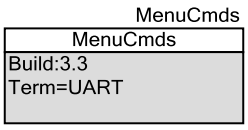
MenuCmds
3.3.4

Features



- Simple interface to create a single character Menu Command structure tied to a Terminal component.
- Allows creation of submenus to group commands
- Allows command protection and hidden attributes.

General description

The MenuCmds Component implements a simple API interface to the designer to capture input or display the current MenuCmd structure using help commands using a Terminal device.

Once a correct MenuCmds structure is allocated, typing a single character that matches one of the elements in the structure will invoke the function assigned to that command.

Although this component can be installed separately in a TopDesign schematic it must be used with a Terminal component such as the Cypress UART or USBUART. For this reason, if you choose to use the custom Term component it already has this component embedded in it to simplify the design of its use in an Application.

When to use MenuCmds component

The MenuCmds Component was developed to simplify the design of Terminal Menu command handling.

Use of this custom component requires the designer to configure the project to reference the component library supplied. Refer to "How to Access Custom Libraries and Component" for methods of access available.

MenuCmds Requirements

The MenuCmds component is tied to a Terminal component in the PsoC design. To complete this link to the Terminal component, you must:

- Allocate a Terminal (ie UART or USBUART) component. In this component we have provided a “Term” component that can be configured very easily to be a UART-style or USBUART-style.
- Pointers to the _PutString() API call of the selected Terminal component need to be installed as callbacks before use of the functions. Here is an example:
  MenuCmds _SetCallbacks((void *)&Term_PutString);
  If you are using the MenuCmds component directly in the TopDesign and you supply the instanced name of the Terminal component in the “Terminal name” field of this component, you can substitute the MenuCmds _Init()or MenuCmds _Start() in place of the MenuCmds _SetCallbacks() call. These API calls will load the appropriate callback pointers.

Device Families Supported

At this time, all PsoC devices support this component. It has been only tested on PsoC5 and PsoC6 devices. This component was intended to be generic and not PsoC device specific. Theoretically, this code could be portable to ANY system supporting C code.

Implementation Limitations

This version of the MenuCmds component should satisfy most Application needs. Should you require a different implementation C source code is included in this component.

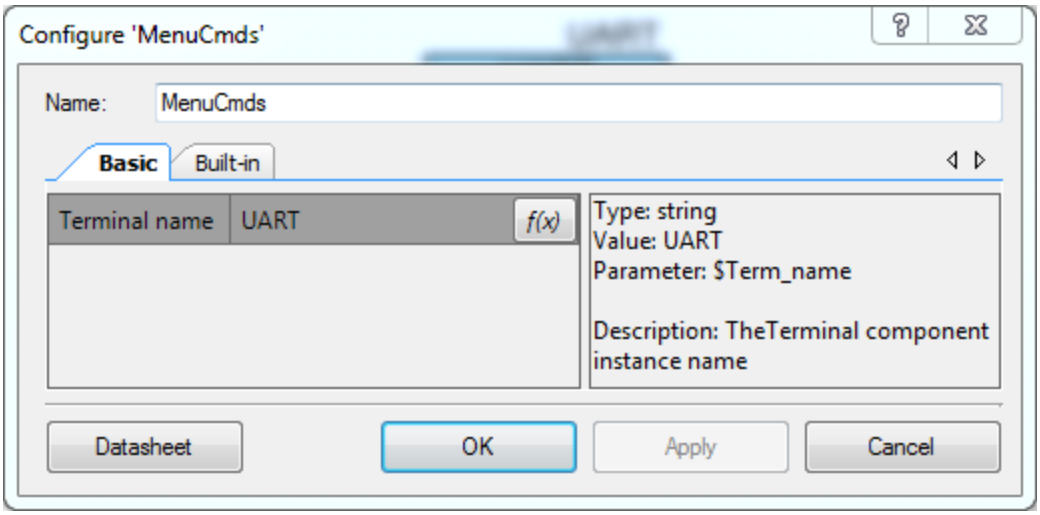
The MenuCmds API functions are implemented as NON-BLOCKING functions. The function you assign to the command is your responsibility whether it is implemented as blocking or non-blocking.

There are additional MenuCmds Structure creation limitations. See
MenuCmds_Structure_Limitations

Parameters and Settings

Here is the parameter setting in the Configuration for MenuCmds component.

Basic tab



This tab provides following parameters:

Terminal name

- Type the name of the Terminal UART component instance name. This name is used to associate the Terminal component name for the MenuCmds_Init() or MenuCmds_Start() functions. For example: if you use the Cypress UART component and its name is "UART_1", type the name in this field.

Application Programming Interface (API)

To use this component, you must:

- Drag the MenuCmds component on the TopDesign schematic.
- Configure the component by typing in the Terminal UART component instance name in the "Terminal name" field.

These are enum types and the API calls supported:

Function	Description
MenuCmds_SetCallbacks()	Set CallBacks for Terminal PutString ()
MenuCmds_Init()	Call to _ SetCallbacks.
MenuCmds_Start()	Call to _ SetCallbacks.
MenuCmds_process()	Find a match to a MenuCmd element from the input Terminal then execute the assigned function.
MenuCmds_search()	Find a match to a MenuCmd element from the input Terminal then return the MenuCmd element.

MenuCmds_num_func_SetCallback()	Set a special function Callback for supporting single numbers [0-9]

int MenuCmds _SetCallbacks(void (*putstring)(char *string));

Mandatory: Initialize the MenuCmds component by installing callbacks to Terminal functions. Not issuing this function with correct pointers to _GetChar() and _PutChar() functions before use will yield no communication results.

This function allows the designer to switch to different Terminal ports at run-time if desired.

Parameters:

&_PutString	Pointer to the Terminal's _PutString function.

Returns:
error // 0 = No Error

Example:

```
      // Term_ is the TopDesign terminal component name.
MenuCmds _SetCallbacks ( (void *)&Term_PutString);
```

int MenuCmds _Init(void)

Use this function to initialize the MenuCmds component. It initializes the callback for _PutString() to another Terminal instance name listed in the "Terminal name" parameter.

int MenuCmds _Start(void)

Use this function calls MenuCmds _Init().

_Bool MenuCmds_process (const menu_cmd menu_array[], uint8_t term_char);

Mandatory: Find a match to a MenuCmds element from the input Terminal then execute the assigned function. If no match found, the function returns.

Parameters:

menu_array[]	Pointer to the menu_cmd array to search/process
term_char	The terminal input data

Returns:
Boolean false = match not found
true = match found

Example:

```
      // Term_ is the TopDesign terminal component name.
MenuCmds _process(Term_GetChar());
```

menu_cmd *MenuCmds_search (const menu_cmd menu_array[], uint8_t term_char);

Find a match to a MenuCmds element from the input Terminal then return this element.

Parameters:

menu_array[]	Pointer to the menu_cmd array to search
term_char	The terminal input data

Returns:

menu_cmd element where the match was found.

if = 0; no match was found.

Example:

```
menu_cmd menu_ary_test [] =  { …allocated menu_cmd elements… };

      // Term_ is the TopDesign terminal component name.
MenuCmds_search(menu_ary_test, Term_GetChar());
```

int MenuCmds_num_func_SetCallback(void (*num_func_cb))

If a special function handling for the numbers 0 through 9 are needed, use this API to pass the pointer to the number handling function as a callbackback to the MenuCmds component.

If this function is not implemented or the parameter num_func_cb = NULL, number commands will be processed normally through the MenuCmds structure. If given a valid number handler, the number command will be processes through this handler as a callback in MenuCmds_process().

Parameters:

&num_func_cb	Pointer to the menu_cmd array to search

Returns:

error // 0 = No Error

Example:

```
void number_handler { …  number handling code …};

MenuCmds_num_func_SetCallback (&number_handler);
```

The MenuCmds Structure

The MenuCmds component provides the functions for your project to process the MenuCmds structure that you supply. This structure is going to be unique to your project and the needs of the application to provide Terminal display information and Terminal contol of application operation.

The MenuCmds structure is one or more menu_cmd struct arrays. You can create the MenuCmds structure as a single-level menu list or you can create multiple levels (up to 4) of sub-menus to logically group menu commands for better viewability in a help command. Additionally submenu commands can inherit some of the properties of the sub-menu such as “password protection” and “hidden”

A top-level menu list must be created and passed into the MenuCmds_process() function.

An simple example of a short menu list:

```
const menu_cmd menu_array_L1[] = {
MENU_HELP_ALLOC('?', "display FULL help list",NULL,0),
MENU_HELP_ALLOC('h', "display minimized help list",NULL,0),
MENU_FUNC_ALLOC('b', "display Build info",cmd_build_info,0),
MENU_HELP_ALLOC('G', "String_Funcs Test commands",menu_array_getstring,MENU_HELP_ALWAYS),
MENU_END_ALLOC,
};      // This list could grow.
```

To simplify the coding of the MenuCmds structure, a few macros were created.

`MENU_END_ALLOC`

Keep this "NULL" element as the last element. This element is used to signal the end of the menu list. **This is a MANDATORY last element in any menu list array.**

`MENU_FUNC_ALLOC(cmdchar, text, func, flags)`

Macro to simplify menu cmd Functions. To construct a menu cmd element that performs a function if the “menu char” inputted matched, here is the general format. In this example “cmd_build_info” is the pointer to the function you provide.

```
Menu list      menu     menu cmd           menu cmd      optional
function macro char    description       function call   attributes
     |           |          |                  |          |
     v           v          v                  v          v
MENU_FUNC_ALLOC('b', "display Build info",cmd_build_info,0),
```

For example:

```
MENU_CMD_PROTO(cmd_build_info) {… your code here … };
```

```
MENU_FUNC_ALLOC('b', "display Build info",cmd_build_info,0),
```

MENU_HELP_ALLOC(cmdchar, text, func, flags)

Macro to simplify SubMenus**.** To construct a menu cmd element that defines a submenu if the "menu char" inputted matched, here is the general format. In this example "menu_array_getstring" is the pointer to the next menu_cmd struct array you provide. Note: The menu_cmd[] ptr is required with the exception that if a "NULL" is passed, it is assumed to be a help menu display.

```
Menu list      menu     menu cmd                        menu cmd        optional
help macro     char    description                   menu_cmd[] ptr     attributes
     |           |          |                             |                |
     v           v          v                             v                v
MENU_HELP_ALLOC('G', "String_Funcs Test commands",menu_array_getstring,MENU_HELP_ALWAYS),
```

For example:

```
const menu_cmd menu_array_getstring[] = {
MENU_FUNC_ALLOC('d', "unsigned Decimal", cmd_gs_udec,0),
MENU_FUNC_ALLOC('D', "signed Decimal", cmd_gs_sdec,0),
… more elements …
MENU_END_ALLOC,
};

MENU_HELP_ALLOC('G', "String_Funcs Test commands",menu_array_getstring,MENU_HELP_ALWAYS),
```

MENU_CMD_PROTO(func_name)

Simplified define of a function call for use with the menu cmd.
For example:

```
MENU_CMD_PROTO(cmd_build_info) {… your code here … };
```

MenuCmds Attributes

The following menu_cmd element attributes are **mandatory**:

MENU_FUNC
If MENU_FUNC_ALLOC macro is used, this attribute is already set. No need to add it.

MENU_HELP
If MENU_HELP_ALLOC macro is used, this attribute is already set. No need to add it.

The following menu_cmd element attributes are **optional**:

MENU_HELP_ALWAYS

The help menu command '?' will always display all menu commands and descriptions unless the 'hidden' attribute is set.
The help menu command 'h' will only display the top-level menu cmds and descriptions UNLESS the MENU_HELP_ALWAYS is added to the menu_cmd element. Then the next level submenu elements are displayed in the help menu commands
For example:

```
MENU_HELP_ALLOC('G', "String_Funcs Test commands",menu_array_getstring,MENU_HELP_ALWAYS),
```

MENU_PROT0, MENU_PROT1, MENU_PROT2, MENU_PROT3,

You can elect to password protect a menu cmd function or submenu. We provide up to 4 password protections. (Although unusual, you can use more than password protections on a single element.)

This is convenient for protecting menu cmds such as functions that control calibration access. Only after the correct password is entered, will this menu cmd be executed. If you protect a submenu, the submenu elements are password protected without having to add the protection attribute to all the elements of the submenu (inheritance).

Note: If you password protect any element, you must provide another menu cmd to unlock or relock the password to access this element.

If a menu cmd element is protected, a "[P]" before the menu description will appear in the help display until the protection is unlocked.
For example:

```
MENU_FUNC_ALLOC('k', "Calibrate ADC gain", cmd_cal_adc_gain, MENU_PROT0),
MENU_HELP_ALLOC('k', "Calibrate menu", menu_cal, MENU_PROT0 | MENU_HELP_ALWAYS),
```

MENU_HIDDEN

If you wish to hide a menu cmd from the help display, set this attribute. In this mode, the application user must know or unintentionally type in this menu cmd char to activate it.
For example:

```
MENU_FUNC_ALLOC('d', "unsigned Decimal", cmd_gs_udec, MENU_HIDDEN),
```

MenuCmds Default Menu Cmd Chars

The following characters are reserved for menu cmd processing:

- '?' => Display the MenuCmds Structure menu cmd chars and descriptions (FULL). All submenu elements are displayed unless the MENU_HIDDEN attribute is set.
- 'h' => Display the MenuCmds Structure menu cmd chars and descriptions (minimum). Only the top-level menu is displayed unless the MENU_HELP_ALWAYS is set in the submenu element is set.
- 'b' => Display the build information. This is not required but useful. An example function of the menu cmd is proved in the demo examples provided with this component library.

Example top-level menu_cmd array:

```
const menu_cmd menu_array_L1[] = {
MENU_HELP_ALLOC('?', "display FULL help list",NULL,0),
MENU_HELP_ALLOC('h', "display minimized help list",NULL,0),
MENU_FUNC_ALLOC('b', "display Build info",cmd_build_info,0),
… additional elements …
MENU_END_ALLOC,
};      // This list could grow.
```

MenuCmds Structure Limitations

The MenuCmds component does not check for menu cmd char redundancies. The first one encountered in the MenuCmds Structure is operated on.

IMPORTANT!: The following MUST be the last element in the menu list array. This is used by the MenuCmds code to signal the end of that menu level. If this element is missing, the results will be unpredictable.

The MenuCmds component code does not block in and of itself. However a call to a menu cmd function can block if the designer does not provide non-blocking implementation if needed.

It is possible to create a submenu circular reference. Avoid this. The implementation does prevent the submenu levels to 4 which will prevent infinite recursiveness.

Resources

The MenuCmds component utilizes some FLASH and SRAM resource resources. If you do not use all MenuCmds functions and you optimize your builds, unused functions will be automatically not included in the object file.

Sample Firmware Source Code

Two demo projects are included with this component library to demonstrate each of the supported functions of this component. I recommend using these demos as instructional examples.

String_Funcs_Demo uses the MenuCmds component in a simplified two-level MenuCmds Structure. It does not use all the features and attributes but is a common implementation of MenuCmds.

MenuCmds_Demo uses the MenuCmds component in a multi-level MenuCmds Structure. It uses all of the features and attributes.

The default device for this project is a PsoC5 and configured for the CY8CKIT-059 board. The project can be modified to work with other PsoC devices. It also uses the embedded instance of this component in the Term component.

The MenuCmds component is also used in this demo project to provide a easy to use single character menu command structure to execute different features.

Component Changes

Version	Description of changes	Reason for changes/impact
3.3.3	Component released version	Placed code in a shareable component.
3.0	first release of the component	Non-component version.

References

MenuCmds_Demo_v3_1_0.pdf	String_Funcs_Demo_v2_1_0.pdf
Term Component: Term_v2_2.pdf	
How to Access Custom Libraries and Components.pdf	