMPARISON_FN compfunc)

Description:

Quick sort using the comparison function supplied 'compfunc'. The comparison function must return:

0 If node1 is greater than to node2.

Parameters:

head	-	Head of array to sort.
num	-	Number of nodes in array.
size	-	Size of each node.
compfunc	-	Comparison function.

Returns:

void

RTPBSEARCH

rtp_bsearch

Function:

Binary search of an already sorted array.

Summary:

void* rtp_bsearch (const void* obj, const void* head, unsigned int num, unsigned int size, RTP_COMPARISON_FN compfunc)

Description:

Binary search using the comparison function supplied 'compfunc'. The comparison function must return:

0 If node1 is greater than to node2.

Parameters:

obj	-	Object that is to be found.
head	-	Head of array to search.
num	-	Number of nodes in array.
size	-	Size of each node.
compfunc	-	Comparison function.

Returns:

Pointer to the found node within the array, 0 if not found or error.

WIDE CHARACTER STRING LIBRARY SERVICES

RTPWCHAR

rtp_iswalnum	- Check if a value is a character or digit.
rtp_iswcntrl	- Check if a character is a control character.
rtp_iswdigit	- Check if a character is a digit.
rtp_iswprint	- Check if a character is printable.
rtp_iswspace	- Check if a character is a space.
rtp_iswupper	- Check if a character is an uppercase character.
rtp_iswxdigit	- Check if a character is a hexidecimal digit.
rtp_towlower	- Convert character to lower case.
rtp_towupper	- Convert character to upper case.

RTPWCS

rtp_wcsicmp	- Compare characters of one unicode buffer with another.
rtp_wcsistr	- Find a unicode string within a string.
rtp_wcsncmp	- Compare characters of one unicode buffer with another.
rtp_wcsnicmp	- Compare characters of one unicode buffer with another.
rtp_wcscat	- Copies the source string to the end of the destination string.
rtp_wcschr	- Find a character within a string.
rtp_wcscmp	- Compare characters of one string with another.
rtp_wcscpy	- Copies the source string to the destination.
rtp_wcslen	- Finds the length of a string.
rtp_wcsncpy	- Copies the source string to the destination.
rtp_wcsrchr	- Find the last occurance of a character within a string.
rtp_wcsstr	- Find a string within a string.

RTPWSCNV

rtp_wtoi	- Convert a unicode string to an integer.
rtp_wtol	- Convert a unicode string to a long.
rtp_wcstol	- Find a string within a string.
rtp_wcstoul	- Convert a string to an unsigned long.
rtp_itow	- Convert an integer to a unicode string.
rtp_ltow	- Convert an long to a unicode string.
rtp_ultow	- Convert an unsigned long to a uniocde string.

These functions **DO NOT** need to be ported.

RTPWCHAR

rtp_iswalnum

Function:

Check if a value is a character or digit.

Summary:

int rtp_iswalnum (int chr)

Description:

Determine if 'chr' is an alphanumaric character (between 0 and 9, a and z, or A and Z).

Parameters:

chr	-	Value to check.

Returns:

Non-zero if it is a hexidecimal digit, 0 if not a digit.

rtp_iswcntrl

Function:

Check if a character is a control character.

Summary:

int rtp_iswcntrl (int ch)

Description:

Determines if ch is a control character.

Parameters:

ch	-	Character to check.

Returns:

1 if a control character, 0 if not a control character.

rtp_iswdigit

Function:

Check if a character is a digit.

Summary:

int rtp_iswdigit (int ch)

Description:

Determines if ch is a digit between 0 and 9.

Parameters:

ch	- Character to check.

Returns:

Greater than 0 if it is a digit, 0 if not a digit.

rtp_iswprint

Function

Check if a character is printable.

Summary:

int rtp_iswprint (int ch)

Description:

Determines if ch is a printable character.

Parameters:

ch	- Character to check.

Returns:

1 if printable, 0 if not printable.

rtp_iswspace

Function:

Check if a character is a space.

Summary:

int rtp_iswspace (int ch)

Description:

Determines if ch is a space.

Parameters:

ch	- Character to check.

Returns:

1 if a space, 0 if not a space.

rtp_iswupper

Function:

Check if a character is an uppercase character.

Summary:

int rtp_iswupper (int ch)

Description:

Determines if ch is an uppercase character.

Parameters:

ch	- Character to check.

Returns:

1 if an uppercase character, 0 if not an uppercase character.

rtp_iswxdigit

Function:

Check if a character is a hexidecimal digit.

Summary:

int rtp_iswxdigit (int chr)

Description:

Determines if ch is a hexidecimal digit between 0 and 9, a and f, or A and F.

Parameters:

chr	-	Character to check.

Returns:

1 if a hexidecimal digit, 0 if not a hexidecimal digit.

rtp_towlower

Function:

Convert character to lower case.

Summary:

int rtp_towlower (int c)

Description:

Convert c to lower case.

Parameters:

c	-	Character to convert.

Returns:

c in lower case if c is a character.

rtp_towupper

Function:

Convert character to upper case.

Summary:

int rtp_towupper (int c)

Description:

Convert c to upper case.

Parameters:

c - Character to convert.

Returns:

c in upper case if c is a character.

RTPWCS

rtp_wcsicmp

Function:

Compare characters of one unicode buffer with another.

Summary:

int rtp_wcsicmp (const unsigned short* s1, const unsigned short* s2)

Description:

Compare characters of s1 to characters of s2 ignoring case.

Parameters:

s1	-	Pointer to unicode buffer to be compared.
s2	-	Pointer to unicode buffer to be compared.

Returns:

Less than 0 if s1 s2, and 0 if s1 == s2.

rtp_wcsistr

Function:

Find a unicode string within a string.

Summary:

unsigned short* rtp_wcsistr (const unsigned short* str, const unsigned short* find_str)

Description:

Find find_str in str ignoring case.

Parameters:

str	-	Unicode string to search.
find_str	-	Unicode string to find.

Returns:

The address of the first place that find_str is found in str, 0 if not found.

rtp_wcsncmp

Function:

Compare characters of one unicode buffer with another.

Summary:

int rtp_wcsncmp (const unsigned short* s1, const unsigned short* s2, unsigned int n)

Description:

Compare a number of characters of s1 to a number of characters of s2.

Parameters:

s1	-	Pointer to unicode buffer to be compared.
s2	-	Pointer to unicodebuffer to be compared.
n	-	Number of characters to compare.

Returns:

Less than 0 if s1 s2, and 0 if s1 == s2.

rtp_wcsnicmp

Function:

Compare characters of one unicode buffer with another.

Summary:

int rtp_wcsnicmp (const unsigned short* s1, const unsigned short* s2, unsigned int n)

Description:

Compare a number of characters of s1 to a number of characters of s2, ignoring case.

Parameters:

s1	-	Pointer to unicode buffer to be compared.
s2	-	Pointer to buffer to be compared.
n	-	Number of characters to compare.

Returns:

Less than 0 if s1 s2, and 0 if s1 == s2.

rtp_wcscat

Function:

Copies the source string to the end of the destination string.

Summary:

unsigned short* rtp_wcscat (unsigned short* a, const unsigned short* b)

Description:

Appends string b to string a.

Parameters:

a	-	Destination buffer.
b	-	Source buffer.

Returns:

Value of a.

rtp_wcschr

Function:

Find a character within a string.

Summary:

unsigned short* rtp_wcschr (const unsigned short* str, int find_chr)

Description:

Find find_chr in str.

Parameters:

str	-	String to search.
find_chr	-	Character to find.

Returns:

The first place find_chr is found in str, 0 if not found.

rtp_wcscmp

Function:

Compare characters of one string with another.

Summary:

int rtp_wcscmp (const unsigned short* s1, const unsigned short* s2)

Description:

Compare characters of s1 to characters of s2.

Parameters:

s1	-	String to be compared.
s2	-	String to be compared.

Returns:

Less than 0 if s1 s2, and 0 if s1 == s2.

rtp_wcscpy

Function:

Copies the source string to the destination.

Summary:

unsigned short* rtp_wcscpy (unsigned short* a, const unsigned short* b)

Description:

Copies string b to string a.

Parameters:

a	-	Destination buffer.
b	-	Source buffer.

Returns:

Value of a.

rtp_wcslen

Function:

Finds the length of a string.

Summary:

unsigned int rtp_wcslen (const unsigned short* string)

Description:

Determines the lenght of string.

Parameters:

string	-	String to find the length of.

Returns:

Length of the string.

rtp_wcsncpy

Function:

Copies the source string to the destination.

Summary:

unsigned short* rtp_wcsncpy (unsigned short* a, const unsigned short* b, unsigned int n)

Description:

Copies n characters of string b to string a.

Parameters:

a	-	Destination buffer.
b	-	Source buffer.
n	-	Number of characters to copy from.

Returns:

Value of a.

rtp_wcsrchr

Function:

Find the last occurance of a character within a string.

Summary:

unsigned short* rtp_wcsrchr (const unsigned short* str, int find_chr)

Description:

Find last occurance of find_chr in str.

Parameters:

str	-	String to search.
find_chr	-	Character to find.

Returns:

The last occurance of find_chr found in str, 0 if not found.

rtp_wcsstr

Function:

Find a string within a string.

Summary:

unsigned short* rtp_wcsstr (const unsigned short* str, const unsigned short* find_str)

Description:

Find last occurance of find_chr in str.

Parameters:

str	-	String to search.
find_str	-	String to find.

Returns:

The last occurance of find_chr found in str, 0 if not found.

RTPWSCNV

rtp_wtoi

Function:

Convert a unicode string to an integer.

Summary:

int rtp_wtoi (const unsigned short* s)

Description:

Convert s to an integer.

Parameters:

s	-	Unicode string to be converted.

Returns:

Integer value of s.

rtp_wtol

Function:

Convert a unicode string to a long.

Summary:

long rtp_wtol (const unsigned short* s)

Description:

Convert s to a long.

Parameters:

s	-	String to be converted.

Returns:

Long value of s.

rtp_wcstol

Function:

Find a string within a string.

Summary:

long rtp_wcstol (const unsigned short* str, unsigned short** delimiter, int base)

Description:

Find last occurance of find_chr in str.

Parameters:

str	-	String to be converted.
delimiter	-	Character that stops the convertion.
base	-	Base of return value.

Returns:

The last occurance of find_chr found in str, 0 if not found.

rtp_wcstoul

Function:

Convert a string to an unsigned long.

Summary:

unsigned long rtp_wcstoul (const unsigned short* str, unsigned short** delimiter, int base)

Description:

Determine the long value of the passed string. If base is 0 then the base is determined by the format of the string; otherwise the base is assumed to be correct.

Parameters:

str	-	String to be converted.
delimiter	-	Character that stops the convertion.
base	-	Base of return value.

Returns:

Unsigned long representation of the passed string to the delimiter, 0 on error.

rtp_itow

Function:

Convert an integer to a unicode string.

Summary:

unsigned short* rtp_itow (int num, unsigned short* dest, int base)

Description:

Convert num to a unicode string using base and store it in dest.

Parameters:

num	-	Integer to convert.
dest	-	Destination unicode string.
base	-	Base of conversion.

Returns:

Value of dest.

rtp_ltow

Function:

Convert an long to a unicode string.

Summary:

unsigned short* rtp_ltow (long num, unsigned short* dest, int base)

Description:

Convert num to a uniocde string using base and store it in dest.

Parameters:

num	-	Long to convert.
dest	-	Destination unicode string.
base	-	Base of conversion.

Returns:

Value of dest.

rtp_ultow

Function:

Convert an unsigned long to a uniocde string.

Summary:

unsigned short* rtp_ultow (unsigned long num, unsigned short* dest, int base)

Description:

Convert num to a unicode string using base and store it in dest.

Parameters:

num	-	Unsigned long to convert.
dest	-	Destination unicode string.
base	-	Base of conversion.

Returns:

Value of dest.

CHAPTER 3

TARGET (PLATFORM-DEPENDENT) MODULES

Target (Platform-dependent) Modules

RTPDOBJ	-	Directory Object Services
RTPENV	-	Environment Variable Services
RTPFILE	-	Low Level File System Services
RTPIRQ	-	Interrupt Service Routine
RTPKERN	-	Kernel Services
RTPMEM	-	Dynamic Memory Services
RTPNET	-	Network Services
RTPSIGNL	-	Thread Synchronization Services
RTPSSL	-	Secure Socket Layer Services
RTPTERM	-	Terminal Services
RTPTHRD	-	Thread Services
RTPTIME	-	System Time Services
RTPWFILE	-	Wide Character Low Level File System Services

RTPDOBJ

DIRECTORY OBJECT SERVICES

rtp_file_gfirst	- Get the first occurrence of a file.
rtp_file_gnext	- Get the next occurrence of a file.
rtp_file_gdone	- Free the directory information originally allocated by rtp_file_gfirst.
rtp_file_get_size	- Get the size of a file.
rtp_file_get_attrib	- Retrieves the files associated attributes.
rtp_file_get_name	- Retrieves the name of the file.

These functions **MUST** be ported.

rtp_file_gfirst

Function:

Get the first occurrence of a file.

Summary:

int rtp_file_gfirst (void** dirobj, char* name)

Description:

Get the first occurrence of a file indicated by the name (which consists of both a path specifier and a search pattern), and store the related information in dirobj. dirobj should not be accessed and/or manipulated upon outside the RTPlatform API.

Parameters:

dirobj	- Storage location for file information found.
name	- The name of the file to find.

Returns:

0 if successful, -1 on error. For debugging purposes; if the cause of the error is obtainable at the native File System layer, turn on RTP_DISPLAY_ERROR in rtpfile.c to display the native error value.

Postconditions:

rtp_file_gdone **MUST** be called after making a successful call to rtp_file_gfirst to free any resources allocated.

rtp_file_gnext

Function:

Get the next occurrence of a file.

Summary:

int rtp_file_gnext (void* dirobj)

Description:

Get the next occurrence of a file indicated by the dirobj, and store the related information, overwriting the current information.

Parameters:

dirobj	-	File information to be altered.

Returns:

0 if successful, -1 on error. For debugging purposes; if the cause of the error is obtainable at the native File System layer, turn on RTP_DISPLAY_ERROR in rtpfile.c to display the native error value.

Preconditions:

MUST have called rtp_file_gfirst to obtain the dirobj before referencing this function.

rtp_file_gdone

Function:

Free the directory information originally allocated by rtp_file_gfirst.

Summary:

void rtp_file_gdone (void* dirobj)

Description:

Given a pointer the file information that was setup by a call to rtp_file_gfirst, free internal elements used by the dirobj.

Parameters:

dirobj	-	File information to be freed.

Returns:

0 if successful, -1 on error. For debugging purposes; if the cause of the error is obtainable at the native File System layer, turn on RTP_DISPLAY_ERROR in rtpfile.c to display the native error value.

Preconditions:

This function **MUST** be called after rtp_file_gfirst, and rtp_file_gnext are done being called so to avoid a memory leak.

rtp_file_get_size

Function:

Get the size of a file.

Summary:

int rtp_file_get_size (void* dirobj, unsigned long* size)

Description:

Determines the size of the file referred to by the dirobj handle returned by the call to rtp_file_gfirst and any successive calls to rtp_file_gnext.

Parameters:

dirobj	-	The dirobj handle returned byrtp_file_gfirst, and rtp_file_gnext.
size	-	The storage location for the acquired size.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native File System layer, turn on RTP_DISPLAY_ERROR in rtpfile.c to display the native error value.

rtp_file_get_attrib

Function:

Retrieves the files associated attributes.

Summary:

int rtp_file_get_attrib (void* dirobj, unsigned char* attributes)

Description:

Returns the attributes that have been assigned to the file. The attributes include:

RTP_FILE_ATTRIB_ISDIR	if it is a directory
RTP_FILE_ATTRIB_ISVOL	if it is a volume
RTP_FILE_ATTRIB_RDONLY	if it is read only
RTP_FILE_ATTRIB_WRONLY	if it is write only
RTP_FILE_ATTRIB_RDWR	if it has read and write permissions
RTP_FILE_ATTRIB_HIDDEN	if it is hidden
RTP_FILE_ATTRIB_SYSTEM	if it is a system file
RTP_FILE_ATTRIB_ARCHIVE	if it is archived

Parameters:

dirobj	-	The dirobj handle returned byrtp_file_gfirst, and rtp_file_gnext.
attributes	-	The storage location to return the attributes that have been assigned to the file.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native File System layer, turn on RTP_DISPLAY_ERROR in rtpfile.c to display the native error value.

rtp_file_get_name

Function:

Retrieves the name of the file.

Summary:

int rtp_file_get_name (void* dirobj, char* name, int size)

Description:

Returns the name of the file associated with the dirobj in the storage location. The native file system name will be truncated if the storage location is not large enough to contain it.

Parameters:

dirobj	- The dirobj handle returned by rtp_file_gfirst, and rtp_file_gnext.
name	- The storage location to return the name of the file.
size	- The size of the storage location.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native File System layer, turn on RTP_DISPLAY_ERROR in rtpfile.c to display the native error value.

RTPENV

ENVIRONMENT SERVICES

rtp_setenv	- Set an environment variable.
rtp_getenv	- Get an environment variable.

These functions **MUST** be ported.

rtp_setenv

Function:

Set an environment variable.

Summary:

int rtp_setenv (const char* envname, char* value)

Description:

Create an environment variable, modifiy an existing environment variable, or remove an environment variable. To remove a variable, pass 0 for its value.

Parameters:

envname	- Name of the environment variable.
value	- Value of the environment variable.

Returns:

0 if successful, -1 otherwise.

rtp_getenv

Function:

Get an environment variable.

Summary:

int rtp_getenv (const char* envname, char** value)

Description:

Get an environment variable with the name specified and store its value in the parameter passed in.

Parameters:

envname	-	Name of the environment variable.
value	-	Storage for environment variable value.

Returns:

0 if successful, -1 otherwise.

RTPFILE

LOW LEVEL FILE SYSTEM SERVICES

rtp_file_open	- Opens a file using the supplied name.
rtp_file_close	- Close a file.
rtp_file_read	- Read from a file.
rtp_file_write	- Write to a file.
rtp_file_lseek	- Moves the file pointer to the offset from the origin passed in.
rtp_file_truncate	- Truncate the file to the desired size.
rtp_file_flush	- Flushes a file stream.
rtp_file_rename	- Rename a file or directory.
rtp_file_delete	- Delete a file.
rtp_file_mkdir	- Create a new subdirectory.
rtp_file_rmdir	- Remove a subdirectory.
rtp_file_setcwd	- Set the current working directory.
rtp_file_pwd	- Print the name of the current working directory to a buffer.
rtp_file_get_time	- Retrieves the last access time, last write time, creation time, and last time the mode was changed.
rtp_file_chmode	- Change file mode.
rtp_file_get_free	- Retrieves disk free space information.
rtp_file_set_time	- Changes the date of creation, access, and modification of a file.

These functions **MUST** be ported.

rtp_file_open

Function:

Opens a file using the supplied name.

Summary:

int rtp_file_open (RTP_HANDLE* fdPtr, const char* name,
    unsigned short flag, unsigned short mode)

Description:

Opens a file using the flag and mode settings, and sets the file descriptor. The number of files available at any one time is dependant on the native File System. When use of the file handle is completed, rtp_file_close() should be called. Default mode is RTP_FILE_O_RDONLY if flag is 0.

Parameters:

fdPtr - Pointer to an RTP_HANDLE to store the file descriptor.

name - The name of the file to open.

flag - For the flag argument:

RTP_FILE_O_APPEND	All writes will be appended to the file.
RTP_FILE_O_RDONLY	Open a file for reading only.
RTP_FILE_O_WRONLY	Open a file for writing only.
RTP_FILE_O_RDWR	Open a file for reading and writing.
RTP_FILE_O_CREAT	Create a file if it does not exist.
RTP_FILE_O_TRUNC	Truncate a file to 0 bytes after opening.
RTP_FILE_O_EXCL	If creating a file, fail if it already exists
RTP_FILE_O_BINARY	Create the file in binary mode.
RTP_FILE_O_TEXT	Create the file in text mode.

Note: If neither the RTP_FILE_O_BINARY, or the RTP_FILE_O_TEXT are used, the default behavior is to open the file in text mode.

mode - For the mode argument:

RTP_FILE_S_IWRITE	Create a file with write permissions.
RTP_FILE_S_IREAD	Create a file with read permissions.

Returns:

0 if successful, -1 on failure, -2 if a non-fatal error occurred. If a -1 or -2 is returned the value of *fdPtr is undefined. For debugging purposes; if the cause of the error is obtainable at the native File System layer, turn on RTP_DISPLAY_ERROR in rtpfile.c to display the native error value.

rtp_file_close

Function:

Close a file.

Summary:

int rtp_file_close (RTP_HANDLE fileHandle)

Description:

Close a file that was opened with rtp_file_open() and update the disk by flushing the directory entry and file allocation table, then free all core associated with the file descriptor.

Parameters:

fileHandle	-	File descriptor used to close the file.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native File System layer, turn on RTP_DISPLAY_ERROR in rtpfile.c to display the native error value.

rtp_file_read

Function:

Read from a file.

Summary:

long rtp_file_read (RTP_HANDLE fileHandle, unsigned char* buffer, long count)

Description:

Attempt to read a number of bytes from the current file location of the file referred to by fileHandle and place the data in the storage buffer. The file location is updated accordingly.

Parameters:

fileHandle	-	File descriptor to read the file.
buffer	-	Buffer to store the read bytes.
count	-	Maximum amount to read.

Returns:

Number of bytes read, 0 if end of file, -1 on error, and -2 on a non-fatal error. For debugging purposes; if the cause of the error is obtainable at the native File System layer, turn on RTP_DISPLAY_ERROR in rtpfile.c to display the native error value.

rtp_file_write

Function:

Write to a file.

Summary:

long rtp_file_write (RTP_HANDLE fileHandle, const unsigned char* buffer, long count)

Description:

Attempt to write a number of bytes from a buffer to the current location in the file referred to by fileHandle. The file current location is updated accordingly.

Parameters:

fileHandle	- File descriptor to write the file.
buffer	- Buffer to write to the file.
count	- Amount to write.

Returns:

Number of bytes written, -1 on error, and -2 on a non-fatal error. For debugging purposes; if the cause of the error is obtainable at the native File System layer, turn on RTP_DISPLAY_ERROR in rtpfile.c to display the native error value.

rtp_file_lseek

Function:

Moves the file pointer to the offset from the origin passed in.

Summary:

long rtp_file_lseek (RTP_HANDLE fd, long offset, int origin)

Description:

Move the file pointer offset bytes in the file relative to the origin. Attempting to seek beyond the end of the file puts the file pointer one byte past the end of the file. Seeking zero bytes from the origin set to 2 (end of the file) returns the file length.

Parameters:

fd	- File descriptor to move its pointer.
offset	- The offset to move the pointer.
origin	- The location that the offset is relative to:
	0 to seek from the beginning of the file.
	1 to seek from the current file pointer position.
	2 to seek from the end of the file.

Returns:

Offset from the beginning of the file, -1 on error. For debugging purposes; if the cause of the error is obtainable at the native File System layer, turn on RTP_DISPLAY_ERROR in rtpfile.c to display the native error value.

rtp_file_truncate

Function:

Truncate the file to the desired size.

Summary:

int rtp_file_truncate (RTP_HANDLE fd, long offset)

Description:

Given a file handle and a new file size, either extend the file or truncate it. If the current file pointer is still within the range of the file, it is unmoved, otherwise it is moved to the end of the file. If the original size of the file is smaller than the offset, the file is padded with '\0' until the offset. If the original size is larger than the offset, the extra bytes are removed and lost.

Parameters:

fd	-	File descriptor to the file to truncate.
offset	-	The new size of the file.

Returns:

0 if successful, -1 on error. For debugging purposes; if the cause of the error is obtainable at the native File System layer, turn on RTP_DISPLAY_ERROR in rtpfile.c to display the native error value.

rtp_file_flush

Function:

Flushes a file stream.

Summary:

int rtp_file_flush (RTP_HANDLE fd)

Description:

Flushes the file stream of any data. After this call completes, the on disk view of the file is completely consistent with the in memory view. It is a good idea to call this function periodically if a file is being extended. If a file is not flushed or closed and a power down occurs, the file size will be wrong on disk and the FAT chains will be lost.

Parameters:

fd	-	File descriptor to the file to be flushed.

Returns:

0 if successful and when nothing to flush, -1 on error. For debugging purposes; if the cause of the error is obtainable at the native File System layer, turn on RTP_DISPLAY_ERROR in rtpfile.c to display the native error value.

rtp_file_rename

Function:

Rename a file or directory.

Summary:

int rtp_file_rename (char* name, char* newname)

Description:

Moves the file or subdirectory named name to the new name specified in newname. name and newname must be on the same drive but they may be in different directories. Both names must be fully qualified. Fails if newname is invalid, already exists or name is not found.

Parameters:

name	-	File name to be changed.
newname	-	New name to be used.

Returns:

0 if successful, -1 on error. For debugging purposes; if the cause of the error is obtainable at the native File System layer, turn on RTP_DISPLAY_ERROR in rtpfile.c to display the native error value.

rtp_file_delete

Function:

Delete a file.

Summary:

int rtp_file_delete (char* name)

Description:

Deletes a file. Fails if not a simple file, if it is open, does not exist, or is read only.

Parameters:

name	-	Name of file to be deleted.

Returns:

0 if successful, -1 on error. For debugging purposes; if the cause of the error is obtainable at the native File System layer, turn on RTP_DISPLAY_ERROR in rtpfile.c to display the native error value.

rtp_file_mkdir

Function:

Create a new subdirectory.

Summary:

int rtp_file_mkdir (char* name)

Description:f

Creates a subdirectory in the path specified by name. Fails if a file or directory of the same name already exists or if the directory component (if there is one) of the path is not found.

Parameters:

name	- Name of the new subdirectory.

Returns:

0 if successful, -1 on error. For debugging purposes; if the cause of the error is obtainable at the native File System layer, turn on RTP_DISPLAY_ERROR in rtpfile.c to display the native error value.

rtp_file_rmdir

Function:

Remove a subdirectory.

Summary:

int rtp_file_rmdir (char* name)

Description:

Remove a subdirectory specified by name. Fails if the path is not a directory, is read only, or is not empty.

Parameters:

name	- Name of the new subdirectory.

Returns:

0 if successful, -1 on error. For debugging purposes; if the cause of the error is obtainable at the native File System layer, turn on RTP_DISPLAY_ERROR in rtpfile.c to display the native error value.

rtp_file_setcwd

Function:

Set the current working directory.

Summary:

int rtp_file_setcwd (char* name)

Description:

Sets the directory specified by name as the new current working directory for this task. The name must be a valid directory name. If a drive letter is used in the name parameter, it will change the default drive letter to this new drive letter.

Parameters:

name	-	Name of the directory to become the current working directory.

Returns:

0 if successful, -1 on error. For debugging purposes; if the cause of the error is obtainable at the native File System layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_file_pwd

Function:

Print the name of the current working directory to a buffer.

Summary:

int rtp_file_pwd (char* name, long size)

Description:

Fill name with the full path name of the current working directory for the current task in the default drive. name must point to enough space to hold the full path without overriding user data. The worst case possible is native File System dependant.

Parameters:

name	-	Storage location for the name of the current working directory.
size	-	The size of the storage location.

Returns:

0 if successful, -1 on error. For debugging purposes; if the cause of the error is obtainable at the native File System layer, turn on RTP_DISPLAY_ERROR in rtpfile.c to display the native error value.

rtp_file_get_time

Function:

Retrieves the last access time, last write time, creation time, and last time the mode was changed.

Summary:

int rtp_file_get_time (void* dirobj, RTP_DATE* adate, RTP_DATE* wdate, RTP_DATE* cdate, RTP_DATE* hdate)

Description:

Gets the time of last access, last write, creation and modification using the dirobj returned by rtp_file_gfirst and any successive calls to rtp_file_gnext. If a time is not to be obtained, pass a NULL to indicate no storage location.

Parameters:

dirobj	- The dirobj handle returned by rtp_file_gfirst, and rtp_file_gnext.
adate	- The storage location for date of last access.
wdate	- The storage location for date of last write.
cdate	- The storage location for date of creation.
hdate	- The storage location for date of last file mode change.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native File System layer, turn on RTP_DISPLAY_ERROR in rtpfile.c to display the native error value.

rtp_file_chmode

Function:

Change file mode.

Summary:

int rtp_file_chmode (char* name, unsigned char attributes)

Description:

Change the mode of a file to the desired attributes. One or more of the attributes may be set by or'ing them together.

Parameters:

name	- The name of the file that needs its mode changed.
attributes	- The new attributes to be assigned to the file:

RTP_FILE_ATTRIB_RDONLY	read only
RTP_FILE_ATTRIB_WRONLY	write only
RTP_FILE_ATTRIB_RDWR	read and write permissions
RTP_FILE_ATTRIB_HIDDEN	hidden
RTP_FILE_ATTRIB_SYSTEM	system file
RTP_FILE_ATTRIB_ARCHIVE	archived

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native File System layer, turn on RTP_DISPLAY_ERROR in rtpfile.c to display the native error value.

rtp_file_get_free

Function:

Retrieves disk free space information.

Summary:

int rtp_file_get_free (char* name, unsigned long* total, unsigned long* free, unsigned long* sectors_per_unit, unsigned short* bytes_per_sector)

Description:

Given a drive name, return the number of bytes on the drive, the number of bytes free on the drive, the number of sectors per unit, and the number of bytes per sector.

Parameters:

name	-	The name of the drive to obtain information on.
total	-	Total number of bytes on the drive.
free	-	Total number of bytes the drive has free.
sectors_per_unit	-	Number of sectors per unit (cluster).
bytes_per_sector	-	Number of bytes per sector.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native File System layer, turn on RTP_DISPLAY_ERROR in rtpfile.c to display the native error value.

rtp_file_set_time

Function:

Changes the date of creation, access, and modification of a file.

Summary:

int rtp_file_set_time (RTP_HANDLE fd, RTP_DATE* adate, RTP_DATE* wdate, RTP_DATE* cdate, RTP_DATE* hdate)

Description:

Changes the date of creation, access, and modification of a file using the file descriptor passed in.

Parameters:

fd	-	File descriptor of file that needs its dates changed.
adate	-	Date of last access.
wdate	-	Date of last write.
cdate	-	Date of creation.
hdate	-	Date of last file mode change.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native File System layer, turn on RTP_DISPLAY_ERROR in rtpfile.c to display the native error value.

RTPKERN

KERNEL SERVICES

rtp_kern_init	- Initialize a kernel.
rtp_kern_run	- Begin any kernel services that need to be started explicitly.
rtp_kern_abort	- Abort a process.
rtp_kern_exit	- Exit an application.

These functions **MUST** be ported.

rtp_kern_init

Function:

Initialize a kernel.

Summary:

void rtp_kern_init (void)

Description:

Initialize a kernel.

Parameters:

Returns:

void

rtp_kern_run

Function:

Begin any kernel services that need to be started explicitly.

Summary:

void rtp_kern_run (void)

Description:

Begin any kernel services that need to be started explicitly.

Parameters:

Returns:

void

rtp_kern_abort

Function:

Abort a process.

Summary:

void rtp_kern_abort (void)

Description:

Abort from the process.

Returns:

void

rtp_kern_exit

Function:

Exit an application.

Summary:

void rtp_kern_exit (int exitvalue)

Description:

Exit an application. The exit value is 0 if successfully exiting, and -1 if exiting with a failure.

Parameters:

exitvalue	-	Exit value; 0 for success, -1 otherwise.

Returns:

void

RTPMEM

DYNAMIC MEMORY SERVICES

_rtp_malloc	-	Allocate memory.
_rtp_realloc	-	Reallocate memory.
_rtp_free	-	Free memory.

These functions **MUST** be ported.

_rtp_malloc

Function:

Allocate memory.

Summary:

void* _rtp_malloc (unsigned int size)

Description:

Allocates, if possible, a portion of memory of 'size' bytes.

Parameters:

size	-	Number of bytes to allocate.

Returns:

Pointer to the location of the allocated spcae, 0 otherwise. For debugging purposes; if the cause of the error is obtainable at the native Memory System layer, turn on RTP_DISPLAY_ERROR in rtpmem.c to display the native error value.

Preconditions:

Only use if dynamic memory is required. **MUST** not be called directly. Use the **rtp_malloc** macro defined in rtpmem.h and turn off RTP_TRACK_LOCAL_MEMORY.

_rtp_realloc

Function:

Reallocate memory.

Summary:

void* _rtp_realloc (void* ptr, unsigned int size)

Description:

Allocates, if possible, a portion of memory of 'size' bytes, then moves 'size' bytes, if possible, of the old region ptr was pointing to, to the newly allocated space.

Parameters:

ptr	-	Pointer to currently allocated space.
size	-	New number of bytes to allocate.

Returns:

Pointer to the location of the newly allocated spcae, 0 otherwise. For debugging purposes; if the cause of the error is obtainable at the native Memory System layer, turn on RTP_DISPLAY_ERROR in rtpmem.c to display the native error value.

Preconditions:

Only use if dynamic memory is required. MUST not be called directly. Use the **rtp_realloc** macro defined in rtpmem.h and turn off RTP_TRACK_LOCAL_MEMORY.

_rtp_free

Function:

Free memory.

Summary:

void _rtp_free (void* ptr)

Description:

Frees the portion of memory that ptr was allocated to by _rtp_malloc or _rtp_realloc.

Parameters:

ptr	-	Location that needs to be freed.

Returns:

void

Preconditions:

Only use if dynamic memory is required. MUST not be called directly. Use the **rtp_free** macro defined in rtpmem.h and turn off RTP_TRACK_LOCAL_MEMORY.

RTPNET

NETWORK SERVICES

rtp_net_init	- Initialize network API.
rtp_net_exit	- Close network API.
rtp_net_socket_stream	- Allocate stream type (TCP) socket.
rtp_net_socket_datagram	- Allocate datagram type (UDP) socket.
rtp_net_bind	- Bind a socket handle to a local port number and optionally an IP address.
rtp_net_listen	- Set up a socket handle for accepting connections.
rtp_net_getpeername	- Get port and IP address of the remote host that the socket handle is connected to.
rtp_net_getsockname	- Get port and IP address of the local host.
rtp_net_gethostbyname	- Get the network address for a host name.
rtp_net_accept	- Accept a stream-type (TCP) connection on a bound and listening socket.
rtp_net_connect	- Connect a socket to a remote host.
rtp_net_is_connected	- Determine if a socket is connected to a peer.
rtp_net_write_select	- Check if socket handle is ready for sending over or if it is connected.
rtp_net_read_select	- Check if socket handle is ready for receiving data or if it is ready to attempt an accept operation.
rtp_net_send	- Send data over a connection.
rtp_net_recv	- Receive data over a connection.
rtp_net_sendto	- Send a datagram.
rtp_net_recvfrom	- Receive a datagram.
rtp_net_closesocket	- Close the connection and free the socket.
rtp_net_shutdown	- Disable receives and/or sends on a socket.
rtp_net_getntoread	- Get the number of bytes available to read.
rtp_net_setblocking	- Set a socket to (non-)blocking mode.
rtp_net_setnagle	- Enable/disable the NAGLE flow control algorithm.
rtp_net_setlinger	- Set time to linger while closing a socket.
rtp_net_setreusesock	- Set the reuse socket option.
rtp_net_setreuseaddr	- Set the reuse address option.
rtp_net_settcpnocopy	- Set the tcp no copy option.
rtp_net_setkeepalive	- Set keep alive socket option.
rtp_net_setmembership	- Set multicast membership option.
rtp_net_setmcastttl	- Set multicast time to live option.
rtp_net_setbroadcast	- Set socket permission to allow/ disallow broadcasts.
rtp_net_htons	- Convert a short integer from host to network format.
rtp_net_ntohs	- Convert a short integer from network to host format.
rtp_net_htonl	- Convert a long integer from host to network format.
rtp_net_ntohl	- Convert a long integer from network to host format.
rtp_net_ip_to_str	- Print an IP address to a dotted decimal string.
rtp_net_str_to_ip	- Extract an IP address from a dotted decimal string.
rtp_fd_zero	- Reset the socket handle list.
rtp_fd_set	- Add a socket handle to a list.
rtp_fd_clr	- Remove a socket handle from a list.
rtp_fd_isset	- Check if a socket handle is in a list.
rtp_net_select	- Select ready sockets.

These functions **MUST** be ported.

rtp_net_init

Function:

Initialize network API.

Summary:

int rtp_net_init (void)

Description:

Initialize network API.

Returns:

0 on success, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native Socket layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

Function:

rtp_net_exit

Function:

Close network API.

Summary:

void rtp_net_exit (void)

Description:

Close network API.

Returns:

void

rtp_net_socket_stream

Function:

Allocate stream type (TCP) socket.

Summary:

int rtp_net_socket_stream (RTP_HANDLE* sockHandle)

Description:

Allocates, if possible, a stream (TCP, connection-based) type socket. The handle then may later be used as an argument to the other RTPlatform Socket API functions. The number of sockets available at any one time is dependant on the native TCP/IP stack. When use of the socket handle is completed, rtp_net_closesocket() should be called to make it available for other calls to rtp_net_socket_stream(). Like the UNIX sockets environment, the TCP socket handle is not useful until rtp_net_connect(), or rtp_net_listen(), rtp_net_bind() and rtp_net_accept() has been called.

Parameters:

sockHandle	- Address of a socket handle to store the allocated socket.

Returns:

0 if successful, -1 otherwise. If a -1 is returned the value of *sockeHandle is undefined. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_socket_datagram

Function:

Allocate datagram type (UDP) socket.

Summary:

int rtp_net_socket_datagram (RTP_HANDLE* sockHandle)

Description:

Allocates, if possible, a datagram (UDP, connectionless) type socket. The handle then may later be used as an argument to the other RTPlatform Socket API functions. The number of sockets available at any one time is dependant on the native TCP/IP stack. When use of the socket handle is completed, rtp_net_closesocket() should be called to make it available for other calls to rtp_net_socket_datagram().

Parameters:

sockHandle	- Address of a socket handle to store the allocated socket.

Returns:

0 if successful, -1 otherwise. If a -1 is returned the value of *sockeHandle is undefined. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_bind

Bind a socket handle to a local port number and optionally an IP address.

Summary:

int rtp_net_bind (RTP_HANDLE sockHandle, unsigned char* ipAddr, int port, int type)

Description:

This function should be called on a socket before calling rtp_net_listen/rtp_net_accept (if the socket is stream-type), or rtp_net_recvfrom (if the socket is datagram-type). The behavior of this function is undefined if sockHandle is not a valid socket handle. If the IP address is not passed in, the default (typically INADDR_ANY) is used. If the default IP address is used, packets sent to any of the devices interfaces are accepted. If rtp_net_bind is called with 0 for its port number, the native TCP/IP stack should assign a unique port number. If the port specified is already in use by another socket handle, this function must return -1.

Server applications will call rtp_net_bind() to establish a t_bind does not establish any connections itself.

For datagram-type sockets, in order to receive broadcast and multicast packets, the socket handle should be bound to the defualt (INADDR_ANY) IP address or to the broadcast or the specific multicast address.

It is not necessary to call rtp_net_bind if rtp_net_connect or rtp_net_sendto is to be called.

Parameters:

sockHandle	-	Socket handle to bind to.
ipAddr	-	The optional IP address to be attached to.
port	-	Port (in host byte order) to be attached to.
type	-	Type of IP address:
		4 Indicates an IPv4 address.
		6 Indicates an IPv6 address.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_listen

Function:

Set up a socket handle for accepting connections.

Summary:

int rtp_net_listen (RTP_HANDLE sockHandle, int queueSize)

Description:

This function is called on a stream-type socket after rtp_net_bind() but before rtp_net_accept() to get a socket ready to accept connections from remote hosts. The behavior of this function is undefined if sockHandle is not a valid socket handle.

After calling rtp_net_listen(), server applications call rtp_net_accept() to establish a session with a client and then _net_listen() call allows a backlog of client connection requests to be queued up for processing by the server. Without rtp_net_listen(), the client connection requests will be rejected.

Parameters:

sockHandle	-	Pointer to a socket handle to listen on.
queueSize	-	The max number of requested connections to queue for acceptance.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_getpeername

Function:

Get port and IP address of the remote host that the socket handle is connected to.

Summary:

int rtp_net_getpeername (RTP_HANDLE sockHandle, unsigned char* ipAddr, int* port, int* type)

Description:

This function extracts the IP address and port number of the host connected to the socket and stores them in the arguments passed to the function. The socket handle must be connected. The ipAddr must have enough room to store the worst case scenario (If IPv6 is enbled, a 16 byte or unsigned char array containing 16 elements must be used).

Parameters:

sockHandle	-	Socket handle connected to the remote host.
ipAddr	-	Location to store the IP address in host byte order.
port	-	Location to store the port in host byte order.
type	-	Location to store the type of IP address: 4 Indicates an IPv4 address. 6 Indicates an IPv6 address.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_getsockname

Function:

Get port and IP address of the local host.

Summary:

int rtp_net_getsockname (RTP_HANDLE sockHandle, unsigned char* ipAddr, int* port, int* type)

Description:

This function extracts the IP address and port number of the local host of the socket handle and stores them in the arguments passed to the function. The ipAddr must have enough room to store the worst case scenario (If IPv6 is enbled, a 16 byte or unsigned char array containing 16 elements must be used).

Parameters:

sockHandle	-	Socket handle connected to the remote host.
ipAddr	-	Location to store the IP address in host byte order.
port	-	Location to store the port in host byte order.
type	-	Location to store the type of IP address: 4 Indicates an IPv4 address. 6 Indicates an IPv6 address.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_gethostbyname

Function:

Get the network address for a host name.

Summary:

int rtp_net_gethostbyname (unsigned char* ipAddr, int* type,
    char* name)

Description:

This function is called to do a DNS lookup on the string The ipAddr must have enough room to store the worst case scenario (If IPv6 is enbled, a 16 byte or unsigned char array containing 16 elements must be used). Note that DNS uses the UDP protocol to retrieve the information from the DNS server.

Parameters:

ipAddr	-	Location to store the IP address in host byte order.
type	-	Location to store the type of IP address: 4 Indicates an IPv4 address. 6 Indicates an IPv6 address.
name	-	Wtring representation of the IP address.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_accept

Function:

Accept a stream-type (TCP) connection on a bound and listening socket.

Summary:

int rtp_net_accept (RTP_HANDLE* connectSock,
    RTP_HANDLE serverSock, unsigned char* ipAddr, int*
    port, int* type)

Description:

This function can block waiting for a remote host to connect to the port/IP address to which serverSock is bound. If a connection is successfully established, it must set connectSock to the socket for the new connection (serverSock continues to listen on its port/IP address), and returns 0. In the event of a successful connection, ipAddr, port, and type should also be set to the port and IP address of the remote host that is connected. If no connection can be established or an error occurs, the values of the arguments are undefined. The ipAddr must have enough room to store the worst case scenario (If IPv6 is enbled, a 16 byte or unsigned char array containing 16 elements must be used).

rtp_net_accept returns a new socket handle to be used for the connection but the original socket handle is still allocated and needs to be closed when no more rtp_net_accept on the original socket will be done. The socket handle returned by rtp_net_accept also needs to be closed when it is no longer needed.

The behavior of this function is undefined in the following cases:

- serverSock is not a valid, stream-type socket.
- rtp_net_bind was never called on serverSock.
- rtp_net_listen was never called on serverSock.

Parameters:

connectSock	-	Location to store the socket handle of the new connection.
serverSock	-	The socket handle to accept the connection on.
ipAddr	-	Location to store the IP address in host byte order.
port	-	Location to store the port in host byte order.
type	-	Location to store the type of IP address: 4 Indicates an IPv4 address. 6 Indicates an IPv6 address.

Returns:

0 if successful, -1 on failure, -2 if a non-fatal error occurred. A non-fatal error includes a socket that is set for non-blocking mode and the native accept call returns immediately with an error indicating

that it would block or that the operation is in progress. If the application making use of rtp_net_accept is using non-blocking sockets, it is advised that the return value is checked for -2 and continues, using rtp_net_read/write_select to determine if the connection has been established before making other RTPlatform Socket API calls on this socket. If the sockets are blocking, the rtp_net_accept will wait infinitely for a connection. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_connect

Function:

Connect a socket to a remote host.

Summary:

int rtp_net_connect (RTP_HANDLE sockHandle, unsigned char* ipAddr, int port, int type)

Description:

rtp_net_connect establishes a connection between a socket and the service provider at the IP address and port number provided in the arguments. If the socket is unbound, it will be bound with a unique port number and the local connecting IP address for the interface. Behavior is undefined if sockHandle is not a valid, socket handle. For stream-type (TCP) sockets, rtp_net_connect initiates the handshake. For connectionless (UDP) sockets, the IP address and port number are bound to the socket and the function returns immediately.

Parameters:

sockHandle	- The socket handle to make the connection on.
ipAddr	- The IP address in host byte order to connect to.
port	- The port in host byte order to connect to.
type	- The type of IP address to connect to: 4 Indicates an IPv4 address. 6 Indicates an IPv6 address.

Returns:

0 if successful, -1 on failure, -2 if a non-fatal error occurred. A non-fatal error includes a socket that is set for non-blocking mode and the native connect call returns immediately with an error indicating that it would block or that the operation is in progress. If the application making use of rtp_net_connect is using non-blocking sockets, it is advised that the return value is checked for -2 and continues, using rtp_net_read/write_select to determine if the connection has been established before making other RTPlatform Socket API calls on this socket. If the sockets are blocking, the rtp_net_connect will wait infinitely for a connection. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_is_connected

Function:

Determine if a socket is connected to a peer.

Summary:

unsigned rtp_net_is_connected (RTP_SOCKET sockHandle)

Description:

rtp_net_is_connected determines if a connection has been made over the indicated socket.

Parameters:

sockHandle	-	The socket handle to determine connected status.

Returns:

1 if successful, 0 on failure. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_write_select

Function:

Check if socket handle is ready for sending over or if it is connected.

Summary:

int rtp_net_write_select (RTP_HANDLE sockHandle, long msecTimeout)

Description:

This function is called to determine if a socket handle is ready for sends or if the socket has completed a connect operation all before the timeout period has expired. The associated socket should be set to non-blocking before calling rtp_net_connect, rtp_net_accept, rtp_net_recv, and rtp_net_send to ensure the function will not block.

Parameters:

sockHandle	-	The socket handle to check its state.
msecTimeout	-	Time period to wait in msec for the desired action:

0	To poll
-1	To indicate use of infinite.

Returns:

0 if successful, -1 on timeout and failure. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_read_select

Function:

Check if socket handle is ready for receiving data or if it is ready to attempt an accept operation.

Summary:

int rtp_net_read_select (RTP_HANDLE sockHandle, long msecTimeout)

Description:

This function is called to determine if a socket handle is ready for receives or if it is ready for an accept operation all before the timeout period has expired. The associated socket should be set to non-blocking before calling rtp_net_connect, rtp_net_accept, rtp_net_recv, and rtp_net_send to ensure the function will not block.

Parameters:

sockHandle	-	The socket handle to check its state.
msecTimeout	-	Time period to wait in msec for the desired action: 0 To poll. -1 To indicate use of infinite.

Returns:

0 if successful, -1 on timeout and failure. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_send

Function:

Send data over a connection.

Summary:

long rtp_net_send (RTP_HANDLE sockHandle, const unsigned char* buffer, long size)

Description:

This function is used to send data over a connected socket. The socket must be a stream-type (TCP) socket; behavior is undefined if sockHandle is not a valid stream-type socket.

If the socket is in blocking mode, it returns after all the data has been sent and acknowledged by the remote host. If the socket is in non-blocking mode, as much data as possible is queued in the output window and as much data as possible is sent according to the remote hosts window size and the mss. The function then returns. If there is no room in the output window, the function returns an error that may be non-fatal. If the socket is in non-blocking mode, select may be called to wait for any room in the output window.

In non-blocking mode, successful completion of the rtp_net_send() indicates that all data is queued in the output window. Call rtp_net_write_select to receive notification when the data was sent and acknowledged.

In blocking mode, successful completion of the rtp_net_send indicates that all data has been received by the remote host.

Parameters:

sockHandle	-	The socket handle to send over.
buffer	-	Pointer to the data to be sent.
size	-	The number of unsigned chars (bytes) to send.

Returns:

The number of unsigned chars (bytes) sent if successful, -1 on error, -2 if a non-fatal error occurred. A non-fatal error includes a socket that is set for non-blocking mode and the native send operation returns immediately with an error indicating that it would block or that the operation is in progress. If the application making use of rtp_net_send is using non-blocking sockets, it is advised that the return value is checked for -2 and continues, using rtp_net_write_select to determine if the send operation has been completed before making other RTPlatform Socket API calls on this socket. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_recv

Function:

Receive data over a connection.

Summary:

long rtp_net_recv (RTP_HANDLE sockHandle, unsigned char* buffer, long size)

Description:

This function is used to receive data over a connected socket. The socket must be a stream-type (TCP) socket; behavior is undefined if sockHandle is not a valid stream-type socket.

If the socket is in blocking mode, it returns after ze has been received. If no data is available and the socket is in blocking mode, rtp_net_recv blocks on the read signal until any data is available. If no data is available and the socket is in non-blocking mode, rtp_net_recv returns immediately with a -2. rtp_net_read_select may be called prior to calling rtp_net_recv to determine when data is available.

Parameters:

sockHandle	-	The socket handle to receive over.
buffer	-	Pointer to the storage location for the data received.
size	-	The maximum number of unsigned chars (bytes) to receive.

Returns:

The number of unsigned chars (bytes) received if successful, -1 on error, -2 if a non-fatal error occurred. A non-fatal error includes a socket that is set for non-blocking mode and the native receive operation returns immediately with an error indicating that it would block or that the operation is in progress. If the application making use of rtp_net_recv is using non-blocking sockets, it is advised that the return value is checked for -2 and continues, using rtp_net_read_select to determine if the rtp_net_recv should be tried again before making other RTPlatform Socket API calls on this socket. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_sendto

Function:

Send a datagram.

Summary:

long rtp_net_sendto (RTP_HANDLE sockHandle, const unsigned char* buffer, long size, unsigned char* ipAddr, int port, int type)

Description:

This function is used to send data over a connectionless socket (UDP). The socket must be a connectionless (UDP) socket; behavior is undefined if sockHandle is not a valid connectionless (UDP) socket.

Parameters:

sockHandle	-	The socket handle to send over.
buffer	-	Pointer to the data to be sent.
size	-	The number of unsigned chars (bytes) to send.
ipAddr	-	The IP address in host byte order to send to.
port	-	The port in host byte order to send to.
type	-	The type of IP address to send to:
		4 Indicates an IPv4 address.
		6 Indicates an IPv6 address.

Returns:

The number of unsigned chars (bytes) sent if successful, -1 on error, -2 if a non-fatal error occurred. A non-fatal error includes a socket that is set for non-blocking mode and the native sendto operation returns immediately with an error indicating that it would block or that the operation is in progress. If the application making use of rtp_net_sendto is using non-blocking sockets, it is advised that the return value is checked for -2 and continues, using rtp_net_write_select to determine if the sendto operation has been completed before making other RTPlatform Socket API calls on this socket. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_recvfrom

Function:

Receive a datagram.

Summary:

long rtp_net_recvfrom (RTP_HANDLE sockHandle, unsigned char* buffer, long size, unsigned char* ipAddr, int* port, int* type)

Description:

This function is used to receive data over a connectionless socket (UDP). The socket must be a connectionless (UDP) socket; behavior is undefined if sockHandle is not a valid connectionless (UDP) socket.

If the socket is in blocking mode, it returns after zeceful close causing a return of 0. If no data is available and the socket is in blocking mode, rtp_net_recvfrom blocks on the read signal until any data is available. If no data is available and the socket is in non-blocking mode, rtp_net_recvfrom returns immediately with a -2. rtp_net_read_select may be called prior to calling rtp_net_recvfrom to determine when data is available. If the datagram is smaller than the f unsigned chars (bytes), then only the number of unsigned chars in the datagram must be read into the buffer, and the function must return.

Parameters:

sockHandle	-	The socket handle to receive over.
buffer	-	Location to store the data received.
size	-	The number of unsigned chars (bytes) to receive.
ipAddr	-	(optional) Location to store the returned IP address.
port	-	(optional) Location to store the returned port in host byte order.
type	-	(optional) Location to store the returned type of IP address:

4 Indicates an IPv4 address.
6 Indicates an IPv6 address.

Returns:

The number of unsigned chars (bytes) received if successful, -1 on error, -2 if a non-fatal error occurred. A non-fatal error includes a socket that is set for non-blocking mode and the native datagram receive operation returns immediately with an error indicating that it would block or that the operation is in progress. If the application making use of rtp_net_recvfrom is using non-blocking sockets, it is advised that the return value is checked for -2 and continues, using rtp_net_read_select to determine if the rtp_net_recv should be tried again before making other RTPlatform Socket API calls on this socket. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_closesocket

Function:

Close the connection and free the socket.

Summary:

int rtp_net_closesocket (RTP_HANDLE sockHandle)

Description:

This function is called to release a socket and shutdown any open connections it has. For stream-type (TCP) sockets, this routine starts a shutdown of a connection. Once handshaking is complete and all the input data queued on the socket has been read, all resources associated with the connection are freed. For connectionless (UDP) sockets, the connection is dropped immediately.

This may be used with rtp_net_setlinger() to obtain the following affects:

TCP CLOSE SOCKET SUMMARY

Type	Linger	Linger Block	Will Block	Data Lost
hard	on	zero	no	possibly
graceful	on	non-zero	yes	possibly
graceful	off	zero	no	no
graceful	off	non-zero	no	possibly*

*Note: Not BSD or Winsock standard.

Parameters:

sockHandle	-	Pointer to a socket handle of the socket to close.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_shutdown

Function:

Disable receives and/or sends on a socket.

Summary:

int rtp_net_shutdown (RTP_HANDLE sockHandle, int how)

Description:

This function is called to disable receive and/or send operations depending upon the tream-type (TCP) sockets, if receives are disabled, incoming data will be accepted until the input window is full but the data will not be acknowledged. If sends are disabled, a FIN will be sent. For connectionless (UDP) sockets, if receives are disabled, incoming packets are queued.

Parameters:

sockHandle	-	Socket handle of the socket to manipulate.
how	-	Specifies which operations to shutdown:

0 Disable recv.

1 Disable send.

2 Disable both recv and send.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_getntoread

Function:

Get the number of bytes available to read.

Summary:

int rtp_net_getntoread (RTP_HANDLE sockHandle, unsigned long* nToRead)

Description:

For stream-type sockets (TCP), the total number of bytes in the input window is returned. For connectionless sockets (UDP), the total number of bytes in the first packet queued on the socket is returned. The number of bytes available is stored in nToRead.

Parameters:

sockHandle	-	Socket handle of the socket to access.
nToRead	-	Storage location of the returned number to read.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_setblocking

Function:

Set a socket to (non-)blocking mode.

Summary:

int rtp_net_setblocking (RTP_HANDLE sockHandle, unsigned int onBool)

Description:

This function sets a socket to blocking or non-blocking mode depending on the t be a valid socket handle.

Parameters:

sockHandle	-	Socket handle of the socket to manipulate.
onBool	-	1 to set blocking, 0 to set non-blocking.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_setnagle

Function:

Enable/disable the NAGLE flow control algorithm.

Summary:

int rtp_net_setnagle (RTP_HANDLE sockHandle, unsigned int onBool)

Description:

This function sets a socket to use the Nagle algorithm or disable the Nagle algorithm depending on the lThe Nagle algorithm prohibits sending of small stream-type (TCP) packets (less than the MSS) while there is any outstanding output data which has not been acknowledged.

Parameters:

sockHandle	-	Socket handle of the socket to manipulate.
onBool	-	1 to turn on nagle, 0 to turn of nagle.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_setlinger

Function:

Set time to linger while closing a socket.

Summary:

int rtp_net_setlinger (RTP_HANDLE sockHandle, unsigned int onBool, long msecTimeout)

Description:

This function sets the length of time the socket lingers before closing. A timeout of 0 causes a hard close (immediately closing the socket). After closing the socket, if the output window is able to become empty, a FIN is sent. If the timeout is reached before this is possible, a RESET is sent and the data still in the output window may be lost.

TCP CLOSE SOCKET SUMMARY

Type	Linger	Linger Block	Will Block	Data Lost
hard	on	zero	no	possibly
graceful	on	non-zero	yes	possibly
graceful	off	zero	no	no
graceful	off	non-zero	no	possibly*

Note: Not BSD or Winsock standard.

Parameters:

sockHandle	-	Socket handle of the socket to manipulate.
onBool	-	Boolean to turn on/off linger.
msecTimeout	-	Timeout period in msec to linger before hard closing.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_setreuseaddr

Function:

Set the reuse address option.

Summary:

int rtp_net_setreuseaddr (RTP_HANDLE sockHandle, unsigned int onBool)

Description:

This function lets the socket to use the same address (is already being used by another socket of the same protocol type in the system.

Parameters:

sockHandle	-	Socket handle of the socket to manipulate.
onBool	-	1 reuse address, 0 don’t reuse address.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_settcpnocopy

Function:

Set the tcp no copy option.

Summary:

int rtp_net_settcpnocopy (RTP_HANDLE sockHandle, unsigned int onBool)

Description:

If no copy mode is set it causes the native TCP/IP layer to not copy any data as it is placed into the input or output windows.

Parameters:

sockHandle	-	Socket handle of the socket to manipulate.
onBool	-	1 no copy, 0 copy.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_setkeepalive

Function:

Set keep alive socket option.

Summary:

int rtp_net_setkeepalive (RTP_HANDLE sockHandle, unsigned int onBool)

Description:

If keep alive is enabled, stream-type (TCP) keep alive packets are sent to the connected host socket if no packet has been received from that host after a period of time. The keep alives are retried until a packet is received or a timeout occurs. If a tiemout occurs, the socket is set to the closed state, i.e. no close handshake is attempted. If the socket is closed, the resources will not be freed until rtp_net_closesocket is called.

Parameters:

sockHandle	-	Socket handle of the socket to manipulate.
onBool	-	1 keep alive, 0 don’t keep alive.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_setmembership

Function:

Set multicast membership option.

Summary:

int rtp_net_setmembership (RTP_HANDLE sockHandle, unsigned char* ipAddr, int port, int type, unsigned int onBool)

Description:

Add or remove membership to a multicast address.

Parameters:

sockHandle	- Socket handle of the socket to manipulate.
ipAddr	- The address to add or remove mutlicast membership on.
port	- The port that mulicast membership will be applied on.
type	- The type of ip address that is passed in: 4 Indicates an IPv4 address. 6 Indicates an IPv6 address.
onBool	- 1 add membership, 0 remove membership.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_setmcastttl

Function:

Set multicast time to live option.

Summary:

int rtp_net_setmcastttl (RTP_HANDLE sockHandle, int ttl)

Description:

Sets the TTL value associated with IP multicast traffic on the socket.

Parameters:

sockHandle	- Socket handle of the socket to manipulate.
ttl	- Time to live value.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_setbroadcast

Function:

Set socket permission to allow/disallow broadcasts.

Summary:

int rtp_net_setbroadcast (RTP_SOCKET sockHandle, unsigned int onBool)

Description:

Set socket permission to allow/disallow broadcasts over this socket.

Parameters:

sockHandle	-	Socket handle of the socket to manipulate.
onBool	-	1 to permit broadcasts, 0 to disallow

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

rtp_net_htons

Function:

Convert a short integer from host to network format.

Summary:

short rtp_net_htons (short i)

Description:

On a little endian system the short will be flipped.

Parameters:

i	-	The short integer to be converted.

Returns:

The converted short int.

rtp_net_ntohs

Function:

Convert a short integer from network to host format.

Summary:

short rtp_net_ntohs (short i)

Description:

On a little endian system the short will be flipped.

Parameters:

i	-	The short integer to be converted.

Returns:

The converted short int.

rtp_net_htonl

Function:

Convert a long integer from host to network format.

Summary:

long rtp_net_htonl (long i)

Description:

On a little endian system the long will be flipped.

Parameters:

i	-	The long to be converted.

Returns:

The converted long int.

rtp_net_ntohl

Function:

Convert a long integer from network to host format.

Summary:

long rtp_net_ntohl (long i)

Description:

On a little endian system the long will be flipped.

Parameters:

i	- The long to be converted.

Returns:

The converted long int.

rtp_net_ip_to_str

Function:

Print an IP address to a dotted decimal string.

Summary:

int rtp_net_ip_to_str (char* str, unsigned char* ipAddr, int type)

Description:

Extracts an IP address and creates a dotted decimal string representation. The contain the IP address. If it is not, the resulting operation is undefined.

{111,111,111,111}	converts to "111.111.111.111"
{111,111,111,0}	converts to "111.111.111.0"
{111,111,0,0}	converts to "111.111.0.0"
{111,0,0,0}	converts to "111.0.0.0"

Parameters:

str	- The storage location for the dotted string representation.
ipAddr	- The IP address to be extracted.
type	- Type of IP address:

4 Indicates an IPv4 address.

6 Indicates an IPv6 address.

Returns:

The length of the string representation, -1 on failure.

rtp_net_str_to_ip

Function:

Extract an IP address from a dotted decimal string.

Summary:

int rtp_net_str_to_ip (unsigned char* ipAddr, char* str, int* type)

Description:

Extracts an IP address from a string and stores it in ipAddr. The ipAddr must be large enough to contain the IP address. If it is not, the resulting operation is undefined. For an IPv4 address, this must be at least a four element unsigned char array. For an IPv6 address, this must be at least a six element unsigned char array.

“111.111.111.111”	converts to {111,111,111,111}
“111.111.111”	converts to {111,111,111,0}
“111.111”	converts to {111,111,0,0}
“111”	converts to {111,0,0,0}

Parameters:

ipAddr	- The storage location for the IP address to be extracted.
str	- The dotted decimal string representation.
type	- Type of IP address:
	4 Indicates an IPv4 address.
	6 Indicates an IPv6 address.

Returns:

0 on success, -1 on error.

rtp_fd_zero

Function:

Reset the socket handle list.

Summary:

void rtp_fd_zero (RTP_FD_SET* list)

Description:

Resets the socket handle list that is used by rtp_net_select.

Parameters:

list	- The list to reset.

Returns:

void

Preconditions:

Support for this functionality is not needed unless rtp_net_select is to be used.

rtp_fd_set

Function:

Add a socket handle to a list.

Summary:

void rtp_fd_set (RTP_FD_SET* list, RTP_HANDLE fd)

Description:

Adds a socket handle to the list that is used by rtp_net_select.

Parameters:

list	-	The list to add to.
fd	-	The handle to add to the list.

Returns:

void

Preconditions:

Support for this functionality is not needed unless rtp_net_select is to be used.

rtp_fd_clr

Function:

Remove a socket handle from a list.

Summary:

void rtp_fd_clr (RTP_FD_SET* list, RTP_HANDLE fd)

Description:

Removes a socket handle from a list that is used by rtp_net_select.

Parameters:

list	-	The list to remove from.
fd	-	The handle to remove from the list.

Returns:

void

Preconditions:

Support for this functionality is not needed unless rtp_net_select is to be used.

rtp_fd_isset

Function:

Check if a socket handle is in a list.

Summary:

int rtp_fd_isset (RTP_FD_SET* list, RTP_HANDLE fd)

Description:

Checks if a socket handle is in a list that is used by rtp_net_select.

Parameters:

list	- The list to check.
fd	- The handle to find.

Returns:

1 if found, 0 on failure.

Preconditions:

Support for this functionality is not needed unless rtp_net_select is to be used.

rtp_net_select

Function:

Select ready sockets.

Summary:

int rtp_net_select (RTP_FD_SET* readList, RTP_FD_SET* writeList, RTP_FD_SET* errorList, long msecTimeout)

Description:

This function is called to obtain the number of sockets that are contained in the RTP_FD_SET lists that meet the condition before the timeout period is up. This function only needs to be implemented if selecting on multiple sockets at a time and/or use of the errorList is desired (See also: rtp_net_read_select, rtp_net_write_select).

rtp_net_select provides a timeout capability for rtp_net_connect, rtp_net_accept, rtp_net_recv, rtp_net_recvfrom, rtp_net_send, and rtp_net_sendto. Each of these functions block infinitely if the associated socket is in blocking mode or they return immediately if the socket is in non-blocking mode (rtp_net_setblocking). rtp_net_select takes three socket lists and a timeout value as inputs. It blocks the select signal until either the timeout period expires or the select criteria, as specified by the lists, is met for at least one socket.

If readList is setup, this will return when any socket in the readList has data. For stream-type (TCP) sockets are listening (rtp_net_listen has been called and -2 has returned on the non-blocking socket handle) rtp_net_select will return when a connection has been established and rtp_net_accept will succeed without blocking.

If the writeList is setup, this will return when any non-blocking stream-type (TCP) socket in the writeList has room in the output window. It will also return when a blocking stream-type (TCP) socket has an empty output window. If there are any connectionless (UDP) sockets on the list, it will return immediately.

If an error occurs on the a socket associated with the errorList, rtp_net_select returns.

The lists will be updated to contain only those socket handles that have passed the above criteria.

Parameters:

readList	- (optional) A list of sockets to check for read access.
writeList	- (optional) A list of sockets to check for write access.
errorList	- (optional) A list of sockets to check for errors.
msecTimeout	- Time period to wait in msec for the desired action: 0 To poll. -1 To indicate use of infinite.

Returns:

Number of sockets meeting the criteria on success, 0 if timed out, and -1 on failure. For debugging purposes; if the cause of the error is obtainable at the native TCP/IP layer, turn on RTP_DISPLAY_ERROR in rtpnet.c to display the native error value.

Preconditions:

Support for this functionality is not needed unless full functions of rtp_net_select is needed.

See Also:

rtp_net_read_select

rtp_net_write_select

RTPSIGNL

THREAD SYNCHRONIZATION SERVICES

rtp_sig_semaphore_alloc	- Create and itialize a semaphore.
rtp_sig_semaphore_free	- Free a semaphore.
rtp_sig_semaphore_wait_timed	- Wait for a semaphore to be signaled.
rtp_sig_semaphore_clear	- Clear the state of a semaphore.
rtp_sig_semaphore_wait	- Wait for a semaphore to be signaled.
rtp_sig_semaphore_signal	- Signal a semaphore.
rtp_sig_semaphore_signal_isr	- Signal an interrupt service routine semaphore.
rtp_sig_mutex_alloc	- Create and itialize a mutex.
rtp_sig_mutex_free	- Free a mutex.
rtp_sig_mutex_claim_timed	- Wait for a mutex to be available.
rtp_sig_mutex_claim	- Wait for a mutex to be available.
rtp_sig_mutex_release	- Release a mutex.

These functions **MUST** be ported.

rtp_sig_semaphore_alloc

Function:

Create and itialize a semaphore.

Summary:

int rtp_sig_semaphore_alloc (RTP_HANDLE* newSem, const char* name)

Description:

Create and initalize a semaphore in the non-signalled state. For example a counting semaphore with an initial count of zero.

Parameters:

newSem	- Storage location for the handle of the newly created semaphore.
name	- The name of the semaphore.

Returns:

0 if successful, -1 otherwise. If a -1 is returned the value of *newSem is undefined. For debugging purposes; if the cause of the error is obtainable at the native Kernel layer, turn on RTP_DISPLAY_ERROR in rtpsignl.c to display the native error value.

rtp_sig_semaphore_free

Function:

Free a semaphore.

Summary:

void rtp_sig_semaphore_free (RTP_HANDLE semHandle)

Description:

Free a semaphore using the handle returned from a successful call to rtp_sig_semaphore_alloc.

Parameters:

semHandle	-	Handle to the semaphore to be freed.

Returns:

void

rtp_sig_semaphore_wait_timed

Function:

Wait for a semaphore to be signaled.

Summary:

int rtp_sig_semaphore_wait_timed (RTP_HANDLE semHandle,
    long msecs)

Description:

Wait for a semaphore to be signaled. If the semaphore has already been signaled this function returns immediatley, otherwise it should block until the semaphore is signaled or the millisecond time value has expired. Use 0 to poll the signal and return with its state immediately. Use -1 to indicate that an infinite timeout value should be used, blocking indefinitely for the signal.

Parameters:

semHandle	-	Handle to the semaphore to be checked.
msecs	-	Timeout value in milliseconds:

0	To poll.
-1	To indicate use of infinite.

Returns:

0 if successful, -1 on error, and 1 if the call timed out. For debugging purposes; if the cause of the error is obtainable at the native Kernel layer, turn on RTP_DISPLAY_ERROR in rtpsignl.c to display the native error value.

rtp_sig_semaphore_clear

Function:

Clear the state of a semaphore.

Summary:

void rtp_sig_semaphore_clear (RTP_HANDLE semHandle)

Description:

Clear the state of a semaphore. An example would be to set a counting semaphore to zero.

Parameters:

semHandle - Handle of the semaphore to clear.

Returns:

void

Preconditions:

Do not call this function unless past events have already been handled or their states are not needed. This will clear the signal and prepare to notify when the next event occurs.

rtp_sig_semaphore_wait

Function:

Wait for a semaphore to be signaled.

Summary:

int rtp_sig_semaphore_wait (RTP_HANDLE semHandle)

Description:

Wait for a semaphore to be signaled. If the semaphore has already been signaled this function returns immediatley, otherwise it should block indefinitely until the semaphore is signaled. This calls functionality is identical to calling rtp_sig_semaphore_wait_timed with a timeout value of -1 (INFINITE).

Parameters:

semHandle - Handle to the semaphore to be checked.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native Kernel layer, turn on RTP_DISPLAY_ERROR in rtpsignl.c to display the native error value.

rtp_sig_semaphore_signal

Function:

Signal a semaphore.

Summary:

void rtp_sig_semaphore_signal (RTP_HANDLE semHandle)

Description:

Signal a semaphore. An example would be to increment a counting semaphore.

Parameters:

semHandle	-	Handle to the semaphore to be signaled.

Returns:

void

rtp_sig_semaphore_signal_isr

Function:

Signal an interrupt service routine semaphore.

Summary:

void rtp_sig_semaphore_signal_isr (RTP_HANDLE semHandle)

Description:

Signal an interrupt service routine semaphore. In most implementations this will be identical to rtp_sig_semaphore_signal.

Parameters:

semHandle	-	Handle to the semaphore to be signaled.

Returns:

void

rtp_sig_mutex_alloc

Function:

Create and itialize a mutex.

Summary:

int rtp_sig_mutex_alloc (RTP_HANDLE* newMutex, const char* name)

Description:

Create and initalize a mutex in the non-claimed state.

Parameters:

newMutex	-	Storage location for the handle of the newly created mutex.
name	-	The name of the mutex.

Returns:

0 if successful, -1 otherwise. If a -1 is returned the value of *newMutex is undefined. For debugging purposes; if the cause of the error is obtainable at the native Kernel layer, turn on RTP_DISPLAY_ERROR in rtpsignl.c to display the native error value.

rtp_sig_mutex_free

Function:

Free a mutex.

Summary:

void rtp_sig_mutex_free (RTP_HANDLE mutexHandle)

Description:

Free a mutex using the handle returned from a successful call to rtp_sig_mutex_alloc.

Parameters:

mutexHandle	-	[tbd]

Returns:

void

rtp_sig_mutex_claim_timed

Function:

Wait for a mutex to be available.

Summary:

int rtp_sig_mutex_claim_timed (RTP_HANDLE mutexHandle,
long msecs)

Description:

Wait for a mutex to be available. If the mutex is already available this function returns immediatley, otherwise it should block until the mutex is available or the millisecond time value has expired. Use 0 to poll the mutex and return with its state immediately. Use -1 to indicate that an infinite timeout value should be used, blocking indefinitely for the mutex to become available.

Postconditon:

For every time this function is successful on a designated mutex handle, a respective call to rtp_sig_mutex_release must be made.

Parameters:

mutexHandle	-	Handle to the mutex to be checked.
msecs	-	Timeout value in milliseconds:
		0 To poll.
		-1 To indicate use of infinite.

Returns:

0 if successful, -1 on error, and 1 if the call timed out. For debugging purposes; if the cause of the error is obtainable at the native Kernel layer, turn on RTP_DISPLAY_ERROR in rtpsignl.c to display the native error value.

rtp_sig_mutex_claim

Function:

Wait for a mutex to be available.

Summary:

int rtp_sig_mutex_claim (RTP_HANDLE mutexHandle)

Description:

Wait for a mutex to be available. If the mutex is already available this function returns immediatley, otherwise it should block indefinitely or until the mutex is available.

Postconditon:

For every time this function is successful on a designated mutex handle, a respective call to rtp_sig_mutex_release must be made.

Parameters:

mutexHandle	-	Handle to the mutex to be claimed.

Returns:

0 if successful, -1 otherwise. For debugging purposes; if the cause of the error is obtainable at the native Kernel layer, turn on RTP_DISPLAY_ERROR in rtpsignl.c to display the native error value.

rtp_sig_mutex_release

Function:

Release a mutex.

Summary:

void rtp_sig_mutex_release (RTP_HANDLE mutexHandle)

Description:

Release a mutex

Parameters:

mutexHandle - Handle to the mutex to be released.

Returns:

void

Preconditions:

A successful call to rtp_sig_mutex_claim should be made before calling this function.

RTPSSL

SECURE SOCKET LAYER SERVICES

rtp_ssl_init_context	- Initialization for a secure client or server.
rtp_ssl_connect	- Establish a secure connection.
rtp_ssl_accept	- Accepts a secure connection.
rtp_ssl_send	- Send data over a secured socket.
rtp_ssl_recv	- Receive data over a secured socket.
rtp_ssl_close_stream	- Close a secure session and free its associated object.
rtp_ssl_close_context	- Close/Free an SSL context.

These functions **MUST** be ported.

rtp_ssl_init_context

Function:

Initialization for a secure client or server.

Summary:

int rtp_ssl_init_context (RTP_HANDLE* sslContext, unsigned int sslMode, unsigned int verifyMode)

Description:

Initializes a secure client or server context and returns it via the RTP_HANDLE* sslContext. This returned value should remain untouched outside of RTSSL and passed on to the other rtp_ssl calls that need it. When the server or client context is no longer needed by the application, a call to rtp_ssl_close_context should be made to free any memory that was allocated by this call.

Parameters:

sslContext - Pointer to the transparent SSL library context.

sslMode - MUST be an or'ed combination of the following:

RTP_SSL_MODE_SERVER	to initialize a server context
RTP_SSL_MODE_CLIENT	to initialize a client context
RTP_SSL_MODE_SSL2	to enable SSLv2
RTP_SSL_MODE_SSL3	to enable SSLv3
RTP_SSL_MODE_TLS1	to enable TLSv1

verifyMode - MUST be an or'ed combination of the following:

RTP_SSL_VERIFY_NONE	don't verify peer, even if a certificate is.passed and fails verification, the session continues.
RTP_SSL_VERIFY_PEER	(Client) if verifies the peer, and fails if no certificate is sent.(Server) verifies the peer only if a certificate is sent.
RTP_SSL_VERIFY_FAIL_IF_NO_PEER_CERT	demands a certificate from the peer.
RTP_SSL_VERIFY_CLIENT_ONCE	(Server ONLY) validate a clients certificate once; doesn't bother checking it again on a renegotiation.

Returns:

0 if successful, -1 otherwise.

rtp_ssl_connect

Function:

Establish a secure connection.

Summary:

int rtp_ssl_connect (RTP_HANDLE* sslStream,
RTP_HANDLE sockHandle, RTP_HANDLE sslContext)

Description:

Establishes a connection on the pre-established socket and forms an SSL session object based on this connection. This SSL session is passed back using the RTP_HANDLE* sslStream, and MUST remain untouched by code outside the RTSSL.

Parameters:

sslStream	-	Pointer to the transparent SSL session object.
sockHandle	-	The socket to communicate over.
sslContext	-	The transparent SSL context returned by rtp_ssl_init_context.

Returns:

0 if successful, -1 otherwise.

Preconditions:

A ‘clear-text’ connect should be successfully done before calling this function.

rtp_ssl_accept

Function:

Accepts a secure connection.

Summary:

int rtp_ssl_accept (RTP_HANDLE* sslStream, RTP_HANDLE
sockHandle, RTP_HANDLE sslContext)

Description:

Accepts a connection on the pre-established socket and forms an SSL session object based on this connection. This SSL session is passed back using the RTP_HANDLE* sslStream, and MUST remain untouched by code outside the RTSSL.

Parameters:

sslStream	-	Pointer to the transparent SSL session object.
sockHandle	-	The socket to communicate over.
sslContext	-	The transparent SSL context returned by rtp_ssl_init_context.

Returns:

0 if successful, -1 otherwise.

Preconditions:

A ‘clear-text’ accept should be successfully done before calling this function.

rtp_ssl_send

Function:

Send data over a secured socket.

Summary:

long rtp_ssl_send (RTP_HANDLE sslStream, unsigned char* buffer, long len, unsigned int boolBlocking)

Description:

Uses the pre-established socket now stored in the SSL session object returned from the successful secure accept or successful secure connect to send data while transparently encrypting the data so that it can be sent securely. This call takes place of the 'clear-text' send.

Parameters:

sslStream	-	The transparent SSL session object returned by rtp_ssl_connect or rtp_ssl_accept.
buffer	-	The buffer containing data to be sent securely.
len	-	Length of the data buffer.

boolBlocking - 0 if the socket associated with the SSL session object is nonblocking, 1 if it is in blocking mode.

Returns:

Number of bytes sent, -1 otherwise.

rtp_ssl_recv

Function:

Receive data over a secured socket.

Summary:

long rtp_ssl_recv (RTP_HANDLE sslStream, unsigned char* buffer, long len, unsigned int boolBlocking)

Description:

Uses the pre-established socket now stored in the SSL session object returned from the successful secure accept or successful secure connect to receive data while transparently decrypting the data so that it can be received. This call takes place of the 'clear-text' recv.

Parameters:

sslStream	-	The transparent SSL session object returned by rtp_ssl_connect or rtp_ssl_accept.
buffer	-	The buffer to store the securely received data.
len	-	Length of the data buffer.
boolBlocking	-	0 if the socket associated with the SSL session object is nonblocking, 1 if it is in blocking mode.

Returns:

Number of bytes received, -1 otherwise.

rtp_ssl_close_stream

Function:

Close a secure session and free its associated object.

Summary:

int rtp_ssl_close_stream (RTP_HANDLE sslStream)

Description:

Shuts down an SSL session, then frees the SSL session object. MUST be called before the socket that the SSL session is being ran over is closed. MUST be called once for every successful rtp_ssl_connect or rtp_ssl_accept call.

Parameters:

sslStream	-	The transparent SSL session object returned by rtp_ssl_connect or rtp_ssl_accept.

Returns:

0 if successful, -1 otherwise.

rtp_ssl_close_context

Function:

Close/Free an SSL context.

Summary:

void rtp_ssl_close_context (RTP_HANDLE sslContext)

Description:

Frees the current SSL context passed in, and if it is the last SSL context owned by the application this function will also free up any SSL library data setup by the rtp_ssl_init_context. MUST be called once for every SSL context initialized.

Parameters:

sslContext	-	The transparent SSL context returned by rtp_ssl_init_context.

RTPTERM

TERMINAL SERVICES

rtp_term_kbhit	- Check and see if there are any characters waiting.
rtp_term_getch	- Receive a single character from a terminal.
rtp_term_putc	- Output a single character from a terminal.
rtp_term_puts	- Output a string to a terminal, with a newline at the end.
rtp_term_cputs	- Output a string to a terminal.
rtp_term_gets	- Input a string from a terminal.
rtp_term_promptstring	- Allow the user to edit a string interactively.
rtp_term_up_arrow	- Used to check the response from rtp_term_promptstring.
rtp_term_down_arrow	- Used to check the response from rtp_term_promptstring.
rtp_term_left_arrow	- Used to check the response from rtp_term_promptstring.
rtp_term_right_arrow	- Used to check the response from rtp_term_promptstring.
rtp_term_escape_key	- Used to check the response from rtp_term_promptstring.

These functions **MUST** be ported.

rtp_term_kbhit

Function:

Check and see if there are any characters waiting.

Summary:

int rtp_term_kbhit (void)

Description:

Check and see if there are any characters waiting.

Returns:

Nonzero if a character is waiting, 0 otherwise.

rtp_term_getch

Function:

Receive a single character from a terminal.

Summary:

int rtp_term_getch (void)

Description:

Receive a single character from a terminal, and it should never echo the character back to the other side. This function will block if there are no characters available.

Returns:

The character received.

rtp_term_putc

Function:

Output a single character from a terminal.

Summary:

void rtp_term_putc (char ch)

Description:

Output a single character to a terminal. This function may block.

Parameters:

ch	- The character to output.

Returns:

void

rtp_term_puts

Function:

Output a string to a terminal, with a newline at the end.

Summary:

void rtp_term_puts (const char* string)

Description:

Output a string to a terminal, with a newline at the end. This function may block.

Parameters:

string	- The string to output.

Returns:

void

rtp_term_cputs

Function:

Output a string to a terminal.

Summary:

int rtp_term_cputs (const char* string)

Description:

Output a string to a terminal. This function may block.

Parameters:

string	- The string to output.

Returns:

0 if successful, -1 otherwise.

rtp_term_gets

Function:

Input a string from a terminal.

Summary:

int rtp_term_gets (char* string)

Description:

Input a string from a terminal. This function may block. It reads a string from the terminal device, and stops reading when it encounters a return (‘\r’) from the data stream. It calls rtp_term_promptstring (a line editing function) with an empty string. In the future, this may be a duplicate implementation of rtp_term_promptstring, because rtp_term_promptstring is at a slightly higher layer than rtp_term_gets. This is because rtp_term_gets is a standard ANSI I/O library function, but rtp_term_promptstring is not.

Parameters:

string	-	The string to received to.

Returns:

0 if successful, -1 otherwise.

rtp_term_promptstring

Function:

Allow the user to edit a string interactively.

Summary:

int rtp_term_promptstring (char* string, unsigned int handle_arrows)

Description:

Allow the user to edit a string interactively. This function may block. The handle_arrows flag is mostly useful in interactive shells and other interactive prompts, where an application might have a history of commands that have been entered, or a list of possible choices for an answer that may be scrolled through with the arrow keys.

Parameters:

string	-	The string to edit.
handle_arrows	-	Boolean flag for interactivity level.

Returns:

0 if successful, -1 otherwise, and if handle_arrows is set to 1 then it may return the result from rtp_term_up_arrow, rtp_term_down_arrow, or rtp_term_escape.

rtp_term_up_arrow

Function:

Used to check the response from rtp_term_promptstring.

Summary:

int rtp_term_up_arrow (void)

Description:

Used to check the response from rtp_term_promptstring. If the response is equivalent to rtp_term_up_arrow, then an up arrow was entered at the terminal.

Returns:

Up arrow value.

rtp_term_down_arrow

Function:

Used to check the response from rtp_term_promptstring.

Summary:

int rtp_term_down_arrow (void)

Description:

Used to check the response from rtp_term_promptstring. If the response is equivalent to rtp_term_down_arrow, then a down arrow was entered at the terminal.

Returns:

Down arrow value.

rtp_term_left_arrow

Function:

Used to check the response from rtp_term_promptstring.

Summary:

int rtp_term_left_arrow (void)

Description:

Used to check the response from rtp_term_promptstring. If the response is equivalent to rtp_term_left_arrow, then a left arrow was entered at the terminal.

Returns:

Left arrow value.

rtp_term_right_arrow

Function:

Used to check the response from rtp_term_promptstring.

Summary:

int rtp_term_right_arrow (void)

Description:

Used to check the response from rtp_term_promptstring. If the response is equivalent to rtp_term_right_arrow, then a right arrow was entered at the terminal.

Returns:

Right arrow value.

rtp_term_escape_key

Function:

Used to check the response from rtp_term_promptstring.

Summary:

int rtp_term_escape_key (void)

Description:

Used to check the response from rtp_term_promptstring. If the response is equivalent to rtp_term_escape_key, then escape was entered at the terminal.

Returns:

Escape value.

RTPTHRD

THREAD SERVICES

rtp_threads_init	- Prepare for use of the RTPlatform thread API.
rtp_threads_shutdown	- Shutdown the RTPlatform thread API.
rtp_thread_spawn	- Spawn a thread.
rtp_thread_handle	- Get the current thread handle.
rtp_thread_user_data	- Gets the user data associated with the current thread.
rtp_thread_user_data_by_handle	- Gets the user data associated with the thread handle.
rtp_thread_name	- Gets the name associated with the current thread.
rtp_thread_name_by_handle	- Gets the name associated with the thread handle.
rtp_thread_sleep	- Put the current thread to sleep.
rtp_thread_yield	- Yield the current thread to any other of equal priority.

These functions **MUST** be ported.

rtp_threads_init

Function:

Prepare for use of the RTPlatform thread API.

Summary:

int rtp_threads_init (void)

Description:

Prepare for use of the RTPlatform thread API.

Returns:

0 on success, -1 otherwise.

rtp_threads_shutdown

Function:

Shutdown the RTPlatform thread API.

Summary:

void rtp_threads_shutdown (void)

Description:

When use of the RTPlatform thread API is no longer needed, this function clears and/or releases any allocated resources setup by the rtp_threads_init function call.

Returns:

void

rtp_thread_spawn

Function:

Spawn a thread.

Summary:

int rtp_thread_spawn (RTP_HANDLE* newThread,
RTP_ENTRY_POINT_FN entryPoint, const char* name,
int stackSizeIndex, int priorityIndex, void* userData)

Description:

Spawn a thread and sets its priority level.

Parameters:

newThread	- Storage location for the handle to the new thread.
entryPoint	- A function pointer to the threads entry point.
name	- The name of the thread.
stackSizeIndex	- An index into the array of possible stack sizes.
priorityIndex	- An index into the array of possible priorities.
userData	- User data to be passed to the entry point function.

Returns:

0 on success, -1 on failure, and -2 on non-fatal error. If a -1 is returned the value of *newThread is undefined. An example of a non-fatal error would be if the thread was spawned but the priority level could not be set.

rtp_thread_handle

Function:

Get the current thread handle.

Summary:

int rtp_thread_handle (RTP_HANDLE* currentThread)

Description:

Retrieves the handle to the current thread.

Parameters:

currentThread	-	Storage location for the thread handle.

Returns:

0 on success, -1 otherwise.

rtp_thread_user_data

Function:

Gets the user data associated with the current thread.

Summary:

int rtp_thread_user_data (void** userData)

Description:

Retrieves the user data from the current thread.

Parameters:

userData	-	Storage location for the user data.

Returns:

0 on success, -1 otherwise.

rtp_thread_user_data_by_handle

Function:

Gets the user data associated with the thread handle.

Summary:

int rtp_thread_user_data_by_handle (RTP_HANDLE handle,
    void** userData)

Description:

Retrieves the user data from the thread with the associated thread handle.

Parameters:

handle	-	Handle to the thread to find its user data.
userData	-	Storage location for the user data.

Returns:

0 on success, -1 otherwise.

rtp_thread_name

Function:

Gets the name associated with the current thread.

Summary:

int rtp_thread_name (char** name)

Description:

Retrieves the name from the current thread.

Parameters:

name	-	Storage location for the name.

Returns:

0 on success, -1 otherwise.

rtp_thread_name_by_handle

Function:

Gets the name associated with the thread handle.

Summary:

int rtp_thread_name_by_handle (RTP_HANDLE handle, char** name)

Description:

Retrieves the name from the thread with the associated thread handle.

Parameters:

handle	-	Handle to the thread to find its name.
name	-	Storage location for the name.

Returns:

0 on success, -1 otherwise.

rtp_thread_sleep

Function:

Put the current thread to sleep.

Summary:

void rtp_thread_sleep (long msecs)

Description:

Suspend the current thread for the millisecond timeout period.

Parameters:

msecs	-	Timeout value in milliseconds.

Returns:

void

rtp_thread_yield

Function:

Yield the current thread to any other of equal priority.

Summary:

void rtp_thread_yield (void)

Description:

Check if any other threads are attempting to run that are of equal priority. If there is one of equal proritey ready to run, let it, otherwise the yield returns and the current thread continues.

Returns:

void

RTPTIME

SYSTEM TIME SERVICES

rtp_get_system_msec	- Retrieves the elapsed milliseconds since the application started.
rtp_get_system_sec	- Retrieves the elapsed seconds since the application started.

These functions **MUST** be ported.

rtp_get_system_msec

Function:

Retrieves the elapsed milliseconds since the application started.

Summary:

unsigned long rtp_get_system_msec (void)

Description:

Retrieves the elapsed milliseconds since the application started. Can be used as a millisecond timer by storing the first call and periodically checking against successive calls until the timeout period has been reached.

Returns:

The elapsed milliseconds. There is no error return.

rtp_get_system_sec

Function:

Retrieves the elapsed seconds since the application started.

Summary:

unsigned long rtp_get_system_sec (void)

Description:

Retrieves the elapsed seconds since the application started. Can be used as a second timer by storing the first call and periodically checking against successive calls until the timeout period has been reached.

Returns:

The elapsed seconds. There is no error return.

APPENDIX A

PORTING AN RTPLATFORM-ENABLED PRODUCT TO A NEW ENVIROMENT

EBSnet's RTPlatform-enabled products all require a certain subset of the total services provided by the RTPlatform library. For each product, there is a list of all the modules required to provide the necessary services to that product. (For example, FTP may require rtpnet for networking, rtpfile for file system access, rtpdate for date management, and rtpsignl/rtpthrd for multi-threaded operation.)

The first step in the porting process is to create a new build environment, i.e. a new makefile or project file, for the new target. All the product's "core" files (the ones in the product's example project file that do not begin with "rtp", and thus are not a part of RTPlatform) should be added to the new makefile/project file.

The next step in porting an RTPlatform-enabled product to a new environment is to identify the list of RTPlatform modules required by that product. Once this has been done, this list can be separated into those modules which have generic, portable implementations (and thus to do need to be modified in order to support a new environment), and those which are specific to a particular OS, CPU, software library, etc. The generic RTPlatform modules are located in the RTPlatform install directory under "source/generic". These files should be added into the makefile/project file "as is".

Once this is complete, environment-specific versions of the non-portable modules must be added into the build environment. Those modules that are already ported should be added to the build environment. The remaining modules must be ported manually to the new target.

To port an RTPlatform module to a new target, first locate the template version of that module under "source/template". Create a new directory inside "source/" for the new target and copy the template version into this directory; this is the copy of the file that you will modify to implement the required functionality in your target environment. Each template file has documentation in the comments above each function that describes what the function should do.