Chapter

7

# *Cmicro Targeting Tutorial*

**This tutorial takes you through the first steps of targeting. Currently this tutorial is designed for using a Borland C or a Microsoft Visual C compiler in Windows, and gcc or cc on UNIX.**

# Prerequisites / Abbreviations Used

Before you run this tutorial, you should be familiar with the SDL Suite tools, especially the Organizer, the SDL Analyzer and the SDL Simulator. If you have not already done so, you are recommended to go through the previous tutorials in this volume.

The following notations and directories concern the rest of this tutorial:

- `<TAUinstallation>` denotes the Telelogic Tau installation directory, which is called `<systemdrive>:\Telelogic\SDL_TTCN_Suite4.5` **in Windows** and `$telelogic` **on UNIX.**
- In this tutorial there is a mixed use of the path separation characters '/' and '\', as several steps **in Windows** and **on UNIX** only differ in these.
- Although "directories" are sometimes called "folders" **in Windows**, this tutorial always uses the expression "directory".

  You will find this tutorial placed in the directories:

  ```
  <TAUinstallation>\sdt\examples\cmicrotutorial\wini386
  <TAUinstallation>/sdt/examples/cmicrotutorial/sunos5
  <TAUinstallation>/sdt/examples/cmicrotutorial/hppa
  ```

- The Cmicro Library can be found in the directories:

  ```
  <TAUinstallation>\sdt\sdtdir\wini386\cmicro
  <TAUinstallation>/sdt/sdtdir/sunos5sdtdir/cmicro
  <TAUinstallation>/sdt/sdtdir/hppasdtdir/cmicro
  ```

  For a description of the Cmicro Library please view chapter 67, *The Cmicro Library, in the User's Manual*.

# Introduction

## General

This tutorial is divided into three sections. In the first section you will take a small SDL system and generate it with the SDL Analyzer. You will learn about configuration possibilities using the Targeting Expert and how to create environment functions. Finally, you will build the target application.

In the second section you will test it and learn something about how to use the SDL Target Tester.

In the third section you will learn how to remove the Target Tester source from the target application.

## Integrations

Targeting may be implemented using the following methods:

- Bare integration:

  The SDL system is running on a bare target, scheduled by the Cmicro Kernel without any other operating system (OS).

- Light integration:

  The SDL system (scheduled by the Cmicro Kernel) is running as one task in an OS, possibly using functions of the OS.

- Tight integration:

  All the SDL process instance sets (and other tasks) are scheduled by the OS of the target.

In this tutorial you will be doing a light integration as the target application is executed as an independent OS task, where the SDL processes are scheduled by the Cmicro Kernel.

## Target Tester Communication

The communication between the Target Tester and the target application is done using sockets (localhost, port 9000 as default) in this tutorial.

# Prerequisites to the Example

## The Pager System

The SDL system that will be used for this tutorial is a pager system. A pager is a small hand-held device used for contacting people. It contains a radio receiver which is capable of receiving signals on a certain frequency consisting of short messages and telephone numbers.

The pager has also a sort of databank with a limited capacity for storing messages as well as a keypad and a display which serve as the interface to the user. The user has the option of scrolling through, reading and deleting the messages that are displayed on the small screen.

The keypad consists of three buttons; one for scrolling to the right, one for scrolling to the left and one for deleting. The pager emits a sound when a new message has arrived and also when the user makes an error or tries to do something which is not allowed. For example, trying to delete a message when the databank is empty or scrolling too far in a certain direction would be instances of illegal actions. Naturally, the pager can only hold a certain amount of messages and therefore at some point eventually fills up.

When the pager has reached its capacity a warning message is given for 2 seconds before the received message is displayed.

The SDL Overview shows the pager system divided into blocks and processes.

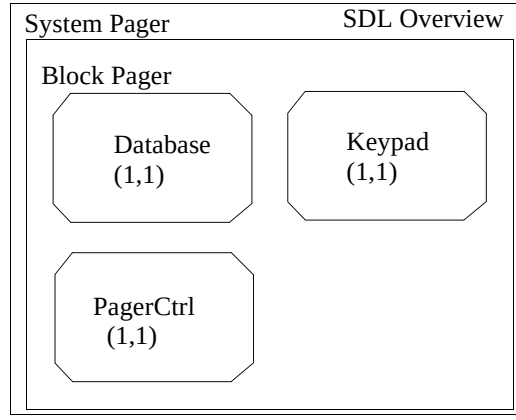


*Figure 186: An overview of the system Pager*

| Process | Description |
|---|---|
| `Database` | The process `Database` manages the array of messages that makes up the pager's memory. It can store messages, retrieve them and delete them while maintaining order in the databank. |
| `PagerCtrl` | `PagerCtrl` basically handles all the input and output of the system. It receives input from the user via the keypad, messages from the radio receiver and information from the database regarding the status of saving and deleting. |
| `Keypad` | The process `Keypad` converts the input from the user into a signal and sends it to `PagerCtrl`. |

## Delivered Files

The files needed for this tutorial can be found in the directory:

`<TAUinstallation>/sdt/examples/cmicrotutorial/<platform>/pager`

The project directory `pager` includes a sub directory called `system`. The directory `system` contains the SDL/GR files of the Pager system.

Furthermore, there is a directory `prepared` in parallel to the `system` directory. Here you can find an environment file `env.c` which can be

used if you are not interested in programming the environment on your own.

# Targeting

## Preparations - File Structure

1. Create a new empty directory `<cmicrotutorial>` in your home directory or on your local hard disk. This directory will be denoted by `<MyTutorial>` in the following.
2. Copy the directory `<TAUinstallation>/sdt/examples/cmicrotutorial/pager` (including all files and subdirectories) to your new `<MyTutorial>` directory and remove all write protections.
3. In the Organizer, open the Pager system (`Pager.sdt`) found in `<MyTutorial>/pager/system`.

## Using the Targeting Expert

1. Select the system symbol in the Organizer view.
2. Start the Targeting Expert from the *Generate* menu. The Targeting Expert will generate a default partitioning diagram model (see "Partitioning Diagram Model File" on page 2896 in chapter 60, *The Targeting Expert*) and will check the directory structure.
3. As the target directory specified in the Organizer does not exist yet, you will be prompted if it should be created. Press the *Yes* button.

A sub-directory structure is added in the target directory afterwards by the Targeting Expert. For further information see "Target Sub-Directory Structure" on page 2921 in chapter 60, *The Targeting Expert*.

> **Note:**
> If the Targeting Expert is started the very first time a welcome window is displayed. Just press the *Close* button and proceed. The welcome window will be shown any time you start the Targeting Expert again until you select the "Do not show again at startup" check box.

The work flow of the Targeting Expert is divided into four steps.

- Step 1: Select the Desired Component
- Step 2: Select the Type of Integration
- Step 3: Configure the Build Process
- Step 4: Make the Component

The very first time you are using the Targeting Expert, an assistant is automatically started showing you how to proceed. When you have closed the assistant, you can always re-start it by choosing the menu option *Help > Assistant.*

The Help Viewer will be displayed and show the appropriate manual page if you click on one of the numbered boxes.

## Step 1: Select the Desired Component

- Click on the component in the partition diagram model.
  The complete SDL system is (per default) generated into the component “component”.

> **Hint:**
>
> The component can be given any name you like if the system is deployed using the Deployment Editor. For more detailed information on how to deploy a system, see chapter 41, *The Deployment Editor, in the User’s Manual*.

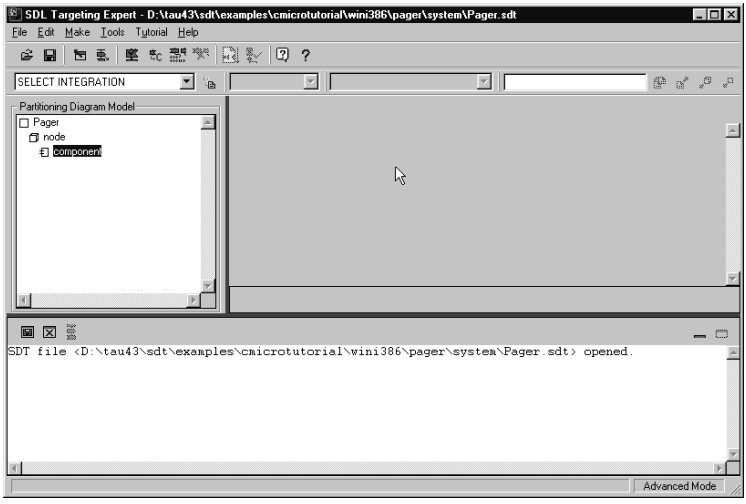


*Figure 187: The Targeting Expert's main window*

### Background Information

- More details about partitioning diagram models can be found in "Partitioning Diagram Model File" on page 2896 in chapter 60, *The Targeting Expert*.
- For further information concerning the selectable entries in the partitioning diagram model, please see "Targeting Work Flow" on page 2852 in chapter 60, *The Targeting Expert*.

## Step 2: Select the Type of Integration

1. Press the left most combo box in the integration tool bar of the main window (or click the component entry in the partitioning diagram model using the right mouse button). This is shown in Figure 188.

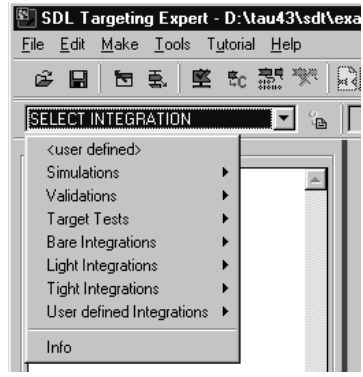


*Figure 188: Popup menu in the Targeting Expert*

2. A tree structure containing all the pre-defined integrations is shown in the popup menu displayed. Select *Light Integrations* > *Application TEST*.

3. As there is no automatic configuration of the pager system available so far, you are prompted to start the automatic configuration. Press the *Yes* button in the dialog. The Targeting Expert then automatically starts the Cmicro SDL to C compiler to generate the file `sdl_cfg.h`.

The SDL system is now checked for correctness and the automatic configuration is done.

> **Hint:**
>
> Although only the file `sdl_cfg.h` is needed the SDL to C compiler generates the files: `component.c`, `component.ifc`, `component.sym` and `sdl_cfg.h`. This is done because only the SDL to C compiler knows all the used and unused features (OO concept).

After the SDL to C compiler has finished, the Targeting Expert:

- generates the file `env.c` and lists all the SDL signals from and to the environment (see event log). A manual adaptation is needed for each signal before the generated files and the library files can be compiled. This is necessary as the Targeting Expert only generates a skeleton with the used signals.

  How to edit the `env.c`, is described in "Edit the Environment File" on page 279.

- generates default Target Tester options (file `sdtmt.opt`)
- generates a default manual configuration (file `ml_mcf.h`)

> **Hint:**
>
> After the integration has been selected the Targeting Expert automatically sets the default compiler. The default compiler's name is taken from the Telelogic Tau Preferences.
>
> If a different compiler is required than the one set as default in the preferences, it is possible to change this in the integration tool bar's combo box.

### Background Information

If you want to do an integration not given in the Tau distribution, please select the entry <*user defined*> in the integration popup menu. Then you are able to do all settings needed for the used hardware. You are also able to set up your own integration accessible in the integration popup menu.

The contents of the files:

- `component.c`
  Describes the SDL system's behavior in C functions.
- `component.ifc`
  The header file for the environment functions. E.g. it provides PIDs and type definitions for signals.
- `component.sym`
  Provides information on SDL symbols. Necessary for tracing the SDL system with the SDL Target Tester.
- `sdl_cfg.h`
  The automatic configuration file for the Cmicro Kernel. For in-

stance, if you use a timer, the file contains a define, which turns the timer implementation on.

### Edit the Environment File

In this section you will learn how to fill in the environment functions in the file `env.c`.

**Hint:**

There is also a prepared `env.c`. You can copy it from directory: `<MyTutorial>/pager/prepared` into `<MyTutorial>/pager/target/pager._0/Application_TEST`.

Do not forget to remove the write protection!

Note the differences between the prepared and generated file.

**Note:**

In the following the Targeting Expert starts a text editor. Per default the built-in editor is used. This can be changed in the *Tools* > *Customize* menu.

1. Select the menu *`Edit > Edit Environment File`* to open the file `env.c`

**Caution!**

In the file `env.c`, you should only edit code between the lines:

```
/* BEGIN User Code ... */
/*   END User Code ... */
```

The reason is:

If the Targeting Expert needs to generate the file a second time, the code in these sections will be read in and copied to the new file. Only the code between the mentioned lines will be unchanged.

Do NOT edit lines with the text:

```
/* BEGIN User Code ... */
/*   END User Code ... */
```

2. Find the lines from the global section:

```
/* BEGIN User Code (global section)*/
/* It is possible to define some global variables here */
```

```
/* or to include other header files. */
```

This tutorial describes a console application. It will use the screen and the keyboard to communicate with the user. It is necessary to include the used header file(s).

For a better style some defines are used. Further some global variables and functions have to be implemented. The functions are called, if there should be something simulated in the environment, like a display. After the line

`/* or to include other header files. */`

the following code needs to be inserted:

```
#if defined(BORLAND_C) || defined(MICROSOFT_C)
  #include <conio.h>
#else
  #include <stdio.h>
#endif

#define key_was_pressed 1
#define key_not_pressed 0

int  KeySignalPresent  = 0;
char LastKeyPressed;
```

**xInitEnv()**

3. We like to have a welcome message displayed when the system is started. This can be done like this in the function `xInitEnv()`

```
printf("-------- Welcome to Pager system--------\n\n");
printf("get message :  0 to 4\n");
printf("scroll right:  r\n");
printf("scroll left :  l\n");
printf("delete      :  d\n\n");
```

The code must be inserted between

```
/* BEGIN User Code (init section) */
```

and

```
/*   END User Code (init section) */
```

**xInEnv()**

4. Now you have to handle the data from the environment. In this tutorial it means you have to handle the input from the keyboard!

**Caution!**

Do not use blocking functions in the environment file.

The environment is polled with every cycle of the Cmicro Kernel. That is the reason why it is not allowed to use blocking functions like `getchar()`. These kind of functions stop the kernel and so it cannot process the SDL system.

The possible handling of the data:

If a key has been pressed the digits 0-4 are recognized as a message and the letters 'r', 'l' and 'd' are the commands for scrolling and deleting.

Please go to the code position of the function `xInEnv()` with the following lines:

```
/* BEGIN User Code (variable section)*/
/* It is possible to define some variables here      */
/* or to insert a functionality which must be polled */
```

Below these lines insert the following code:

```
char my_inkey;

KeySignalPresent = key_not_pressed;

#if defined(XMK_UNIX)
  my_inkey = 0;
  if((my_inkey = getchar_unlocked()) != 0)
  {
    KeySignalPresent=key_was_pressed;
  }
#elif defined(BORLAND_C) || defined(MICROSOFT_C)
  if (kbhit())
  {
    my_inkey=_getch();
    KeySignalPresent=key_was_pressed;
  }
#endif

if (KeySignalPresent==key_was_pressed)
{
  if ((my_inkey == 'r') ||
      (my_inkey == 'l') ||
      (my_inkey == 'd') ||
      ((my_inkey>='0')&&(my_inkey<='4')))
    LastKeyPressed = my_inkey;
  else
    LastKeyPressed=0;
}
else
 LastKeyPressed = 0;
```

5. Find the following lines of code in the function `xInEnv()`

```
/* BEGIN User Code <ScrollRight>_1 */
  if (i_have_to_send_signal_ScrollRight)
/*   END User Code <ScrollRight>_1 */
```

In step 2 we implemented the variable `LastKeyPressed`. In step 4 we assigned it the value of `my_inkey` which has the value of the last key pressed. Modify the `if()` statement into:

```
if (LastKeyPressed == 'r')
```

6. Find the following line of code in the function `xInEnv()`

```
GLOBALPID(Who_should_receive_signal_ScrollRight,0));
```

The process type ID which should receive the signal `ScrollRight` needs to be inserted. To get an overview of the process type IDs look at the dialog window that has pop-ed up by the Targeting Expert. All the used process type IDs are given here.

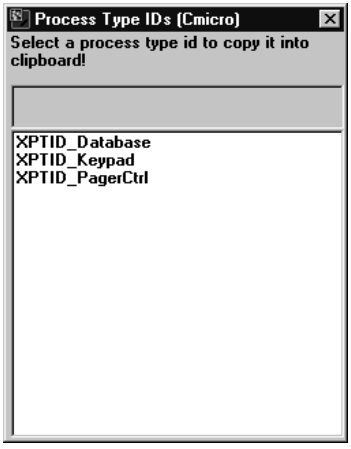


*Figure 189: Process type ID dialog*

Select the `XPTID_Keypad` entry in this dialog to copy it into the clipboard. Then paste it into the `env.c` as shown below.

```
GLOBALPID(XPTID_Keypad,0));
```

The function returns, and the signal is treated.

7. Find the following lines in the function `xInEnv()`

```
/* BEGIN User Code <ScrollLeft>_1 */
  if (i_have_to_send_signal_ScrollLeft)
/*   END User Code <ScrollLeft>_1 */
```

Modify the `if()` statement to:

```
if (LastKeyPressed == 'l')
```

8. Find the following line of code in the function `xInEnv()`

```
GLOBALPID(Who_should_receive_signal_ScrollLeft,0));
```

Copy the `XPTID_keypad` as described in step 6.

```
GLOBALPID(XPTID_Keypad,0));
```

9. Find the following lines of code in the function `xInEnv()`

```
/* BEGIN User Code <Delete>_1 */
  if (i_have_to_send_signal_Delete)
/*   END User Code <Delete>_1 */
```

Modify the `if()` statement to:

```
if (LastKeyPressed == 'd')
```

10. Find the following line of code in the function `xInEnv()`

```
GLOBALPID(Who_should_receive_signal_Delete,0));
```

Copy the `XPTID_keypad` as described in step 6.

```
GLOBALPID(XPTID_Keypad,0));
```

11. Because we do not use a real target hardware, but simulate the Pager system, we have to predefine some messages.
Find the following lines of code in the function `xInEnv()`

```
/* BEGIN User Code <ReceivedMsg>_1 */
if (i_have_to_send_signal_ReceivedMsg)
/*   END User Code <ReceivedMsg>_1 */
```

Modify the `if()` statement to:

```
if ((LastKeyPressed>='0')&&(LastKeyPressed<='4'))
```

This if-statement checks whether the key hit on the keyboard was one of the defined keys or not.

12. Go to the next empty "`User Code`" section and insert following lines:

```
char *p;
  xmk_var.Param1.MyText = (SDL_Charstring)NULL;

  switch(LastKeyPressed)
  {
    case '0':
         p = " Hello user";
         xmk_var.Param1.TelNumber = 12345;
         break;
    case '1':
         p = " How do you feel doing targeting?";
         xmk_var.Param1.TelNumber = 555555;
         break;
    case '2':
         p = " Targeting is all so easy!";
```

```
            xmk_var.Param1.TelNumber = 987654;
            break;
        case '3':
            p = " I only wanted to check if it works.";
            xmk_var.Param1.TelNumber = 45454;
            break;
        case '4':
            p = " ... and it works very fine!";
            xmk_var.Param1.TelNumber = 911911;
            break;
        default :
            break;
    }
    xAss_SDL_Charstring(&(xmk_var.Param1.MyText), p,
XASS_AC_ASS_FR);
```

In this part the messages to the equivalent numbers 0-4 are stored. With the `switch` statement it is decided which one is handed over to the environment.

**Note:**

This way of "receiving" messages is of course just a helper function because we do not use a real interface here!

The line `xmk_var.Param1.MyText = (SDL_Charstring)NULL;`

means that the element `MyText` of the parameter `message` which is a parameter of the signal `ReceivedMsg` is set to null.

The Signal `ReceivedMsg` and the parameter `message` are declared in the SDL system.

The line `xAss_SDL_Charstring(&(xmk_var.Param1.MyText), p, XASS_AC_ASS_FR);` allocates memory for the pointer `p`.

**Note:**

If an SDL charstring is mapped to an array of char in C, the first character in this array (index 0) is for internal use only, i.e. the text message should start at index 1. This is done by having a space in front of the text in the implementation shown above.

13. Find the following line of code in the function `xInEnv()`

```
GLOBALPID(Who_should_receive_signal_ReceivedMsg,0));
```

Modify the statement as showed below.

```
GLOBALPID(XPTID_PagerCtrl,0));
```

### xOutEnv()

**Caution!**

The function `xOutEnv()` provides a pointer named: `xmk_TmpDataPtr`. The data referenced by this pointer is valid only as long as the function `xOutEnv()` is processed.

If you need to treat the data after leaving the function, copy it to variables defined by you.

14. Find the following code section:

```
case CurrentMsg :
          {
            /* BEGIN User Code <CurrentMsg>_1 */
/* Use (yPDP_CurrentMsg)xmk_TmpDataPtr to access
the signal's parameters */
/* ATTENTION: the data needs to be copied.  Otherwise it */
/* will be lost when leaving xOutEnv */
            /*   END User Code <CurrentMsg>_1 */

            /* BEGIN User Code <CurrentMsg>_2 */
            /* Do your environment actions here. */
            xmk_result = XMK_TRUE; /* to tell the caller that
*/
                                   /* signal is consumed
*/
            /*   END User Code <CurrentMsg>_2 */
          }
```

This code fragment handles the signal `CurrentMsg`. The chosen message is displayed on the screen (telnumber, message, current message position and the total number of messages). Insert the following code after: `/* Do your environment actions here. */`

```
printf("\r                                                    ");
printf( "\rCurrentMessage:  %6d  %s   (%d/%d)",
      ((yPDP_CurrentMsg)xmk_TmpDataPtr)->Param3.TelNumber,
      ((yPDP_CurrentMsg)xmk_TmpDataPtr)->Param3.MyText+1,
      ((yPDP_CurrentMsg)xmk_TmpDataPtr)->Param1,
      ((yPDP_CurrentMsg)xmk_TmpDataPtr)->Param2);
xFree(&(((yPDP_CurrentMsg)xmk_TmpDataPtr)->Param3.MyText));
```

The line

```
xFree(&(((yPDP_CurrentMsg)xmk_TmpDataPtr)->Param3.MyText));
```

frees the memory allocated in the kernel when sending the signal.

15. Find the following code in the function `xOutEnv()`:

```
case ServiceMsg :
  {
    /* BEGIN User Code <ServiceMsg>_1 */
    /* Use (yPDef_Close*)xmk_TmpDataPtr to access
       the signal's parameters */
    /* ATTENTION: the data needs to be copied.
                  Otherwise it */
```

```
    /*                will be lost when leaving xOutEnv */
    /*   END User Code <ServiceMsg>_1 */

    /* BEGIN User Code <ServiceMsg>_2 */
    /* Do your environment actions here. */

    xmk_result = TRUE; /* to tell the caller that */
    /* signal is consumed      */
    /*   END User Code <ServiceMsg>_2 */
  }
```

The code fragment handles the signal `ServiceMsg`. The handling of the data is same as in the step before. So insert the following code after: `/* Do your environment actions here. */`

```
printf("\r                                                    ");
printf( "\rServiveMessage: %s",
       ((yPDP_ServiceMsg)xmk_TmpDataPtr)->Param1+1);
xFree(&(((yPDP_ServiceMsg)xmk_TmpDataPtr)->Param1));
```

16. Find the following code section in the function `xOutEnv()`:

```
case ShortBeep :
     {
       /* BEGIN User Code <ShortBeep>_1 */
       /*   END User Code <ShortBeep>_1 */

       /* BEGIN User Code <ShortBeep>_2 */
       /* Do your environment actions here. */
          xmk_result = XMK_TRUE; /* to tell the caller that */
                                 /* signal is consumed      */
       /*   END User Code <ShortBeep>_2 */
     }
     break ;
```

The code fragment handles the signal `ShortBeep`. A beep sounds when the pager receives a message or you do something which is not allowed. After the code `/* BEGIN User Code <ShortBeep>_1 */` insert

```
putchar(07);
```

17. Find the following code section in the function `xOutEnv()`:

```
case LongBeep :
     {
       /* BEGIN User Code <LongBeep>_1 */
       /*   END User Code <LongBeep>_1 */

       /* BEGIN User Code <LongBeep>_2*/
       /* Do your environment actions here. */
          xmk_result = XMK_TRUE; /* to tell the caller that */
                                 /* signal is consumed      */
       /*   END User Code <LongBeep>_2*/
     }
     break ;
```

Insert the following code after

`/* BEGIN User Code <LongBeep>_1 */`

```
putchar(07);
```

```
putchar(07);
```

### Closing the Environment

In this tutorial there is no need to close the environment. In other cases, e.g. microprocessor hardware, it is probably necessary to do so.

Have a look in the file `env.c`, find code like this in the function `xCloseEnv()`:

```
/* BEGIN User Code (close section) */
/* Do the actions here to close your environment */
/*   END User Code (close section) */
```

Insert any code you need to have here.

## Step 3: Configure the Build Process

1. Press the items below the `Application TEST` in the partitioning diagram. If it is necessary to add or remove settings for your job, you can edit the settings.
2. Click *Save* to close the dialog.

For this section of the tutorial there it is not necessary to modify anything, though we will do some modifications later in section "Run Target EXE without Tester" on page 294.

### Background Information

Short description of the different areas:

- *Compiler / Linker / Make*

  In this area it is possible to configure all the settings used for the Compiler, Linker and Make tools.

  Something special regarding *Additional Compiler*. For example, you have to use an ANSI C- compiler to compile the generated files, and it is necessary to link the objects with the object from one file, which needs to be compiled with a C++ compiler. In this instance you could enter the regarding file and the used compiler in the section *Additional Compiler*.

  For more information see: "Configure Compiler, Linker and Make" on page 2856 in chapter 60, *The Targeting Expert*.

- *Target Library*

In this area it is possible to set defines and values to scale the target library. For more information see "Configure and Scale the Target Library" on page 2872 in chapter 60, *The Targeting Expert*.

All settings will be stored in the file `ml_mcf.h`.

- *Target Tester*

  In this area it is possible to set defines and values to scale the functionality of the tester. For more information see "Configure the SDL Target Tester (Cmicro only)" on page 2873 in chapter 60, *The Targeting Expert*.

  All settings will be stored in the file `ml_mcf.h`.

- *Host Connection*

  In this area it is possible to set the parameter of the connection to the host. For example, it contains the description of the message coding and the name of the executable, etc. The configuration of the *Host Connection* is always stored in the file `sdtmt.opt`. This file is mandatory for the SDL Target Tester. For more information see: "Configure the Host (Cmicro only)" on page 2874 in chapter 60, *The Targeting Expert*.

## Step 4: Make the Component

1. In the dialog which is displayed by default (when the `Application TEST` is selected) you have to select two check boxes. *Analyze / Generate Code* and *Environment functions*.

2. Click on the button *Full Make* to start the code generation and make.

After the SDL to C compiler has finished the code generation, the Targeting Expert will re-generate the `env.c` (and keep your modifications). Afterwards it generates a makefile with the given settings and the code will be compiled and linked.

The Targeting Expert then starts the SDL Target Tester.

# Use of the SDL Target Tester

## Differences between SDL Simulator and SDL Target Tester

The difference compared to the SDL Simulator is that the generated SDL system is running on a target hardware and is sending messages to the host system on which the SDL Target Tester's host part is running.

In this Cmicro tutorial, the SDL Target Tester's host part is running on the host as well as the generated SDL system (target).

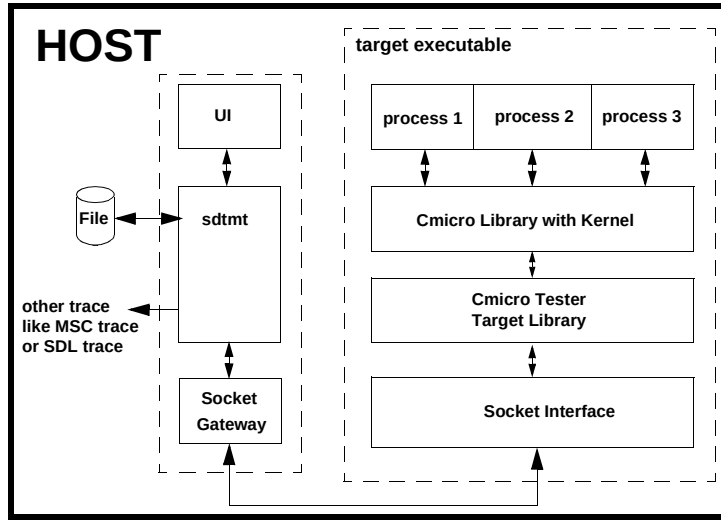


*Figure 190: Communication links between the processes*

## Restrictions

### In this Tutorial

It is possible to use all the features supported by Cmicro (e.g. signal priorities, error checks, tester) except the preemptive Cmicro Kernel. This is due to the concept behind Cmicro which is designed for small targets on a stand-alone hardware (bare integration).

### Background Information

- To get information on how it works with a real target hardware, see chapter 67, *The Cmicro Library, in the User's Manual*.
- If you want to edit the makefile, or have a look into it, please use the menu *Edit > Makefile* in the Targeting Expert. For more information please view chapter 60, *The Targeting Expert, in the User's Manual*.

## Testing the Pager System

### Running the SDL Target Tester

The SDL Target Tester could be started automatically from the Targeting Expert or by:

- selecting *Tools > SDL > SDL Target Tester* in the menu of the Targeting Expert.
- pressing the quick button in the quick button bar.

After it has been started the following steps have to be done (in the given order)

1. Start the communication with the executable by pushing the button *StartGateway* in the *Communication* group or with the menu *Execute > StartGateway*.
2. Go back to the Targeting Expert and select the menu entry *Tutorial > Start target (Windows)* or respectively *Tutorial > Start target (UNIX)*

**Hint:**

The menu *Tutorial* is a configurable menu not available if working on other SDL systems. Please refer to "Configurable Menus" on page 2831 in chapter 60, *The Targeting Expert, in the User's Manual* for more information on how to create your own menu entries.

**Note:**

**In Windows** the target application is started in a separate command prompt window.

**On UNIX** the target application is started in a separate xterm.

3. Now the communication between SDL Target Tester and target is established. The SDL Target Tester displays the message `Start with "Go Forever"`. The corresponding button is located in the *Execute* group. Press the button.

   All messages of the target application are displayed in the UNIX shell or DOS command prompt where it was started. The pager will display its start-up message which was implemented in "xInitEnv()" on page 280.

4. You can "receive" messages by pressing the keys *0, 1, 2, 3, 4* and scroll with the keys *r* and *l*. To delete a message press *d*.
   See "xInEnv()" on page 280 on how the environment was implemented.

5. The pager displays the last received message, the number of the current message and the total number of received messages. Now you can scroll, delete and receive new messages. If you receive a new message while the maximum amount of messages(3) is reached, the pager saves the message temporary and displays the warning: `Memory full, please free memory to get new messages` for about 2 seconds.

6. Press the d key on your keyboard, the last message is then deleted and the received message is displayed.

**Note:**

To exit the target application press `CTRL+C` on the keyboard.

## SDL Target Tester Commands

The SDL Target Tester has button groups. Each button represents a Tester command. You can use the buttons or the command line at the bottom of the Tester to enter a command. If you enter `help` in the command line you will see a list of SDL Target Tester commands.

In the following some Tester commands are explained briefly. For more information, see chapter 68, *The SDL Target Tester*.

### Tracing the SDL System -> MSC Editor

Similar to the SDL Simulator GUI it is possible to generate MSC traces while testing the system with the SDL Target Tester.

- In the *Trace* group you can select the *Start MSC* button, for example, to start the MSC trace.
- Now you can select between output via display or output to a file.

#### Tracing the SDL System -> SDL Editor

It is possible to trace the target system with the SDL Editor.

- You can start the trace with the command line of the SDL Target Tester by typing **`start-sdle`**, or by using the button *Start SDLE*.

The MSC trace and SDL trace functions are powerful tools for understanding the system.

#### Target Information

To get more information about target configuration, you must open the *Configuration Group* in the button area of the SDL Target Tester and press *Target* to get the current target configuration.

To get information about the kernel, you have to open the *Examine Group*. If you press *Queue* you will get information about the current state of the internal queue of your system. You will see the peak hold and the amount of signals of your current system. By pressing the other buttons in the *Examine Group*, you will get more information of the running system.

#### Memory

With the `?memory` command you can see how the current memory state is.

1. Start the Pager system as described in "Running the SDL Target Tester" on page 290.
2. Type `?memory` (the short command `?m` can also be used) on the command line. You can check the memory pool size, the current memory fill and so on.

Now we will see how the memory is handled in the Pager system. Notice that the current amount of blocks in pool is four and the peak hold is five.

3. Switch to the target application and press a key (1-4) to get a message.

4. Go back to the Target Tester and execute the `?m` command again. This time you can see that two more blocks are allocated. If you delete the last message the memory should be freed again and show four blocks.

#### Breakpoints and Queues

To debug the system you can use the *Breakpoints* button group. You can set a breakpoint on a signal input or a process state. If a breakpoint was reached you can continue the system with the button *Continue*.

1. Restart the target as described in "Running the SDL Target Tester" on page 290.
2. Expand the `Breakpoints` group and select `Break input`.
3. Now you can choose a process ID. In this example we take the `Keypad` ID.
4. A signal ID list is shown afterwards, select the `delete` signal.
5. `Breakpoint on input is set` is shown in the text area now, switch to the target application window and insert some messages first, then press `d`. Nothing happens.
6. Switch to the Tester again. Now you can use the `?memory` command or look how the `Queue` looks like (`?queue`).
7. Press `Continue` after you have examined the system state.

As the system will be halted every time the signal `delete` is to be consumed in process `Keypad`, it is probably useful to delete the breakpoint(s) by entering the command `BA`.

# Run Target EXE without Tester

It is also possible to run the target executable without starting the Target Tester. Following has to be done first:

1. Exit the target application and the Target Tester if not already done.
2. Switch back to the Targeting Expert.

In the following there is a description what has to be done to remove the Target Tester source code form the target application.

1. Press the entry `Target Tester` below `Application TEST` in the Partitioning Diagram Model. As you can see all the selected Target Tester flags are disabled, i.e. it is not possible to switch them off because the pre-defined integration selected prevents doing so.

   To get access to these flags, the Targeting Expert provides a so called "Advanced Mode". (For details see "Advanced Mode" on page 2850 in chapter 60, *The Targeting Expert, in the User's Manual*.

2. Select the menu entry *Tools* > *Customize* and a dialog pops up. Select the check box *Advanced Mode* and press the dialog's *OK* button.

> **Caution!**
>
> The Advanced Mode is now switched on each time you enter the Targeting Expert again. Make sure you switch off the Advanced Mode to take advantage of the restrictions given with all the other pre-defined integration settings.

3. In the dialog displayed when the `Application TEST` entry is selected in the Partitioning Diagram Model, you have to select the *`Execution`* tab. Press the *`None`* radio button in the *Test application* group box.
   This is done to disable the execution of the Target Tester after the target application has been successfully build.
4. Select `Target Tester` in the Partitioning Diagram Model and deselect the `Use the Target Tester` check box. Press *OK* in all following windows. (Depending flags are switched off). After the last

dialog has been executed, all check boxes on the *Tester* tab should be un-selected now.

5. Press the *Save* button below the dialog.

**Hint:**

A new manual configuration file `ml_mcf.h` is generated with all the Target Tester flags undefined.

6. Now press the `Make` button. (All the files are compiled again because the `ml_mcf.h` has been modified.)

When compile and link is completed, the target can be started via the menu *Tutorial* > *Start target* and can be used now without SDL Target Tester.

# Chapter 7 Cmicro Targeting Tutorial