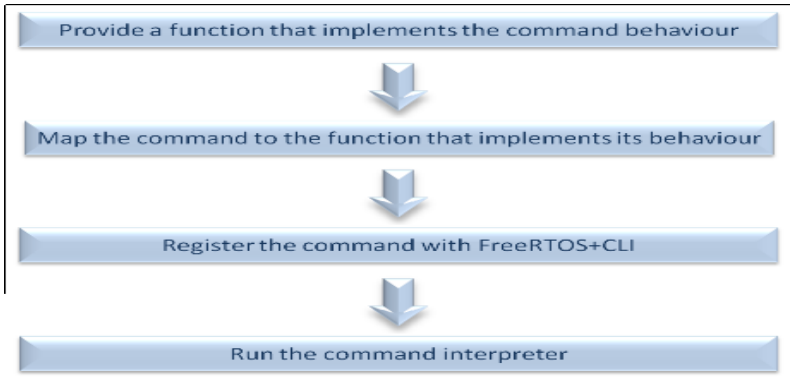
FreeRTOS+CLI
An Extensible Command Line Interface Framework

Introduction

FreeRTOS+CLI (Command Line Interface) provides a simple, small, extensible and RAM efficient method of enabling your FreeRTOS application to process command line input. The generic process for this is shown as follows.



Implementing a Command

FreeRTOS+CLI is an extensible framework that allows the application writer to define and register their own command line input commands. This page describes how to write a function that implements the behaviour of a command.

Function inputs and outputs

Functions that implement the behaviour of a user defined command must have the following interface (prototype):

```
portBASE_TYPE xFunctionName( int8_t *pcWriteBuffer,
                             size_t xWriteBufferLen,
                             const int8_t *pcCommandString );
```

Following is a description of the parameters that will be passed into the function when it is called, and the value that must be returned.

Parameters:

pcWriteBuffer	This is the buffer into which any generated output should be written. For example, if the function is simply going to return th fixed string "Hello World", then the string is written into pcWriteBuffer. Output **must always** be null terminated.
xWriteBufferLen	This is the size of the buffer pointed to by the pcWriteBuffer parameter. Writing more than xWriteBufferLen characters into pcWriteBuffer will cause a buffer overflow.
pcCommandString	A pointer to the entire command string. Having access to the entire command string allows the function implementation to extract the command parameters - if there are any. FreeRTOS+CLI provides helper functions that accept the command string and return the command parameters - so explicit string parsing is not required. Examples are provided on this page.

Returns:

Executing some commands will result in more than a single line of output being produced. For example, a file system "dir" (or "ls") command will generate a line of output for each file in a directory. If there are three files in the directory, the output might look as below:

```
file1.txt
file2.txt
file3.txt
```

To minimise RAM usage, and ensure RAM usage is deterministic, FreeRTOS+CLI allows functions that implement command behaviour to output a single line at a time. The function return value is used to indicate whether the output line is the end of the output, or if there are more lines to be generated.

Return **pdFALSE** if the generated output is the end of the output, meaning there are no more lines to be generated, and the command execution is complete.

Return **pdTRUE** if the returned output is not the end of the output, and there are still one or more lines to be generated before the command execution is complete.

To continue the example of the "dir" command that outputs three file names:

1. The first time the function that implements the dir command is called, it is possible to only output the first line (file1.txt). If this is done the function must return pdTRUE to indicate there are more lines to follow.

2. The second time the function that implements the dir command is called, it is possible to only output the second line (file2.txt). If this is done the function must return pdTRUE again to indicate there are more lines to follow.

3. The third time the function that implements the dir command is called, only the third line (file3.txt) will be output. This time, there are no more lines to output, so the function must return pdFALSE.

Alternatively, if there is sufficient RAM, and the value passed in xWriteBufferLen is large enough, all three lines could have been returned at once - in which case the function must return pdFALSE on its first execution.

Each time a command is executed, FreeRTOS+CLI will repeatedly call the function that implements the command behaviour, until the function returns pdFALSE.

Examples

The following examples are provided below:

1. A command that takes no parameters and returns a single string.
2. A command that takes no parameters and returns multiple strings, one line at a time.
3. A command that expects a fixed number of parameters.
4. A command that accepts a variable number of parameters, and returns a variable number of strings one line at a time.

Example 1: A command with no parameters

The FreeRTOS vTaskList() API function generates a table containing information on the state of each task. The table contains a line of text for each task. The command implemented in Example 1 outputs this table. Example 1 demonstrates the simple case where the entire table is output at once. The comments in the code provide more explanation.

```
/* This function implements the behaviour of a command, so must have the correct
prototype. */
static portBASE_TYPE prvTaskStatsCommand( int8_t *pcWriteBuffer,
                                          size_t xWriteBufferLen,
                                          const int8_t *pcCommandString )
{
    /* For simplicity, this function assumes the output buffer is large enough
```

```
    to hold all the text generated by executing the vTaskList() API function,
    so the xWriteBufferLen parameter is not used. */
    ( void ) xWriteBufferLen;

    /* pcWriteBuffer is used directly as the vTaskList() parameter, so the table
    generated by executing vTaskList() is written directly into the output
    buffer. */
    vTaskList( pcWriteBuffer + strlen( pcHeader ) );

    /* The entire table was written directly to the output buffer.  Execution
    of this command is complete, so return pdFALSE. */
    return pdFALSE;
}
```

Example 1: Outputting multiple lines at once

Example 2: Returning multiple lines one line at a time

Every command registered with FreeRTOS+CLI has its own help string. The help string is one line of text that demonstrates how the command is used. FreeRTOS+CLI includes a "help" command that returns all the help strings, providing the user with a list of available commands along with instructions on how each command is used. Example 2 shows the implementation of the help commend. Unlike example 1, where all the output was generated in one go, example 2 generates a single line at a time. Note this function is **not** re-entrant.

```
/* This function implements the behaviour of a command, so must have the correct
prototype. */
static portBASE_TYPE prvHelpCommand( int8_t *pcWriteBuffer,
                                     size_t xWriteBufferLen,
                                     const int8_t *pcCommandString )
{
/* Executing the "help" command will generate multiple lines of text, but this
function will only output a single line at a time.  Therefore, this function is
called multiple times to complete the processing of a single "help" command.  That
means it has to remember which help strings it has already output, and which
still remain to be output.  The static pxCommand variable is used to point to the
next help string that needs outputting. */
static const xCommandLineInputListItem *pxCommand = NULL;
signed portBASE_TYPE xReturn;

    if( pxCommand == NULL )
    {
        /* pxCommand is NULL in between executions of the "help" command, so if
        it is NULL on entry to this function it is the start of a new "help" command
        and the first help string is returned.  The following line points pxCommand
        to the first command registered with FreeRTOS+CLI. */
        pxCommand = &xRegisteredCommands;
    }
```

```
    /* Output the help string for the command pointed to by pxCommand, taking
    care not to overflow the output buffer. */
    strncpy( pcWriteBuffer,
             pxCommand->pxCommandLineDefinition->pcHelpString,
             xWriteBufferLen );

    /* Move onto the next command in the list, ready to output the help string
    for that command the next time this function is called. */
    pxCommand = pxCommand->pxNext;

    if( pxCommand == NULL )
    {
        /* If the next command in the list is NULL, then there are no more
        commands to process, and pdFALSE can be returned. */
        xReturn = pdFALSE;
    }
    else
    {
        /* If the next command in the list is not NULL, then there are more
        commands to process and therefore more lines of output to be generated.
        In this case pdTRUE is returned. */
        xReturn = pdTRUE;
    }

    return xReturn;
}
```

Example 2: Generating multiple lines of output, one line at a time

Example 3: A command with a fixed number of parameters

Some commands take parameters. For example, a file system "copy" command needs the name of the source file and the name of the destination file. Example 3 is a framework for a copy command and is provided to demonstrate how command parameters are accessed and used.

Note that, if this command is declared to take two parameters when it is registered, FreeRTOS+CLI will not even call the command unless exactly two parameters are supplied.

```
/* This function implements the behaviour of a command, so must have the correct
prototype. */
static portBASE_TYPE prvCopyCommand( int8_t *pcWriteBuffer,
                                     size_t xWriteBufferLen,
                                     const int8_t *pcCommandString )
{
int8_t *pcParameter1, *pcParameter2;
portBASE_TYPE xParameter1StringLength, xParameter2StringLength, xResult;

    /* Obtain the name of the source file, and the length of its name, from
    the command string. The name of the source file is the first parameter. */
```

```
    pcParameter1 = FreeRTOS_CLIGetParameter
                        (
                          /* The command string itself. */
                          pcCommandString,
                          /* Return the first parameter. */
                          1,
                          /* Store the parameter string length. */
                          &xParameter1StringLength
                        );

    /* Obtain the name of the destination file, and the length of its name. */
    pcParameter2 = FreeRTOS_CLIGetParameter( pcCommandString,
                                             2,
                                             &xParameter2StringLength );

    /* Terminate both file names. */
    pcParameter1[ xParameter1StringLength ] = 0x00;
    pcParameter2[ xParameter2StringLength ] = 0x00;

    /* Perform the copy operation itself. */
    xResult = prvCopyFile( pcParameter1, pcParameter2 );

    if( xResult == pdPASS )
    {
        /* The copy was successful.  There is nothing to output. */
        *pcWriteBuffer == NULL;
    }
    else
    {
        /* The copy was not successful.  Inform the users. */
        snprintf( pcWriteBuffer, xWriteBufferLen, "Error during copy¥r¥n¥r¥n" );
    }

    /* There is only a single line of output produced in all cases.  pdFALSE is
    returned because there is no more output to be generated. */
    return pdFALSE;
}
```

Example 3: Accessing and using the command parameters

Example 4: A command with a variable number of parameters

Example 4 demonstrates how to create and implement a command that accepts a variable number of parameters. FreeRTOS+CLI will not check the number of supplied parameters, and the implementation of the command simply echos parameter back, one at a time. For example, if the assigned command string was "echo_parameters", if the user enters:

```
"echo_parameters one two three four"
```

Then the generated out will be:

```
The parameters were:
1: one
2: two
3: three
4: four
```

```
static portBASE_TYPE prvParameterEchoCommand(     int8_t *pcWriteBuffer,
                                                 size_t xWriteBufferLen, c
                                                 onst int8_t *pcCommandString )
{
int8_t *pcParameter;
portBASE_TYPE lParameterStringLength, xReturn;

/* Note that the use of the static parameter means this function is not reentrant. */
static portBASE_TYPE lParameterNumber = 0;

    if( lParameterNumber == 0 )
    {
        /* lParameterNumber is 0, so this is the first time the function has been
        called since the command was entered.  Return the string "The parameters
        were:" before returning any parameter strings. */
        sprintf( pcWriteBuffer, "The parameters were:¥r¥n" );

        /* Next time the function is called the first parameter will be echoed
        back. */
        lParameterNumber = 1L;

        /* There is more data to be returned as no parameters have been echoed
        back yet, so set xReturn to pdPASS so the function will be called again. */
        xReturn = pdPASS;
    }
    else
    {
        /* lParameter is not 0, so holds the number of the parameter that should
        be returned.  Obtain the complete parameter string. */
        pcParameter = ( int8_t * ) FreeRTOS_CLIGetParameter
                                    (
                                        /* The command string itself. */
                                        pcCommandString,
                                        /* Return the next parameter. */
                                        lParameterNumber,
                                        /* Store the parameter string length. */
                                        &lParameterStringLength
                                    );

        if( pcParameter != NULL )
        {
            /* There was another parameter to return.  Copy it into pcWriteBuffer.
```

```
            in the format "[number]: [Parameter String". */
            memset( pcWriteBuffer, 0x00, xWriteBufferLen );
            sprintf( pcWriteBuffer, "%d: ", lParameterNumber );
            strncat( pcWriteBuffer, pcParameter, lParameterStringLength );
            strncat( pcWriteBuffer, "¥r¥n", strlen( "¥r¥n" ) );

            /* There might be more parameters to return after this one, so again
            set xReturn to pdTRUE. */
            xReturn = pdTRUE;
            lParameterNumber++;
        }
        else
        {
            /* No more parameters were found.  Make sure the write buffer does
            not contain a valid string to prevent junk being printed out. */
            pcWriteBuffer[ 0 ] = 0x00;

            /* There is no more data to return, so this time set xReturn to
            pdFALSE. */
            xReturn = pdFALSE;

            /* Start over the next time this command is executed. */
            lParameterNumber = 0;
        }
    }

    return xReturn;
}
```

Example 3: Accessing a variable number of parameters

FreeRTOS_CLIRegisterCommand()

FreeRTOS_CLI.h

```
portBASE_TYPE FreeRTOS_CLIRegisterCommand( CLI_Command_Definition_t *pxCommandToRegister )
```

FreeRTOS+CLI is an extensible framework that allows the application writer to define and register their own command line input commands. Functions that implement the behaviour of a user defined command have to use a particular interface.

This section describes FreeRTOS_CLIRegisterCommand(), which is the API function used to register commands with FreeRTOS+CLI. A command is registered by associating the function that implements the command behaviour with a text string, and telling FreeRTOS+CLI about the association. FreeRTOS+CLI will then automatically run the function each time the command text string is entered. This will become clear after reading this page.

Note: The FreeRTOS_CLIRegisterCommand() prototype that appears in the code take a const pointer to a const structure of type CLI_Command_Definition_t. The const qualifiers have been removed here

to make the prototype easier to read.

Parameters:

pxCommandToRegister	The command being registered, which is defined by a structure of type CLI_Command_Definition_t. The structure is described below this table.

Returns:

pdPASS is returned if the command was successfully registered.

pdFAIL is returned if the command could not be registered because there was insufficient FreeRTOS heap available for a new list item to be created.

CLI_Command_Definition_t

Commands are defined by a structure of type CLI_Command_Definition_t. The structure is shown below. The comments in the code describe the structure members.

```
typedef struct xCLI_COMMAND_DEFINITION
{
    /* The command line input string.  This is the string that the user enters
    to run the command.  For example, the FreeRTOS+CLI help function uses the
    string "help".  If a user types "help" the help command executes. */
    const int8_t * const pcCommand;

    /* A string that describes the command, and its expected parameters.  This
    is the string that is output when the help command is executed.  The string
    must start with the command itself, and end with "¥r¥n".  For example, the
    help string for the help command itself is:
    "help: Returns a list of all the commands¥r¥n" */
    const int8_t * const pcHelpString;

    /* A pointer to the function that implements the command behaviour
    (effectively the function name). */
    const pdCOMMAND_LINE_CALLBACK pxCommandInterpreter;

    /* The number of parameters required by the command.  FreeRTOS+CLI will only
    execute the command if the number of parameters entered on the command line
    matches this number. */
    int8_t cExpectedNumberOfParameters;
} CLI_Command_Definition_t;
```

The CLI_Command_Definition_t structure

Examples

One of the FreeRTOS+CLI featured demos implements a file system "del" command. The command definition is given below.

```
static const CLI_Command_Definition_t xDelCommand =
```

```
{
    "del",
    "del <filename>: Deletes <filename> from the disk¥r¥n",
    prvDelCommand,
    1
};
```

The definition of the file system del command

Once this command is registered:

- ⑩ prvDelCommand() is executed each time the user types "del".
- ⑩ *"del <filename>: Deletes <filename> from the disk\r\n"* is output to describe the del command when the user types "help".
- ⑩ The del command expects one parameter (the name of the file being deleted). FreeRTOS+CLI will output an error string instead of executing prvDelCommand() if the number of input parameters is not exactly 1.

The del command is then registered with FreeRTOS+CLI using the following function call:

```
FreeRTOS_CLIRegisterCommand( &xDelCommand );
```

Registering the xDelCommand structure with FreeRTOS+CLI

FreeRTOS_CLIRegisterCommand()

FreeRTOS_CLI.h

```
portBASE_TYPE FreeRTOS_CLIRegisterCommand( CLI_Command_Definition_t *pxCommandToRegister )
```

FreeRTOS+CLI is an extensible framework that allows the application writer to define and register their own command line input commands.

This section describes FreeRTOS_CLIRegisterCommand(), which is the API function used to register commands with FreeRTOS+CLI. A command is registered by associating the function that implements the command behaviour with a text string, and telling FreeRTOS+CLI about the association. FreeRTOS+CLI will then automatically run the function each time the command text string is entered. This will become clear after reading this page.

Note: The FreeRTOS_CLIRegisterCommand() prototype that appears in the code take a const pointer to a const structure of type CLI_Command_Definition_t. The const qualifiers have been removed here to make the prototype easier to read.

Parameters:

pxCommandToRegister	The command being registered, which is defined by a structure of type CLI_Command_Definition_t. The structure is described below this table.

Returns:

pdPASS is returned if the command was successfully registered.

pdFAIL is returned if the command could not be registered because there was insufficient FreeRTOS heap available for a new list item to be created.

CLI_Command_Definition_t

Commands are defined by a structure of type CLI_Command_Definition_t. The structure is shown below. The comments in the code describe the structure members.

```
typedef struct xCLI_COMMAND_DEFINITION
{
    /* The command line input string.  This is the string that the user enters
    to run the command.  For example, the FreeRTOS+CLI help function uses the
    string "help".  If a user types "help" the help command executes. */
    const int8_t * const pcCommand;

    /* A string that describes the command, and its expected parameters.  This
    is the string that is output when the help command is executed.  The string
    must start with the command itself, and end with "¥r¥n".  For example, the
    help string for the help command itself is:
    "help: Returns a list of all the commands¥r¥n" */
    const int8_t * const pcHelpString;

    /* A pointer to the function that implements the command behaviour
    (effectively the function name). */
    const pdCOMMAND_LINE_CALLBACK pxCommandInterpreter;

    /* The number of parameters required by the command.  FreeRTOS+CLI will only
    execute the command if the number of parameters entered on the command line
    matches this number. */
    int8_t cExpectedNumberOfParameters;
} CLI_Command_Definition_t;
```

The CLI_Command_Definition_t structure

Examples

One of the FreeRTOS+CLI featured demos implements a file system "del" command. The command definition is given below.

```
static const CLI_Command_Definition_t xDelCommand =
{
    "del",
    "del <filename>: Deletes <filename> from the disk¥r¥n",
    prvDelCommand,
    1
};
```

The definition of the file system del command

Once this command is registered:

- prvDelCommand() is executed each time the user types "del".
- *"del <filename>: Deletes <filename> from the disk\r\n"* is output to describe the del command when the user types "help".
- The del command expects one parameter (the name of the file being deleted). FreeRTOS+CLI will output an error string instead of executing prvDelCommand() if the number of input parameters is not exactly 1.

The del command is then registered with FreeRTOS+CLI using the following function call:

```
FreeRTOS_CLIRegisterCommand( &xDelCommand );
```

Registering the xDelCommand structure with FreeRTOS+CLI

FreeRTOS+IO Integration

FreeRTOS+CLI is an extensible framework that allows the application writer to define and register their own command line input commands.

This section describes how to port FreeRTOS+CLI onto real hardware by providing input output (IO) routines, and a FreeRTOS+CLI task.

Input and output

A command line interface receives characters from an input, and writes characters to an output. The low level details of how this is achieved is dependent on the microcontroller being used, and the interfaces the microcontroller provides.

There is a FreeRTOS+CLI featured demo that uses the FreeRTOS+IO FreeRTOS_read() and FreeRTOS_write() API functions to provide the necessary input and output to a UART. The command line interface it creates is accessed using a standard dumb terminal program, such as HyperTerminal. There is another FreeRTOS+CLI featured demo that uses a TCP/IP sockets interface to provide the necessary input and output. The command line interface it creates is accessed using a telnet client. The structure of the FreeRTOS task that runs the FreeRTOS+CLI code is similar in both cases, and is provided below.

An example FreeRTOS+CLI task

The source code below implements a task that manages a FreeRTOS+CLI command interpreter interface. The FreeRTOS+IO FreeRTOS_read() and FreeRTOS_write() API functions are use to provide the IO interface. It is assumed that the FreeRTOS+IO descriptor has already been opened and configured to use the interrupt driven character queue transfer mode.

The task uses the FreeRTOS+CLI FreeRTOS_CLIProcessCommand() API function.

The comments in the source code provide more information. Note this function is not re-entrant.

```
#define MAX_INPUT_LENGTH    50
#define MAX_OUTPUT_LENGTH   100

static const int8_t * const pcWelcomeMessage =
  "FreeRTOS command server.¥r¥nType Help to view a list of registered commands.¥r¥n";

void vCommandConsoleTask( void *pvParameters )
{
Peripheral_Descriptor_t xConsole;
int8_t cRxedChar, cInputIndex = 0;
portBASE_TYPE xMoreDataToFollow;
/* The input and output buffers are declared static to keep them off the stack. */
static int8_t pcOutputString[ MAX_OUTPUT_LENGTH ], pcInputString[ MAX_INPUT_LENGTH ];

    /* This code assumes the peripheral being used as the console has already
    been opened and configured, and is passed into the task as the task parameter.
    Cast the task parameter to the correct type. */
    xConsole = ( Peripheral_Descriptor_t ) pvParameters;

    /* Send a welcome message to the user knows they are connected. */
    FreeRTOS_write( xConsole, pcWelcomeMessage, strlen( pcWelcomeMessage ) );

    for( ;; )
    {
        /* This implementation reads a single character at a time.  Wait in the
        Blocked state until a character is received. */
        FreeRTOS_read( xConsole, &cRxedChar, sizeof( cRxedChar ) );

        if( cRxedChar == '¥n' )
        {
            /* A newline character was received, so the input command stirng is
            complete and can be processed.  Transmit a line separator, just to
            make the output easier to read. */
            FreeRTOS_write( xConsole, "¥r¥n", strlen( "¥r¥n" ) );

            /* The command interpreter is called repeatedly until it returns
            pdFALSE.  See the “Implementing a command” documentation for an
            exaplanation of why this is. */
            do
            {
                /* Send the command string to the command interpreter.  Any
                output generated by the command interpreter will be placed in the
                pcOutputString buffer. */
                xMoreDataToFollow = FreeRTOS_CLIProcessCommand
                            (
                                pcInputString,   /* The command string.*/
                                pcOutputString,  /* The output buffer. */
                                MAX_OUTPUT_LENGTH/* The size of the output buffer. */
                            );

                /* Write the output generated by the command interpreter to the
                console. */
```

```
                FreeRTOS_write( xConsole, pcOutputString, strlen( pcOutputString ) );

            } while( xMoreDataToFollow != pdFALSE );

            /* All the strings generated by the input command have been sent.
            Processing of the command is complete.  Clear the input string ready
            to receive the next command. */
            cInputIndex = 0;
            memset( pcInputString, 0x00, MAX_INPUT_LENGTH );
        }
        else
        {
            /* The if() clause performs the processing after a newline character
            is received.  This else clause performs the processing if any other
            character is received. */

            if( cRxedChar == '¥r' )
            {
                /* Ignore carriage returns. */
            }
            else if( cRxedChar == '¥b' )
            {
                /* Backspace was pressed.  Erase the last character in the input
                buffer - if there are any. */
                if( cInputIndex > 0 )
                {
                    cInputIndex--;
                    pcInputString[ cInputIndex ] = '¥0';
                }
            }
            else
            {
                /* A character was entered.  It was not a new line, backspace
                or carriage return, so it is accepted as part of the input and
                placed into the input buffer.  When a ¥n is entered the complete
                string will be passed to the command interpreter. */
                if( cInputIndex < MAX_INPUT_LENGTH )
                {
                    pcInputString[ cInputIndex ] = cRxedChar;
                    cInputIndex++;
                }
            }
        }
    }
}
```

An example of a task that implements a FreeRTOS+CLI command console