Chapter 68 *The SDL Target Tester*

The SDL Target Tester is a tool which can be used to find errors and illegal or unwanted behavior within a target system. It allows the programmer to follow and to reproduce the execution flow of the SDL system (running on the target) on the host.

The strength of the SDL Target Tester is the minimum amount of storage it occupies on the target. With the exception of certain traces, the SDL Target Tester's memory requirements do not increase proportionally with system size. That means, that the memory amount is a compiler specific constant. If the memory is scarce it is also possible to reduce the memory by reducing the functionality.

In order to use the SDL Target Tester, the system must be developed with the Cmicro SDL to C Compiler (see chapter 66, *The Cmicro SDL to C Compiler*) and the Cmicro Library (see chapter 67, *The Cmicro Library*).

The host and target systems can be connected by any type of communications link, as any kind of communications drivers can easily be connected to the SDL Target Tester system.

Chapter 68 The SDL Target Tester

Introduction

The SDL Target Tester offers the following features:

- Tracing of internal and external SDL events with the Tracer
- Generating MSC Traces with the Tracer
- Trace of system errors
- Debugging facilities
- Profiling
- Open interface to connect other tools
- Scalable functionality
- Error reproduction using the Recorder

Note:
The SDL Target Tester's Record and Play functions are only available if a Cmicro Recorder license is available.

The SDL Target Tester will be executed in a distributed fashion, i.e. some parts of the SDL Target Tester run on a host machine, while other parts run together with the Cmicro Kernel and the SDL application on the target machine, for example a micro controller. A prerequisite is a communications link between the host and target systems.

The SDL Target Tester – An Overview

Prerequisites

The SDL Target Tester is an optional part of the Cmicro Package and is delivered when specifically ordered. It is an addition to the Cmicro Library and contains the following parts:

- the host tool chain (see “Using the SDL Target Tester’s Host” on page 3516)
- the target library (see “The Target Library” on page 3582)

The SDL Target Tester’s host executable is a tool chain which consists of the executables sdtmt, sdtmtui, sdtmpm and sdtgate and sockgate. The sdtgate module contains the V.24 and sockgate the TCP/IP implementation for connecting host and target. All open interface functionality is utilized here.

The target library is mandatory when the SDL Target Tester is ordered. It must be used together with the Cmicro Library on the target. By using the open interface, it is possible to connect user specific tools to the SDL Target Tester’s target library.

This diagram gives an overview of the SDL Target Tester functionality:

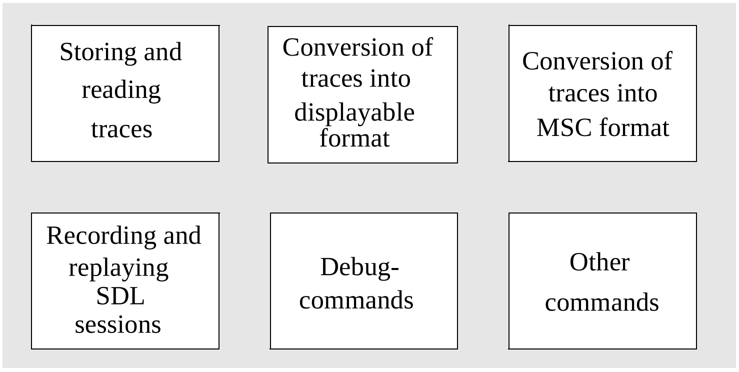

Storing and reading traces
Conversion of traces into displayable format
Conversion of traces into MSC format
Recording and replaying SDL sessions
Debug-commands
Other commands

Figure 589: Functionality of the SDL Target Tester

The following components, as seen from the user's point of view, represent the SDL Target Tester:

- The Cmicro Tracer - trace of SDL and system events.
- The Cmicro Recorder - recording and replaying SDL sessions.
- The command and debug interface.

The Cmicro Tracer is introduced in the subsection "Following the Execution Flow – The Cmicro Tracer" on page 3510.

The Cmicro Recorder is introduced following the subsection "Reproduction of Errors – The Cmicro Recorder" on page 3512.

The command and debug interface is introduced in the subsection "Commands for the Host and Debugging Facilities" on page 3515.

The following subsections give an overview of how to use the SDL Target Tester. A detailed description of the necessary preparations and adaptations will be given in the sections itself.

- "Using the SDL Target Tester's Host" on page 3516"
- "Graphical User Interface" on page 3562
- "The Target Library" on page 3582
- "Connection of Host and Target" on page 3594
- "More Technical Descriptions" on page 3636.

These sections follow after the introduction subsections.

Following the Execution Flow – The Cmicro Tracer

General

The Cmicro Tracer transmits information about the execution flow of the SDL system on the target. Information traced includes the signals sent and received by processes and the SDL states and symbols entered in execution flow. A filter can be defined so that only partial trace information is created for particular process instances or for particular SDL constructs. The tracer can be connected with the MSC Editor on the host, thus allowing observation of dynamic signal trace between processes in the system or from the environment to the system on the target.

Tracing SDL Events

The following SDL events can be traced.

State	The SDL nextstate operation
Input	The SDL input operation
Save	The SDL save operation
Output	The SDL output operation
Create	The SDL create operation
Stop	The SDL stop operation
Static Create	Used at system start, appears for each statically created process
Dynamic Create	Used at start of a dynamically created process
Decision	Trace of SDL action decision
Task	Trace of SDL action task
Procedure	Trace of SDL procedure call
Set	Trace of SDL action: Set timer
Reset	Trace of SDL action: Reset timer
Signalparams	Used to switch on/off trace of signal parameters
Implicit consump-tion	Trace of SDL implicit transition
Discard	Trace of SDL discard

For each symbol in each process in the system the trace can be separate-ly switched on or off.

Please consult the subsection “The API on Target” on page 3585 for in-formation about how to set the different options.

Tracing Other Events

The following system events can be traced.

Schedule	Used to trace scheduling events.
Error	Used to trace System errors, which are detected either within the generated code, within the Cmicro Kernel or within the SDL Target Tester.
Showprio	Used to trace the event, when the scheduling changes to a different priority level.

Each system event can be separately switched on or off.

Please consult the subsection “The API on Target” on page 3585 for information about how to set the different options.

Display of Traces in the MSC Editor

By using the SDL Target Tester, it is possible to create Message Sequence Charts with the MSC Editor. Two modes are of special interest:

- Generating an MSC trace during an SDL session.
- Conversion of a binary file into the MSC format.

The first case, however, only makes sense if the host has enough time to handle and display all the traces created during the run-time of the SDL system. Alternatively there might be SDL systems which do not have these timing constraints. For example, the SDL system may wait until the MSC trace is completed.

The second possibility does not need that much processor power on the host site. In practice, this method is more useful.

The method with using the Cmicro Recorder is the one with the best real-time properties. In comparison to the trace, the Cmicro Recorder stores only a reduced subset of information in a compact format.

Reproduction of Errors – The Cmicro Recorder

Note:

The SDL Target Tester’s Record and Play functions are only available if a Cmicro Recorder license is available.

General

Using the Cmicro Recorder, the user can “record” the actions taken within an SDL system running on the target and save these to a file on the host machine. This file can later be “played” in order to run the system through the same state changes, so that the same actions are performed. In this way error situations, perhaps recorded by a client or a developer at another location, can be reproduced. In the following, a general description is given. A more detailed description of the internal procedures can be found in the subsection ““Type and Amount of Stored Information” on page 3639”. The subsection ““General Restrictions on Record and Play” on page 3643”may also provide more information about how it works.

Recording an SDL Session

Use of the Cmicro Recorder makes sense whenever an SDL system is executed in real-time on the target in combination with a host.

The amount of information transferred via the communications link and stored into a file on the hard disk is kept small and compact, so that the real-time properties are not influenced. Only information is stored, which is necessary in order to replay an SDL session later. It is the communication of the environment to the SDL system, and the expiration of timers, which is recorded.

Caution!

The real-time properties, and correspondingly the behavior of the SDL system may be affected when recording. The user has to ensure that there is enough time to process the functions of the Cmicro Recorder, and to transfer all the information via the communications link. In general, it is a good idea to use a high performance communications link with a large transmit buffer and a host machine, which is able to handle all recorded events at any time during the SDL execution.

It is not possible to receive information about the inner events of the recorded SDL system, by using the recorder only. The purpose of the Cmicro Recorder has nothing to do with the purpose of the Cmicro Tracer. The user also has to use the Cmicro Tracer if he wants to get information about internal events. Additional information is then transmitted via the communications link and stored into a file which can be displayed dynamically or later on. The performance of the communications link dictates whether real-time trace is possible. The more information traced via the link, the higher the performance should be.

The record mode must be switched on within the target (by using the C function interface on target), and on the host - see subsection "Record a Session" on page 3532).

Re-Playing an SDL Session

After producing a file with the record mode, the file can be replayed in order to run the SDL system in the same order through the same system states.

When the end of file is reached which is indicated with <Play mode off> to the target, the SDL system is able to react on ordinary environment signals coming from the real environment (if programmed by the user).

The play mode must be switched on within the target by using the SDL Target Tester Command Recorder-Play. No environment signals may be handled by the user in the C function `xInEnv` during the play.

Reaching the Erroneous System State

Sometimes, when a file is replayed, the erroneous situation can directly be viewed at another device, i.e. if the SDL system runs in an emulator with breakpoints set, or if there is, for instance, an LC display connected to the SDL environment.

In this case, it is not necessary to compare output files. In other cases, it might be of interest to do so as described later.

Comparing the Results

A comparison of two executions is possible, when two files containing the same type and amount of information are compared.

The procedure goes as follows:

- Change the format from binary into ASCII for both files (command "Convert-File" on page 3541).
- Use UNIX diff or an ASCII editor with options in order to evaluate the differences.

Commands for the Host and Debugging Facilities

The command interface and the debugging facilities are very helpful in finding errors produced within the target.

For instance, after an SDL interpretation error has occurred, it is possible to suspend the SDL system and to inspect the global state by viewing the signal queue(s). After the queue size has been re-dimensioned, the queue can be inspected again on the host site during a target execution.

There are several commands to check that timers are correctly processed.

The ability to set breakpoints during real-time execution and to execute a single step also helps to find an error situation.

The amount of trace information can dynamically be defined via commands. A differentiation is made between the information on the communications link and the information displayed.

Using the SDL Target Tester's Host

Introduction

The SDL Target Tester on host site is a tool chain consisting of the parts: GUI (sdtmtui), the logical part (sdtmt), the communications link gateway (sdtgate or sockgate) and the glue between these components the Cmicro Postmaster (sdtmpm). Sometimes, e.g. when the SDL Target Tester uses the MSC Editor, a further tool is executed called Cmicro Link (sdtmlnk).

The SDL Target Tester's host site is the tool, which allows communication with the target via a communications interface (see "The Communications Link's Host Site" on page 3624) to read or write to files (sdtmt), display information on the screen (sdtmtui) and/or the MSC Editor. It allows tracing of SDL and system events and has the ability to record and replay events to reproduce error situations.

Host features:

- Trace of SDL and system events
- Symbols and processes can be selected and filtered in order to reduce the amount of information
- Record and replay of SDL executions (error reproduction)
- Execution trace into file, to screen and to MSC Editor
- Files can be read in later and displayed either on screen or on the MSC Editor
- Small SDL Debugger

It is important to know that the host works independently from the target. The communication between both is built up automatically by the host if you start to communicate with the target. Thus, the host part of the SDL Target Tester remains the same executable when changing the target hardware! Only some adaptations in the configuration file `sdtmt.opt` have to be made, see "Preparing the Host to Target Communication" on page 3520.

The graphical user interface is described in "Graphical User Interface" on page 3562.

Different Ways of Using the SDL Target Tester

The standard method for using the SDL Target Tester is to connect a host with a target machine. A communications link implemented by the user must exist between host and target. The target can be seen as a remote system, where the SDL Target Tester running on the host machine can be considered a controlling unit. This is described in Figure 590.

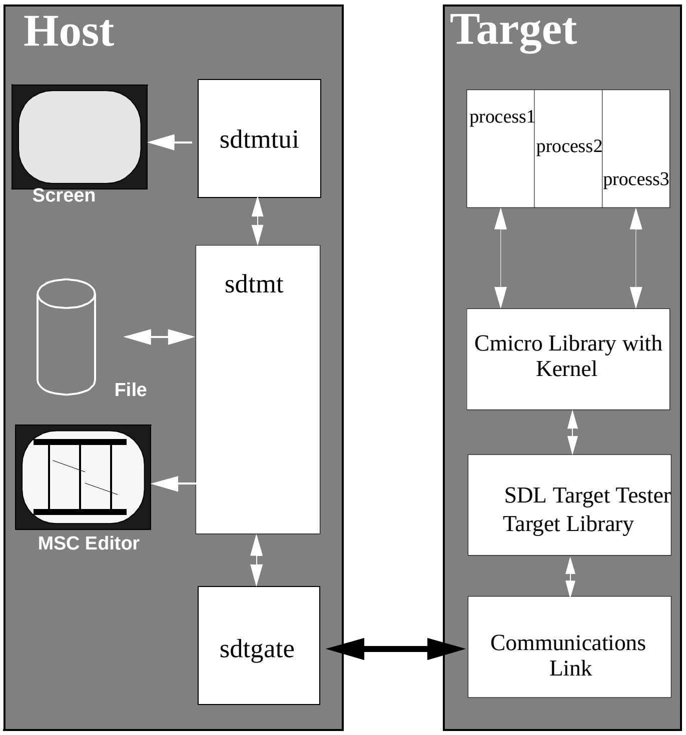


Figure 590: SDL Target Tester used for target control

The second way to use the SDL Target Tester is as a front-end for a host simulation. Therefore, a host simulation for the SDL System is built with the Targeting Expert. As described in Figure 591 the Gateway used for connection to the target is replaced by the host simulation.

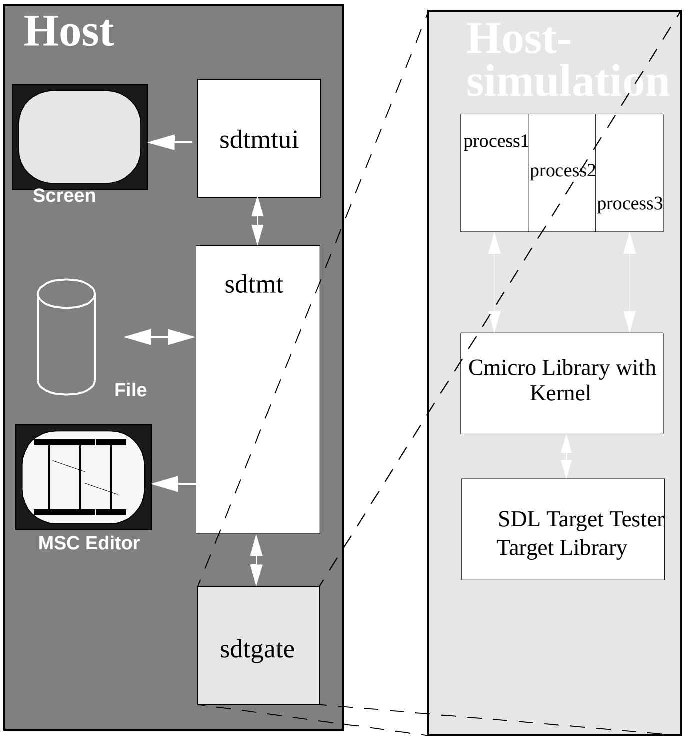


Figure 591: SDL Target Tester used for host simulation

Another way to use the host software is to use it purely as a conversion tool. This is of interest if a binary file has been written during earlier sessions with the target. The following picture shows the SDL Target Tester as a conversion tool:

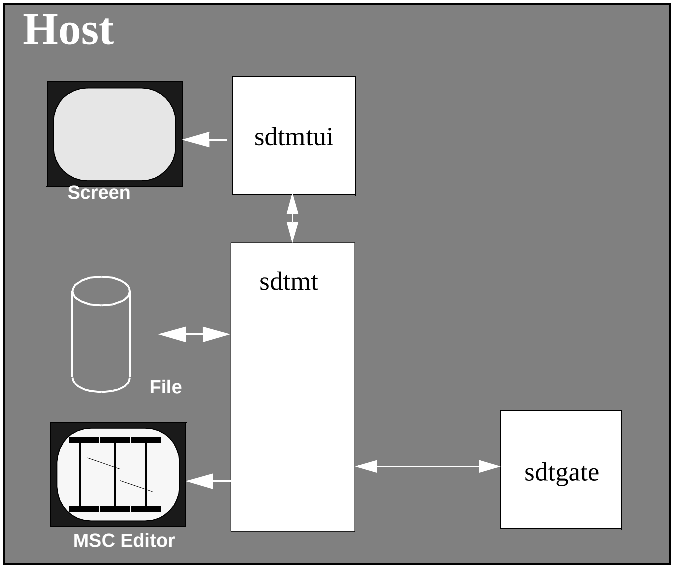


Figure 592: Sdtmt used for file conversion

Furthermore, by using the commands Output-NPAR and Output-PAR the SDL environment of the target or host simulation can be simulated with the host site.

Preparing the Host to Target Communication

After implementing the parts in the target, which are necessary to have a communication between host and target, the appropriate device on the host site is to be selected. This can be done in the file named

```
sdtmt.opt.
```

The Targeting Expert should be used to generate or modify `sdtmt.opt` (see “Configure the Host (Cmicro only)” on page 2874 in chapter 60, *The Targeting Expert*).

The Default Communications Link of Sdtmt

There is a V.24 interface implementation to transfer data between target and host. This is delivered as an additional executable called `sdtgate`.

The sdtgate default implementation uses the following protocol:

- XON/XOFF protocol possible
- 8 Bit data
- One start bit
- One stop bit
- No parity
- All data is transmitted as binary.
- Control characters are masked and de-masked.
- Binary protocol: Cmicro Protocol.

There is also a socket interface implementation to transfer data between target and host. This is delivered as an additional executable called `sockgate`.

The sockgate default implementation uses the following protocol:

- All data is transmitted as binary.
- Control characters are masked and de-masked.
- Binary protocol: Cmicro Protocol.

Communication Setup on the Host System

If sdtmt is invoked, it searches for the configuration file `sdtmt.opt`. This file is mandatory!

Each line beginning with a '#' is treated as a comment.

It contains information on target options such as:

- Target timer's scale and unit.
  These entries affect only the text trace.
- Target data types' endian, alignment and size

Caution!

The file needs to be updated if another target system is connected, or if you change from a host simulation to your real target.

The file `sdtmt.opt` should have the following entries in order to configure the communications link:

- USE_GATE:

  The path and the name of the used gate executable has to be specified here. In the delivered version of the SDL Target Tester, two executables called sdtgate and sockgate are included. They handle the V.24 and the socket interface on the host system.
  In case of a host simulation, the name of the executable which contains the simulated SDL system is inserted instead of the gateway.

- GATE_CHAR_PARAM_, GATE_INT_PARAM_

  When the gate is forked by sdtmt these values are handed over to the gateway to configure the interface.

  – V.24

    Select the device: `GATE_CHAR_PARAM_1      <devicename>`

    On UNIX `<devicename>` might be `/dev/ttya` or `/dev/ttyb` (Please ask your system administrator).

    In Windows `<devicename>` might be `COM1` or `COM2`

    Select the appropriate baud rate: `GATE_INT_PARAM_1  19200`

All other `GATE_...` entries are not used with V.24 implementation.

- Socket

  Select the device: `GATE_CHAR_PARAM_1    <IP-address>`

  `<IP-address>` has to be given in the standard format `192.168.1.4` or `localhost.`

  Select the socket port number: `GATE_INT_PARAM_1  12345`

  All other `GATE_...` entries are not used with Socket implementation.

- UNIT-NAME, UNIT-SCALE

  The target's timer scale is multiplied with the value of UNIT-SCALE and displayed in units UNIT_NAME in the SDL Target Tester. The default values are "sec" and "1".

- USE_AUTO_MCOD

  Switch on or of the use of the automatic message coder configuration on the host. Valid values are "yes" and "no".

  If the value is "no" you have to use the values described in the following to configure the message coder.

  If the value is "yes" the message coder will be configured by the target itself. Therefore it sends its configuration in a message at start-up.The following values will be ignored in that case.

Note:
The target has to be compiled with the flag `XMK_USE_AUTO_MCOD` defined

- LENGTH_

  The length (in octets) of the C basic types within the target must be given here. See the compiler manual for further information (header file `limits.h`).

- ALIGN_

  The alignment for each C basic type must be given here.

  Alignment is the distance (counted in bits) from the first bit of a character to the named basic type. The Character is stored at an address dividable by four. See Figure 593.

  Allowed values are 8, 16, 32

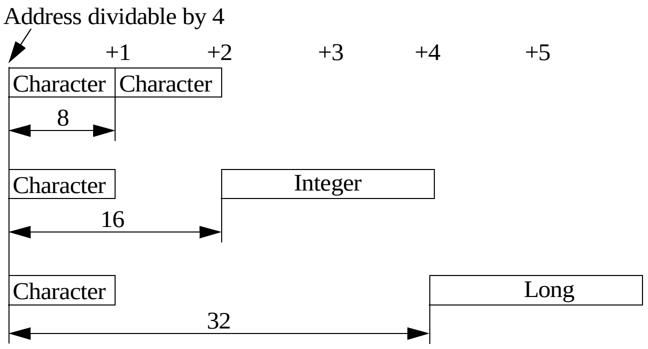


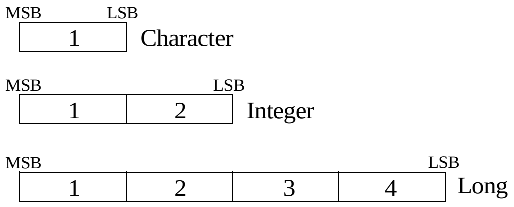
Figure 593: Target's alignment types

- ENDIAN_

  The order of the C basic types within the target's memory must be given here. See the microcontroller data sheets and Figure 594 to get further information.

  Allowed values are: 1, 12, 21, 14, 41, 23, 32, 18, 81

The octets of some C basic types are numbered as follows:



According to the type sizes above, the following Endians are possible within the target's memory:

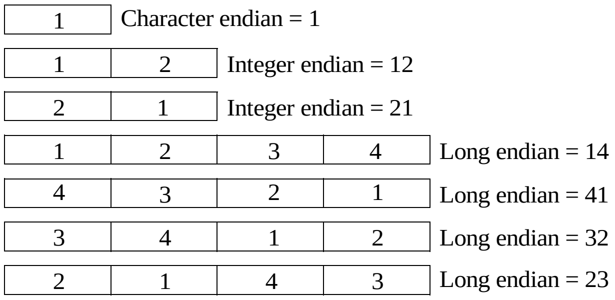


Figure 594: Definitions of the target's endian

Example 582: sdtmt.opt

```
USE_GATE      $(sdtdir)/sdtgate   (on UNIX)
USE_GATE      $(sdtdir)\sdtgate.exe  (in Windows)
GATE_CHAR_PARAM_1     COM2
GATE_CHAR_PARAM_2     0
GATE_CHAR_PARAM_3     0
GATE_INT_PARAM_1      9600
GATE_INT_PARAM_2      0
GATE_INT_PARAM_3      0

UNIT-NAME             sec
UNIT-SCALE            0.001

USE_AUTO_MCOD         no

LENGTH_CHAR           1
LENGTH_SHORT          2
LENGTH_INT            4
LENGTH_LONG           4
LENGTH_POINTER        4
LENGTH_FLOAT          4
LENGTH_DOUBLE         8
ALIGN_CHAR            8
```

```
ALIGN_SHORT         16
ALIGN_INT           32
ALIGN_LONG          32
ALIGN_POINTER       32
ALIGN_FLOAT         32
ALIGN_DOUBLE        32
ENDIAN_CHAR          1
ENDIAN_SHORT        12
ENDIAN_INT          14
ENDIAN_LONG         14
ENDIAN_POINTER      14
ENDIAN_FLOAT        14
ENDIAN_DOUBLE       18
-----------------------------------------------------------------
```

Invoking the SDL Target Tester's Host

Command Line Options of the SDL Target Tester

Option	Description
`-h`	Show a short help on the command line options.
`-v`	Show the version of the SDL Target Tester without opening it.
`-t <targetdir>`	Change the working directory to <targetdir>.

Invocation from the Organizer

There are two ways to invoke the SDL Target Tester from the Organizer with the actual project.
First you can use the menu *Tools > SDL >Target Tester UI*.
The other way is to add the following lines into the file `org-menus.ini`. For more information, see *chapter 12, The Telelogic Tau Public Interface*.

```
[MENU]
Name=&Cmicro
[MENUITEM]
ItemName=&Tester
Separator=0
StatusbarText=Start Target Tester
ProprietaryKey=1
AttributeKey=0
Scope=ALWAYS
ConfirmText=
ActionInterpretation=OS_COMMAND
BlockCommand=0
FormattedCommand=sdtmtui -t%v
```

Note:

Since code generation and compilation for Cmicro is only possible with the Targeting Expert, and the Organizer does not know about the directory where the Targeting Expert generates all files, you will be prompted automatically to enter the right directory.

In host simulation mode, internally, the -s flag is used additionally. The output of the Tester then contains some more helpful information that cannot be given in target debug mode.

Invocation from the Command Line or Desktop

Before using the SDL Target Tester described in the sections below, it is assumed, that a communications link between host and target exists (see "Connection of Host and Target" on page 3594), that the target executable has included the SDL Target Tester and that the trace is switched on within the target.

Several files have to be read from the user's current working directory.

On UNIX, the SDL Target Tester's host simply needs to be started from the working (project) directory.

1. Change the current directory to the project directory (assuming that this directory is `~/cmicro_project`).
   `cd ~/cmicro_project`
2. Now, type
   `sdtmtui`

Note:

If the command `sdtmtui` is not found, the `$path` variable needs to be set up correctly. The user should ask his system administrator or the person that is responsible for the Telelogic Tau environment.

In Windows it is recommended that you create a new shortcut to the SDL Target Tester in the Window's Start menu or on the desktop. The shortcut must contain the following entries in its properties sheet:

- Target:
  `<tauinstdir>\bin\wini386\sdtmtui.exe -t<targetdirectory> -`

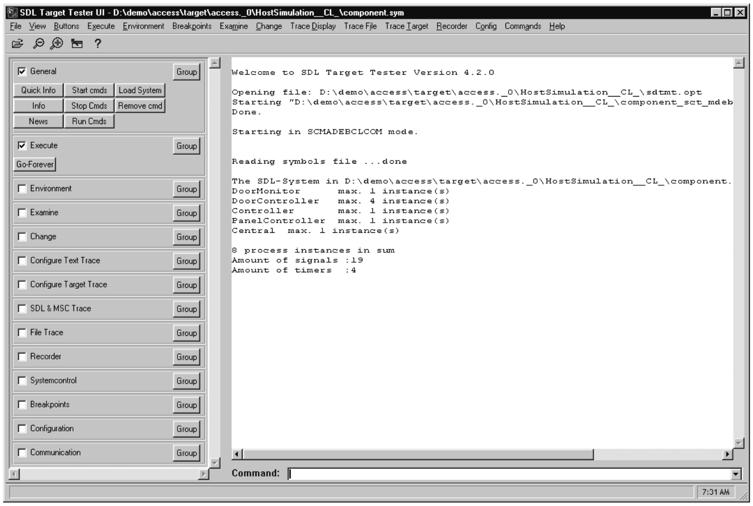
- Entry Start in:
  `<tauinstdir>\bin\wini386\`

Note:

The user should ask his system administrator about adding new links to the Start menu or the desktop.

The SDL Target Tester's host site is started when you double-click the SDL Target Tester icon.

The SDL Target Tester's host tool chain responds by showing the SDL Target Tester window:



```
Welcome to SDL Target Tester Version 4.2.0

Opening file: D:\demo\access\target\access._0\HostSimulation__CL_\sdtmt.opt
Starting "D:\demo\access\target\access._0\HostSimulation__CL_\component_sct_mdeb
Done.

Starting in SCMADEBCLCOM mode.


Reading symbols file ...done

The SDL-System in D:\demo\access\target\access._0\HostSimulation__CL_\component.
DoorMonitor       max. 1 instance(s)
DoorController    max. 4 instance(s)
Controller        max. 1 instance(s)
PanelController   max. 1 instance(s)
Central  max. 1 instance(s)

8 process instances in sum
Amount of signals :19
Amount of timers  :4
```

Figure 595: SDL Target Tester window

In the text area of the UI the following is displayed at startup:

```
Welcome to SDL Target Tester Version 4.5.0

Opening file: D:\...\sdtmt.opt
Starting "D:\...\component_sct_mdebcom.exe" ...
Done.

Starting in SCMADEBCOM mode.
```

```
Reading symbols file ...done

The SDL-System in D:\...\component.sym contains these processes:
DoorMonitor max. 1 instance(s)
DoorController max. 4 instance(s)
Controller max. 1 instance(s)
PanelController max. 1 instance(s)
Central  max. 1 instance(s)

8 process instances in sum
Amount of signals :19
Amount of timers  :4
```

After the target is reset its whole configuration is shown:

Example 583: Target's Start-up / Configuration Message

```
The current target configuration is:

Compiled Cmicro KERNEL Features:
  Preemption is                         not active
  Signal Priorities are                 used
  Receiver PId in signals are           used
  Sender PId in signals are             used
  Sending Signals the simple way is     not used
  Pid addressing is                     possible
  SDL System Stop can                   not be used
  Additional errorchecks are switched   ON
  Timer handling is                     implemented
  Timers with parameters are            not used
  Dynamic create is                     implemented
  Save is                               not implemented
  Dynamic stop is                       not implemented
  Only single instance processes        NO

Compiled SDL Target Tester Features:
  Host commands can                     be used
  Profiling can                         be used
  Debugging can                         be used
  Trace is                              implemented
  Signalfilter can                      be set
  Time stamps are                       implemented
  Recorder can                          not be used
  Realtime play can                     not be used
  Trace is currently switched           ON
  Auto MessageCoder is                  ON

Selected TARGET Modes:
  Recorder is currently switched        OFF
  Player is currently switched          OFF

Message Coder configuration: (size in octetts)
-------------------------------------------------------
  SDL process ID's (xPID)              :  4
  SDL state ID's   (xmk_T_STATE)       :  1
  SDL signal ID's  (xmk_T_SIGNAL)      :  1
  SDL time values  (xmk_T_TIME)        :  4
  Message length  (xmk_T_MESS_LENGTH ) :  1
-------------------------------------------------------

SDL Data Type sizes (octetts):
-------------------------------------------------------
  Boolean    :  1 | Bit         :  1 | Character  :  1
  Charstring :Ptr | Integer     :  4 | Natural    :  4
  Real       :  8 | PId         :  4 | Duration   :  4
  Time       :  4 | Null        :  4 | Octet      :  1
  Octetstring: 12 | ObjId       : 16
-------------------------------------------------------

  Target was generated at: Mon Jan 01 12:00:00 2001

SDL System ready for configuration.

Start with "Go-Forever".
```

Note:

Within the target's configuration message the timestamp (seconds since the 1st of January 1970) of the generated C file is sent to the host, too.

If this timestamp is not equal to the one in the symbols file (`.sym`) a message box is displayed giving hints as to what has to be done.

Getting a Target Trace

After the start-up message is received the target is ready to be traced.

The command `Go-Forever` starts the SDL system in the target and the static create trace messages will be sent.

When a process instance executes actions within a transition, trace information describing the different actions is either printed on the screen, on the MSC Editor or written into a file specified by the user. The amount of information printed can be selected using the trace commands in the monitor system. All trace information is self explanatory. This trace example was created from the AccessControl:

Example 584: Trace for the AccessControl ––––––––––––––––––––––

```
*** STATIC CREATE
*     PId      : DoorMonitor:0
*   TASK  LastOffspring :=
*** NEXTSTATE Idle

*** STATIC CREATE
*     PId      : Controller:0
*   TASK  NextDoor :=
*   OUTPUT of  Allocate to DoorMonitor:0
*     Prio     : 100
*     Now      : 0.000000 sec
*     Param(s) : 1
*   SET  MyTimer
*     Time     : 10.000000 sec
*     Value    : 0
*** NEXTSTATE WaitAllocated

*** STATIC CREATE
*     PId      : PanelController:0
*   OUTPUT of  Display to ENVIRONMENT
*     Prio     : 100
*     Now      : 0.000000 sec
*     Param(s) : 1
*** NEXTSTATE Idle
*** STATIC CREATE
*     PId      : Central:0
*   TASK  NextFree :=
*** NEXTSTATE Idle
```

```
-----------------------------------------------------
```

Debugging – A Few Guidelines

The following few guidelines may help to understand the different possibilities when searching for a specific command.

Breakpoints

A lot of error situations may occur when the system is downloaded for the first time. Possibly the basic functions needed to allow the SDL system to run the target do not work. Possibly the compiler adaptations are wrong. The initialization of the environment may be incomplete. The compiler may generate faulty object code. The memory of the target may be exceeded, i.e. either variables, constants, program code, stack or dynamic allocated memory. Also, real-time requirements may not be fulfilled.

Breakpoints can greatly help to discover such situations. The breakpoint logic is especially useful as it has only a negligible impact on the systems real-time performance.

Setting Breakpoints

Does a specific process instance (pid) receive a specific signal?

```
BPI processname nr signalname
```

Does a specific process instance (pid) go into a specific state (does it end any transition, which leads to that state)?

```
BPS processname nr statename
```

Trace Scaling

The target’s trace can be scaled. Sometimes it is necessary to omit trace altogether.

This may be necessary to reduce the amount of trace messages via the communications link, but can also be used to expand the view of some details.

Switch off the trace of signal parameters

```
Tr-Params 0
```

or switch it on again.

```
Tr-Params 1
```

The main interest may be lay on the work of one special process. This process can be explicitly defined with

```
Tr-Process processname nr
```

The trace for all other processes will be switched off.

Corresponding to this command the scope view can be set to a specific signal

```
Tr-Signal signalname
```

If the trace of the whole SDL system should be reduced, e.g. the trace of TASKs and DECISIONs is of no interest

```
Tr-Detail 4
```

switches the trace of these symbols off.

The command Tr-Detail can be used with five levels, for further information see “Tr-Detail” on page 3558.

Record a Session

Note:

The SDL Target Tester’s Record and Play functions are only available if a Cmicro Recorder license is available.

See “Invoking the SDL Target Tester’s Host” on page 3525 to start-up the SDL Target Tester. When the target’s start-up message is displayed in the text area, the target must be configured to record a session, i.e. the Recorder must be switched on in the target.

If the user wants to replay a recorded session in real-time mode he must be sure that the target executable is compiled with signals including time stamps (see “XMK_USE_SIGNAL_TIME_STAMP” on page 3415).

The user has to do several things

1. Open an output file by using the command
   `Output-File filename`
2. Switch the Cmicro Recorder on by using
   `Recorder-On`
3. Start the SDL system with
   `Go-Forever`

As with a standard trace the target's messages are displayed in the text area. The user can now use the target as if the SDL Target Tester is not involved. All traces and recorded messages are stored into the output file.

4. To end the session the Cmicro Recorder should be stopped:
   Recorder-Off
5. And the output file has to be closed:
   Close-File

Re-Play a Recorded Session

Before replaying a session it is required to record a session by following the steps listed in "Record a Session" on page 3532.

First the target has to be reset. The start-up message reappears.

Now the user can start the replay with the commands

1. `Input-File filename`
   where `filename` is the name of the file written during a recording session (this file must contain Cmicro Tracer messages as well as Cmicro Recorder information).
2. Switch the play mode on:
   `Recorder-Play`
3. If desired switch the real-time play mode on:
   `Recorder-Realtime`
   Remember: the target has to be compiled using this feature.
4. Start the target:
   `Go-Forever`

The target now starts by tracing the static creates and in accordance with the design of the SDL system the first transitions are executed and traced.

Now the SDL Target Tester's host inserts all the signals coming from the environment so that the recorded session will be replayed in the same order.

Note:

The real environment is not being polled during the replay session.

Restrictions of the SDL Target Tester

The SDL Target Tester has a few restrictions:

- Only values of the following SDL data types are displayed in readable form: Bit, Boolean, Char, Charstring, Integer, Natural, Octet, Real, PId, Duration and Time.
  Syntypes, Structures, Enumeration Types and Arrays which contain only SDL-data types can be decoded correctly.
- Complex data types, e.g. Array of Structures indexed by syntype may be decoded wrong due to alignment.
- If the SDL Target Tester encounters a unknown data type, decoding is interrupted. The first unknown data type is printed to the screen and the rest of the parameters is displayed as a hexadecimal buffer.
- Record and play mode may not be entered together.
- During the play mode only certain commands are allowed. The available commands are marked with an asterisk.
- Any SDL sort that is implemented as pointer in C (like Charstring, Object_Identifier, Ref, Own, etc.), cannot be handled in trace, record or play mode. This comes as a result of the implementation of trace (explained in chapter 67, *The Cmicro Library*). The trace does not distinguish if there is a buffer containing a pointer to be traced or not, because there is no symbol information generated into the target system for efficiency.

SDL Target Tester Commands

Syntax of SDL Target Tester Commands

Command Names

Command names are entered character by character on the keyboard. Each command consists of ASCII characters, terminated by a carriage return. A command is interpreted, after the carriage return is given. Commands may have parameters. Command names may be abbreviated by giving sufficient characters to distinguish it from other command names. The command names are interpreted from left to right. There is no distinction between upper and lower case letters.

Consider, as an example, the command ?Breaklist. The command may be entered as "?b". However, if only '?' is typed, sdtmt will respond with the error message:

```
Command was ambigous, choose one from:
< list of commands >
```

since the command cannot be distinguished from, for example, the ?Queue command. As additional information a list of all commands which would fit to the entered command and which are allowed in the current context is given.

Command Parameters

A parameter is separated from the command name by one or more spaces. The same is applicable to the separation of two parameters. There is no distinction between upper and lower case letters.

Each parameter is mandatory from a syntactical point of view and must be specified. No parameters may be left out.

If the parameter list following a command name is not complete, a dialog window is opened to enter the missing parameter(s).

Specifying more parameters than the command can handle, sdtmt prompts

```
**ERR:Extra parameters specified in command.
```

If the type of any parameter, which was entered, does not match the expected one, then sdtmt responds with an appropriate error message according to the expected type. Type

```
help commandname
```

or refer to the manual.

Abbreviation of SDL Names

Command parameters that are SDL names, may also be abbreviated, as long as the abbreviation is unique. If a name is equal to the first part of another name, as for example Wait and Wait_For_Me, then any abbreviation of Wait is ambiguous. However, if all characters of the shorter name are given (Wait in this example), this is considered as an exact match and the short name will be selected.

Errors in Commands

If an error is detected in a command name or in one of its parameters, an appropriate, self explanatory error message is displayed on the screen. The command will be ignored.

Input and Output of Data Types

<Processtype Instance>

Process type is an ASCII string identifying the SDL name of the process. Process names must be unique for all processes within the system. It is also possible to use the decimal value of the process type. The decimal value can be found by looking into the symbol file, which is also automatically generated.

The separator for process type and instance is a blank.

Instance

The value of the instance number is a decimal value beginning from 0 for the first instance of a process type. If the SDL System contains only (x, 1) declarations on SDL level, then the user has to specify “0” in any case.

Note:

There is a difference between the notation in the simulator and in the SDL Target Tester concerning instance numbering. The simulator always uses the value one as the first instance number whereas the SDL Target Tester begins numbering these at zero!

Caution!

If no symbol file was specified, then neither process type nor instance is checked within sdtmt! If the target is compiled without error checks, that might lead to a fatal error in the target system.

Symbols

Symbols are read in from a file called `<systemname>.sym`. There are a lot of error checks within the SDL Target Tester's host site which only work if a symbol file has been loaded.

Help

SDL Target Tester gives the user some quick help when entering:

```
help
```

A complete list of commands will be displayed on the screen. Commands, which are available in the current mode, are marked with an asterisk. If a command is entered which is not available in the current context the SDL Target Tester responds with:

```
Sorry. The current mode of sdtmt does not allow you
to enter that command.
The command has been ignored.
```

To get a short explanation, type:

```
help <commandname>
```

for instance type:

```
help help
```

To get a complete list of commands, type:

```
help *
```

Most of the commands marked with the type "Remote" can only be executed (in the target) if the flag XMK_ADD_MICRO_COMMAND is defined in the file `ml_mcf.h`. See "Manual Scaling" on page 3394.

??

Parameters:

```
(None)
```

Type: Local

Open a dialog with a list of all allowed commands. By pressing the OK-Button the selected command is copied to the Input line.

Active-Timer

Parameters:

```
<Processtype Instance> <Timername> <Timerparameter>
```

Type: Remote

Check, if a specific timer is set by a process, and has not been expired.

Note:

The input of a timer parameter is mandatory. The type of the parameter has to be an integer value. For a timer without parameter enter a 0 in the parameter dialog.

?All-Processes

Parameters:

```
(None)
```

Type: Remote

Note:

This command is obsolete. Use ?Process-State * * instead.

BA

Parameters:

```
(None)
```

Type: Remote

This command requests to clear all the breakpoints in the break list.

BC

Parameters:

```
<breakpoint number>
```

Type: Remote

This command requests to clear the given breakpoint number from the breakpoint list. The breakpoint number depends on the total number of defined breakpoints.

BP

Parameters:

```
<Processtype Instance> <SignalName> <StateName>
```

Type: Remote

This command sets a breakpoint on the input and the state of a process. Execution is halted only if the given input in the given state is detected.

No parameter is optional. These combinations are allowed:

```
processname nr * statename     ->Use command BPS
processname nr signalname *    ->Use command BPI
processname nr * *
processname nr signalname statename
```

BPI

Parameters:

```
<Processtype Instance> <SignalName>
```

Type: Remote

This command sets a breakpoint on the input of a given process. Execution is halted on any input of the signal in this process.

It is not mandatory to provide a parameter. These combinations are allowed:

```
processname nr *
processname nr signalname
processname *
processname signalname
```

BPS

Parameters:

`<Processtype Instance> <StateName>`

Type: Remote

This command sets a breakpoint on the state of a given process. Execution is halted if the given state is reached.

No parameter is optional. These combinations are allowed:

```
processname nr statename
processname *
processname statename
processname nr *
```

?Breaklist

Parameters:

`(None)`

Type: Remote

This command requests the current breaklist from the target and displays it on the screen as it was entered. If a symbol file has been specified during the invocation of sdtmt, then process IDs, signal IDs and state IDs are displayed with their symbolic value.

Change-Directory

Parameters:

`<directoryname>`

Type: Local

The working directory of the current SDL Target Tester session is switched to `<directoryname>`.

Close-File

Parameters:

`<filename>`

Type: Local

The current output/input file (including binary trace) with the name `<filename>` will be closed when using this command. (See “Output-File” on page 3546 and “Input-File” on page 3545)

?Coder

Parameters:

`(None)`

Type: Local

After invoking this command the current settings of the Target Message Coder TMCOD are display in the text area. The settings given here have to be previously made in the configuration file `sdtmt.opt` or have been sent by the target itself. See “Preparing the Host to Target Communication” on page 3520.

Continue

Parameters:

`(None)`

Type: Remote

This command can be used to continue processing, i.e. after a breakpoint was h. Notification is given when the command cannot be processed within the target system.

Convert-File

Parameters:

`<input filename> <output filename>`

Type: Local

This command converts the trace messages of the target (stored in the `<input filename>`) into readable ASCII trace. This ASCII trace is stored into the `<output filename>`.

Create

Parameters:

`<Parent-Processtype Instance> <Child-Processtype>`

Type: Remote

This command creates an SDL process of the given type `<Child-Processtype>`. The creating process must be specified in `<Parent-Processtype Instance>`. If there is a free process instance, which is currently dormant, it will be created and the start transition will be executed. This can be seen in the trace.

If the system is under a break condition, the command will be ignored as it is not possible to send signals (dynamic creation uses signal mechanism).

Note:

Please specify the `<Parent-Processtype Instance>` according to the SDL system. There is no check if the parent is specified correctly from the semantic point of view.

Disable-Timer

Parameters:

`(None)`

Type: Remote

This command disables the processing of timers.

The command affects the SDL timers only! However, timers which are already expired and are in the queue will be input as normal. Keep in mind that the disabling of timers changes the system behavior.

Display-Off

Parameters:

`(None)`

Type: Local

The ASCII trace into the text area is switched off.

Display-On

Parameters:

`(None)`

Type: Local

The ASCII trace into the text area is switched on.

Enable-Timer

Parameters:

`(None)`

Type: Remote

This command enables the processing of timers again, if it was previously disabled with the command Disable-Timer. Due to the fact that time (and so the system time) cannot be stopped, it might happen that all the timers set earlier may expire at the same time this command is entered. This may cause illegal system behavior, i.e. a queue overflow and the call of the ErrorHandler or another illegal behavior.

?Errors

Parameters:

    <const>

Type: Remote

Query last occurred error. The target system traces the last error within the SDL execution and stores it. The result will be displayed on the screen, both as a decimal value and as ASCII text with an explanation of the error situation.

If the value `<const>` is not equal to zero, further information about the error is given from this manual (the on-line Help Viewer is started).

Especially, when the SDL system is executed for the first time in a new hardware environment, the command may help in finding problems. It is also useful after a long execution of the SDL system, as any error situation arising may be detected. All the errors, warnings and information, which are handled by the ErrorHandler are explained in the section "User Defined Actions for System Errors – the ErrorHandler" on page 3455 in chapter 67, *The Cmicro Library*.

Exit-Single-Step

Parameters:

    (None)

Type: Local

The single step mode (which has been entered with Single-Step) is finished.

Get-Configuration

Parameters:

    (None)

Type: Remote

The target responds with its configuration. Compiled features are displayed in the text area as well as the current settings (e.g. trace on/off).

Go-Forever

Parameters:

```
(None)
```

Type: Remote

After initializing the target (for example switching the recorder mode on) the target is started with this command. It is necessary that the target is compiled with the flag XMK_WAIT_ON_HOST.

Help

Parameters:

```
[<commandname>  or  *]
```

Type: Local

A complete list of commands will be displayed on the screen. Commands available in the current mode of the SDL Target Tester are marked with an asterisk.

Example 585 –––––––––––––––––––––––––––––––––––––––

Type

```
Command>help <commandname> ,
```

to get a short explanation of one or several commands, e.g.

```
help help
```

which is the same as

```
he he
```

Example 586 –––––––––––––––––––––––––––––––––––––––

A list of all commands beginning with **b** can be requested by typing:

```
help b
```

A complete list of commands can be requested by typing:

```
help *
```

–––

Input-File

Parameters:

`<filename>`

Type: Local

The file named `<filename>` is opened to get the stored trace information (binary format). This command must be used if, for example, a recorded session should be replayed. The file `<filename>` must contain trace information in binary format.

Line

Parameters:

`(None)`

Type: Local

The command displays the line status of the communications interface. It can be used for checking if characters, frames and XONs / XOFFs have been received/transmitted and the kind of device that is actually connected.

?Memory

Parameters:

`(None)`

Type: Local

If the Cmicro Memory Management is selected (the flag XMK_USE_SDL_MEM must be defined, please view "Use of Memory" on page 3422 in chapter 67, *The Cmicro Library*) this command offers the advantage of checking currently allocated memory.

News

Parameters:

`(None)`

Type: Local

The SDL Target Tester Release's news are displayed in the text area.

Next-Step

Parameters:

`(None)`

Type: Remote

This command performs one step (one transition) if single step is active. The Single-Step command must be given prior to this.

Nextstate

Parameters:

`<Processtype Instance> <Statename>`

Type: Remote

This command sets the state of the given process instance to the value of `<Statename>`. `<Statename>` must be one of the states which are defined for the corresponding process type in SDL. Normally, that only makes sense if the system is under break condition or is idle.

Output-File

Parameters:

`<filename>`

Type: Local

All the target's trace information is stored as binary data into the file `<filename>`. This command can be used to store the information of a recording session in binary format.

Output-NPAR

Parameters:

`<Receiver-Processtype Instance> <Signalname>`

Type: Remote

An Output from the environment without any parameters is sent into the SDL system.

Output-PAR

Parameters:

`<Receiver-Processtype Instance> <Signalname> <Signal Parameter>`

Type: Remote

An output from the environment into the SDL system with a signal including parameters is sent into the SDL system. The `<Signal Parameter>` has to be given as a buffer of hexadecimal values. See Example 587.

Example 587: Output with parameters ––––––––––––––––––––––––––

```
   Output-PAR Process_A 0 Signal_1 “00-12-a3”
```

Signal ‘Signal_1’ is sent to the instance 0 of process type ‘Process_A’. ‘Signal_1’ contains parameters with the length of three octets and the values 0x00, 0x12 and oxa3. The ordering of the parameter octets must fit into the signal’s parameter structure and the target’s memory layout (alignment, endian etc.).

–––

Options-File

Parameters:

`<filename>`

Type: Local

The options file `<filename>` is read in so that the message coder can be initialized. By default the options file `sdtmt.opt` (see “Preparing the Host to Target Communication” on page 3520) is read in when the SDL Target Tester is started.

Page-File

Parameters:

`<filename> <EventsPerPage>`

Type: Local

The file `<filename>` containing trace information (in binary format) is read in and displayed as ASCII trace in the text area. The constant decimal value `<EventsPerPage>` gives the amount of trace information to be displayed until the user confirms the next page.

Print-Conf

Parameters:

`(None)`

Type: Local

This command gives all configuration information together, see the commands ?Coder and Line.

?Process-Profile

Parameters:

```
<Processtype Instance>
```

Type: Remote

The profiling of one process instance can be requested with this command. The result is the transition execution time of the longest transition and the last transition of the specified process instance. The flag XMK_ADD_PROFILE must be set when compiling the target executable to have access to this command in the target.

Please view the section "Initialization" on page 3411 in chapter 67, *The Cmicro Library*.

?Process-State

Parameters:

```
<Processtype Instance>
```

Type: Remote

Query the state of the given process(es).

The amount of queried process instances can be configured:

- If you specify a valid process instance (?Process-State P1 0), the current state of this instance is printed.
- If you enter a process type and * (?Process-State P1 *) you get the states of all instances of this type.
- If you enter ?Process-State * * all possible Process instances of the system are queried.

Depending on the traffic load on the communications interface the responses may be delayed.

Caution!

The results printed on the screen may be inconsistent. That depends on what the SDL system is doing during the query procedure. The results will be correct, however, if the whole SDL system is idle.

?Queue

Parameters:

`(None)`

Type: Remote

Some characteristics of the queue will be displayed. The command is very useful for determining the queue's dimensions. There is a peak hold, which shows the maximum number of entries in the queue since the system's start.

Example 588 ––

After typing the command, information is printed on the screen, which may look like:

```
Current Queue state:
Queue size dimensioned to:Max.20 entries
Peak hold                :1 entry/entries
Currently                :0 entry/entries in queue
Queue memory located at  :36a40010,x
```

–––

The explanation for this is: a maximum of 20 entries in the queue can be handled by the system. Since system start, there was never more than one entry being used. Currently, no signal instance has been detected in any process input port. The last value displays the physical memory allocation of the queue, which helps in debugging.

Recorder-Delay

Parameters:

`<duration>`

Type: Local

If a recorded session should be replayed it is possible to insert each environment signal with a delay of `<duration>` seconds. This feature can be used instead of the real-time play (Recorder-Realtime).

Recorder-Off

Parameters:

`(None)`

Type: Remote

Switches the Cmicro Recorder off. This command may be specified any time during a recording session. It can be used in order to reduce the amount of information sent via the communication interface.

Recorder-On

Parameters:

`(None)`

Type: Remote

Switch Cmicro Recorder to record mode. It is not a good idea to enter this command when the SDL system is not idle. (Furthermore, the question is, whether it makes sense to start a record session in the middle of an SDL execution or from the start-up of the SDL system.) It may not be relevant to start a recording session after the system has started.

> **Note:**
>
> The SDL Target Tester's Record and Play functions are only available if a Cmicro Recorder license is available.

Recorder-Play

Parameters:

`(None)`

Type: Remote

The Cmicro Recorder's play mode will be set. This command can be entered in a few states of the Cmicro Kernel only. Notification is given if necessary. The same explanation as for Recorder-on apply.

> **Note:**
>
> The SDL Target Tester's Record and Play functions are only available if a Cmicro Recorder license is available.

Recorder-Realtime

Parameters:

`(None)`

Type: Local

The play mode of the SDL Target Tester is started as real-time play. It is necessary to compile the target with signals including time stamps.

The SDL Target Tester's host sends all environment signals with the recorded time stamp into the target while replaying. Without using this command all signals are inserted with time 0 which leads to an immediate execution in the target.

Reinitialize

Parameters:

`(None)`

Type: Remote

This command tries to reinitialize the system. The reinitialization may fail, if the hardware drivers cannot be reinitialized again. Furthermore, all global auto variables in the user's C program parts cannot be reinitialized. However, if there is a clean implementation of driver initialization and de-initialization, and initialization of global variables, the command will work well.

Remove-All-Signals

Parameters:

`<Processtype Instance>`

Type: Remote

Remove all signals of the given receiver from the signal queue.

Remove-Command

Parameters:

`(None)`

Type: Local

For each command, which is sent to the target, the host waits for an explicit acknowledgment. The last command can be removed, so that the host does not wait for that acknowledgment. That can be used when the communication is hanging for any reason, for example, when the command could not be sent to the target, or was not received by the target, or the acknowledgment could not be received by the host.

Remove-Queue

Parameters:

`(None)`

Type: Remote

All signal instances in the queue will be removed. The signal queue will be empty.

Remove-Signal

Parameters:

```
<Processtype Instance> <Signalname>
```

Type: Remote

Remove a specific signal of the given receiver from the queue.

> **Note:**
>
> E.g. when a breakpoint is hit the signal which led to the current transition still remains in the signal queue. This means the first signal in the queue cannot be removed.

Reset-All-Timers

Parameters:

```
<Processtype Instance>
```

Type: Remote

This command resets all SDL timers of the given process instance.

Reset-Timer

Parameters:

```
<Processtype> <Instance> <Timername>
<Timerparameter>
```

Type: Remote

This command resets a specific SDL timer of the given process instance and with a specific timer value.

> **Note:**
>
> The input of a timer parameter is mandatory. The type of the parameter has to be an integer value. For a timer without parameter enter a 0 in the parameter dialog.

Resume

Parameters:

```
(None)
```

Type: Remote

Resume Cmicro Kernel, after a suspend has been accepted. The same explanations as for the Suspend command apply.

Run-Cmd-Log

Parameters:
`<filename>`

Type: Local

The initialization commands in the log file `<filename>` will be executed.

Caution!

In principle it is possible to use all kinds of commands in this command log file. But the correct execution cannot be guaranteed because there are to many interactions between the SDL Target Tester, the Gateway (`sdtgate.exe`), the target and the Organizer.

E.g. the MSC Editor can only be started from a command log file if the requested performance of the host machine is supported.

Caution!

It is still possible to use debugging commands in this log file, too. But the user has to think about the usefulness as all commands will be executed directly after one another without getting the correct timestamp of execution.

Scale-Timers

Parameters:
`<factor>`

Type: Remote

Scale time when setting a timer.

Often there is a need to simulate time during an SDL execution. By specifying this command, it is possible to make SDL timers run faster or slower. The command is applied on newly set timers only, i.e. if a timer is already running then it remains unaffected from the time scale. Due to this fact, the command should only be given at the start of the SDL system.

In record mode the scaled time will be stored. This results in time being scaled in play mode. It is impossible to scale time manually in play mode!

Set-Timer

Parameters:

```
<Processtype Instance> <Time value> <Timername>
<Timerparameter>
```

Type: Remote

Set an SDL Timer for the given process instance with the given time value in units. The time value is a float value re-calculated to the target's timer ticks. Please view "UNIT-NAME, UNIT-SCALE" on page 3522.

Note:

The input of a timer parameter is mandatory. The type of the parameter has to be an integer value. For a timer without parameter enter a 0 in the parameter dialog.

Example 589 Setting a timer without parameters

```
set   <Processtype> <Instance> 4711 < timername> 0

set   <Processtype> <Instance> 1.456 <timername> 0
```

Shutdown

Parameters:

```
(None)
```

Type: Remote

The command tries to shutdown the system, which means normally to exit the main program. The user can define the actions on `exit` within the target. The same explanations as for the reinitialization command apply.

Single-Step

Parameters:

```
(None)
```

Type: Remote

This command switches the single step mode on. Either Next-Step or Continue may follow.

Note:

Timers are checked normally during single step and may expire, as documented for the Disable-Timer and Enable-Timer commands. It may be better to disable the timers (with Disable-Timer) before going into single step.

Start-Cmd-Log

Parameters:

```
<filename>
```

Type: Local

All the initialization commands will be written to the log file `<filename>`. Please view the command "Run-Cmd-Log" on page 3553, too.

Start-Gateway

Parameters:

```
(None)
```

Type: Local

This command is used to start the communication in the default implementation of the sdtgate or a host simulation.

Start-MSC-Log

Parameters:

```
<level> <"Diagramname">
```

Type: Local

The MSC Editor is started and the target's trace messages will be displayed in the MSC Editor. The `<level>` specifies the amount of MSC symbols:

0 - Basic Message Sequence Chart
1 - like level 0 but with the trace of states
2 - like level 1 plus the trace of actions
4 - Basic Message Sequence Chart to file
5 - like level 0 but with the trace of states to file
6 - like level 1 plus the trace of actions to file

Start-SDLE-Trace

Parameters:

```
<Processtype>  or
*
```

Type: Remote

The trace on SDL symbol level is started for the specified process. The Organizer has to be running and the target needs to be compiled with the flag XMK_ADD_SDLE_TRACE (see “Trace” on page 3413 in chapter 67, *The Cmicro Library*).

Start-Trace-Log

Parameters:

```
<filename>
```

Type: Local

A log file containing the whole ASCII trace of the SDL Target Tester will be written.

Stop

Parameters:

```
<Processtype Instance>
```

Type: Remote

This command stops an SDL Process according to the SDL semantics, no matter if the system is suspended or under break condition. The command is, after it is received, directly executed within the target. All signals which are in the queue for this process are removed, including create signals.

Stop-Cmd-Log

Parameters:

```
<None>
```

Type: Local

The write of initialization commands will be stopped. Please view the command “Run-Cmd-Log” on page 3553, too.

Stop-Gateway

Parameters:

```
(None)
```

Type: Local

This command is used to stop the XON communication in the default implementation of the sdtgate.

Stop-MSC-Log

Parameters:

`(None)`

Type: Local

The trace on the MSC Editor is stopped.

Stop-SDLE-Trace

Parameters:

`<None>`

Type: Local

The trace on SDL symbol level is stopped.

Stop-Trace-Log

Parameters:

`<None>`

Type: Local

The trace to the trace log file is stopped. The file is closed.

Suspend

Parameters:

`(None)`

Type: Remote

Disables the Cmicro Kernel from scheduling. The transition currently running will not be affected and will be ended first, before the command is accepted. The suspended SDL system cannot process any signals, neither internal signals nor signals coming from the environment. This may lead to a queue overflow. It is recommended to disable the timers to prevent this.

System

Parameters:

`<systemname>.sym`

Type: Local

The automatically generated symbol file `<systemname>.sym` is loaded. The symbolic names for process types and Signal IDs can only be used if this file is read in.

Additional error checks can be performed within the SDL Target Tester's host if this file is read in.

?Timer-Table

Parameters:

`(None)`

Type: Remote

Some characteristics of the current timer tables are displayed. The command is useful in order to inspect the state of the SDL system, or to see if it is hanging.

Example 590 ––

After typing the command, information is printed on screen which may look like:

```
Current Timer states:
SDL NOW                        :9147,d
Timer dimensioned to           :6
Currently                      :0 timer(s) active
Timer memory location at       :36bf0014,
```

––

Where SDL NOW represents the amount of units since system start. The value of 6 shows the maximum number of timer instances in the system. The third line is self explanatory. The last line gives the physical memory allocation of the timer linked lists which may ease debugging.

Tr-Detail

Parameters:

`<level>`

Type: Local

Define Trace detail. The command is applied locally within the SDL Target Tester's host and it works on ASCII traces to the text area only.

Output to either a file or to the MSC Editor will not be affected by this command. Thus if the target is configured correctly no inconsistency can occur and no information stored in a file or in the MSC Editor will be incomplete.

Five levels are defined (0–5). The higher the level, the more SDL events are traced. Different events are assigned the levels depicted in the following table:

Trace subject	**L.1**	**L.2**	**L.3**	**L.4**	**L.5**
PRINT STRING	*	*	*	*	*
TASK	-	-	-	-	*
DECISION	-	-	-	-	*
PROCEDURE	-	-	-	-	*
SET	-	-	-	*	*
RESET	-	-	-	*	*
ACTUAL RESET	-	-	-	-	-
STOP	-	-	-	*	*
STATE	-	*	*	*	*
STATIC CREATE	-	-	-	*	*
DYNAMIC CREATE	-	-	-	*	*
CREATE	-	-	-	*	*
SAVE	*	*	*	*	*
TIMER	*	*	*	*	*
INPUT	*	*	*	*	*
OUTPUT	-	-	*	*	*
DISCARD	*	*	*	*	*
IMPLICIT CONSUMPTION	*	*	*	*	*

Tr-Params

Parameters:

`<const>`

Type: Local

The trace of signal parameters can be switched on or off. This command has an influence on the display in the text area only.

`<const>` can take the values:

0 - to switch trace off or
1 - to switch the trace of signal parameters on

Tr-Process

Parameters:

```
<Processtype Instance> <Bitmask>  or
ENV <Bitmask>  or
* <Bitmask>
```

Type: Remote

This command defines which processes in the system are traced. It is applied in the target (remote command), so that the traffic load on the communications interface can be reduced interactively.

This trace command works for all instances of the given process type. No differentiation is possible between different instances of one type.

Tr-Signal

Parameters:

```
<signalname> <flag>
```

Type: Remote

This command defines the trace for one or all signals in the system. It is applied in the target (remote command), so that the communications interface's traffic load can be reduced interactively.

The signal name is the one from the SDL system. No qualifiers are possible, which means, that signal names have to be unique for the whole SDL system. The flag may be 0 (trace off) or any other value (trace on). If an asterisk is specified for signal name, then flag is applied to all signal types. The command works for SDL output only.

Tr-Off

Parameters:

```
(None)
```

Type: Remote

The command switches the SDL Target Tester's tracer off within the target. The current option settings stored in the target will not be affected. After this command is received within the target, the trace is immediately suspended and can only be resumed with the command Tr-On. The output trace that has been written is interrupted.

Note:

The user has to consider misinterpretation of the trace because of missing traces after resuming the trace via Tr-On.

Tr-On

Parameters:

```
(None)
```

Type: Remote

This command switches the Cmicro Tracer on, if it is compiled within the target. None of the current option settings concerning trace within the target will be affected.

Unit-Name

Parameters:

```
<unitname>
```

Type: Local

The timer unit’s name can be assigned with this command, e.g. if the target’s system time is ‘milliseconds’ and the Unit-Scale is dimensioned to 1000, the <unit> should be named as ‘sec’.

Unit-Scale

Parameters:

```
(realvalue)
```

Type: Local

The target’s system time, which is received with the trace data, is multiplied by this constant factor to get a “readable” system time. See Unit-Name too.

Graphical User Interface

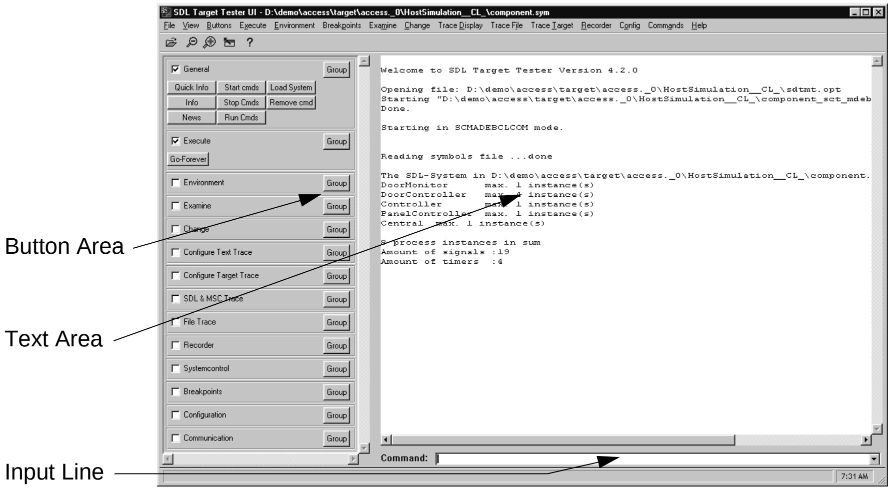
This section describes the appearance and functionality of the graphical user interface of the SDL Target Tester (sdtmtui). Some user interface descriptions common to all tools in Telelogic Tau can be found in chapter 1, *User Interface and Basic Operations*. These general descriptions are not repeated in this chapter.

Starting the SDL Target Tester UI

The SDL Target Tester UI is automatically started from the Cmicro Postmaster. When the UI is started the definition file for the button groups (see "The Button Area" on page 3564) is read in. The default name of this file is `sdtmt.btn`.

The Main Window

The main window provides a text area (which displays output from the monitor system), an input line (used for entering and displaying textual command line input to the monitor system) and a button area (with buttons for execution of monitor commands).



```
Welcome to SDL Target Tester Version 4.2.0

Opening file: D:\demo\access\target\access._0\HostSimulation__CL_\sdtmt.opt
Starting "D:\demo\access\target\access._0\HostSimulation__CL_\component_sct_mdeb
Done.

Starting in SCMADEBCLCOM mode.


Reading symbols file ...done

The SDL-System in D:\demo\access\target\access._0\HostSimulation__CL_\component.
DoorMonitor      max. 1 instance(s)
DoorController   max. 4 instance(s)
Controller       max. 1 instance(s)
PanelController  max. 1 instance(s)
Central  max. 1 instance(s)

8 process instances in sum
Amount of signals :19
Amount of timers  :4
```

Figure 596: The Main window

The Text Area

The *text area* displays all text output from the monitor system, including user prompts, error messages and command results.

Commands cannot be entered in this area, but a command given on the input line or through the use of the command buttons is echoed after the displayed prompt:

```
Command>
```

The Input Line

The *input line* is used for entering and editing monitor commands from the keyboard. For information on available monitor commands, see "Syntax of SDL Target Tester Commands" on page 3535.

The last commands entered on the input line are saved in a history list. This list can be traversed by using the `<Up>` and `<Down>` keys on the input line.

When `<Return>` is pressed anywhere on the input line, the complete string is saved in the history list and is moved to the text area. The command is then executed.

Parameter Dialogs

If a command entered on the input line requires additional user input (i.e. parameter values), the information will automatically be prompted for in a dialog:

- Parameter values of enumeration type are presented in lists, from which the value can be selected (see Figure 597).

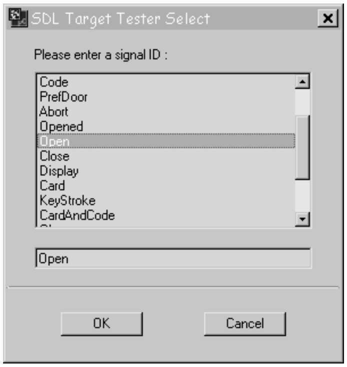


Figure 597: A typical selection dialog

The value can also be entered or edited on the text input line below the list in the dialog. For those commands that take an optional variable component, the component must be entered on the text input line after the selected variable name. It will not be asked for in a dialog.

- Other parameter values are asked for in simple text input dialogs.

Each parameter dialog has an *OK* button for confirming the value and a *Cancel* button for cancelling the command.

The Button Area

The *button area* is used for entering monitor commands by clicking the left mouse button on a command button. Each button corresponds to a specific monitor command. The buttons are divided into groups, roughly corresponding to the different types of commands listed in "SDL Target Tester Commands" on page 3535. Each group is shown as a *module* in the button area. Any number of button modules may reside in the button area. If the modules do not fit in the button area, a vertical scroll bar is added.

The definition of the buttons and button groups are stored in a *button definition file* (see "Button and Menu Definition File" on page 3579). New buttons can be added and existing ones deleted or redefined by using the *Group* menu in a button module.

If a button's definition contains parameters, the parameter values are prompted for in dialog boxes before the command is executed, in the same way as described for commands entered from the input line. See "Parameter Dialogs" on page 3563.

When no more parameter values are requested, the string shown on the input line is saved in the history list and moved to the text area. The command is then executed.

A Button Module

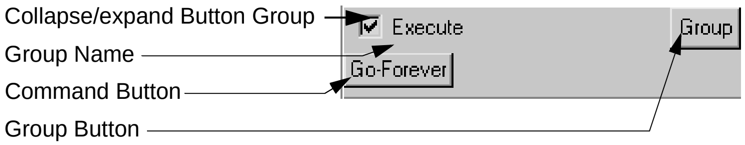


Figure 598: A button module

Each *button module* consists of a title bar and a number of command buttons arranged in rows and columns. The title bar displays:

- A *collapse/expand* toggle button. Click this to collapse (so that only the title bar is visible) and expand the module.
- The *group name* of the button module.
- A *Group* button, providing a menu with commands affecting the buttons in the module.

The *Group* button contains the following menu choices:

Add

This menu choice opens a dialog where you may add one or several new buttons to the button module. You should specify a label of the new button and a command that is to be executed when the button is clicked. Click *OK* or *Apply*, to add the button to the end of the module. For the syntax of a button definition, see "Button and Menu Definitions" on page 3580.

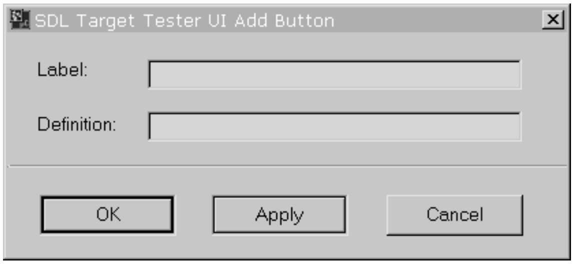


Figure 599: Adding a new button

Note:

Several buttons are allowed to have the same name within the same module. When you edit the button definition or delete the button, it is always the first occurrence that will be deleted or modified.

Edit

This menu choice opens a dialog where you can select a button to edit, that is, its label and definition. You edit the button in a similar way as described in “Add” on page 3565.

Note:

If several buttons have **the same button label**, it is always **the first button** found that will be edited, regardless of the selection.

Delete

This menu choice opens a dialog where you can select one or several buttons to be deleted.

Note:

If several buttons have **the same button label**, it is always **the first button** found that will be deleted, regardless of the selection.

Rename Group

This menu choice opens a dialog where you can edit the name of the current button module.

Delete Group

Select this menu choice deletes to delete the current module from the button area. You will be asked to confirm the operation.

The Default Button Modules

The following tables list the default buttons in the button modules and the corresponding monitor command. See “SDL Target Tester Commands” on page 3535 for more information.

> **Note:**
>
> The buttons in the button modules are specified in the button definition file. If the default button file is not used, the button modules may differ from those described here. See “Button and Menu Definitions” on page 3580 for more information.

The *Environment* Module

Button	SDL Target Tester command
Send without params	Output-NPAR
Send with params	Output-PAR

The *Execute* Module

Button	SDL Target Tester command
Go-Forever	Go-Forever

The *Examine* Module

Button	SDL Target Tester command
Queue	?Queue
Process	?Process-State
Last Error	?Errors
Timers	?Timer-Table
Process List	?All-Processes

Button	SDL Target Tester command
Active	Active-Timer
Profile	?Process-Profile

The *Change* Module

Button	SDL Target Tester command
Set Timer	Set-Timer
Reset Timer	Reset-Timer
Create	Create
Stop	Stop
Nextstate	Nextstate
Remove Q	Remove-Queue
Remove S	Remove-Signal
Remove all S	Remove-All-Signals
Scale Timers	Scale-Timers

The *Configure Text Trace* Module

Button	SDL Target Tester command
ON	Display-On
OFF	Display-Off
Detail Level	Tr-Detail
Show Params	Tr-Params 1
Hide Params	Tr-Params 0

The *Configure Target Trace* Module

Button	SDL Target Tester command
ON	Tr-On

Button	**SDL Target Tester command**
OFF	Tr-Off
Signal	Tr-Signal
Process	Tr-Process
NO Symbols	Tr-Process * B'00000000000000000000000000000000'
ALL Symbols	Tr-Process * B'11111111111111111111111111111111'

The *SDL & MSC Trace* Module

Button	**SDL Target Tester command**
Start MSC	Start-MSC-Log
Stop MSC	Stop-MSC-Log
Start SDL	Start-SDLE-Trace
Stop SDL	Stop-SDLE-Trace

The *File Trace* Module

Button	**SDL Target Tester command**
Binary Infile	Input-File
Binary Outfile	Output-File
Close binaries	Close-File
ASCII Outfile	Start-Trace-Log
Close ASCII	Stop-Trace-Log
Page File	Page-File
Convert File	Convert-File

The *Recorder* Module

Note:

The SDL Target Tester's Record and Play functions are only available if a Cmicro Recorder license is available.

Button	**SDL Target Tester command**
Recorder off	Recorder-Off
Record	Recorder-On
Play	Recorder-Play
Delay	Recorder-Delay
Real-time	Recorder-Realtime
Input recfile	Input-File recfile.trc
Output recfile	Output-File recfile.trc
Close recfile	Close-File recfile.trc

The *Systemcontrol* Module

Button	**SDL Target Tester command**
Single-Step	Single-Step
Next-Step	Next-Step
Exit-Single-Step	Exit-Single-Step
Reinit	Reinitialize
Shutdown	Shutdown
Resume	Resume
Disable Timer	Disable-Timer
Enable Timer	Enable-Timer
Suspend	Suspend

The *Breakpoint* Module

Button	SDL Target Tester command
Break input	BPI
Break nextstate	BPS
Break all	BP
View Breakpoints	?Breaklist
Clear all	BA
Continue	Continue

The *Configuration* Module

Button	SDL Target Tester command
Unit scale	Unit-Scale 1.0
Unit name	Unit-Name “milliseconds”
Target	Get-Configuration

The *Communication* Module

Button	SDL Target Tester command
Start Gateway	Start-Gateway
Coder Config	?Coder
Interface State	Line

The Menu Bar

This section describes the menu bar of the SDL Target Tester main window and all the available menu choices in the default configuration.

The menu bar contains the following fixed menus:

- *File Menu*
- *View Menu*
- *Buttons Menu*

- *Help Menu*
  (See “Help Menu” on page 15 in chapter 1, *User Interface and Basic Operations*.)

The menu bar can also contain additional menus listed in a menu definition file (see “Button and Menu Definition File” on page 3579). The following menus are defined in the delivered version of this file:

- *Execute Menu*
- *Environment Menu*
- *Breakpoint Menu*
- *Examine Menu*
- *Change Menu*
- *Trace Display Menu*
- *Trace File Menu*
- *Trace Target Menu*
- *Recorder Menu*
- *Config Menu*
- *Commands Menu*

File Menu

The *File* menu contains the following menu choices

- *Open*
  (See “Open” on page 9 in chapter 1, *User Interface and Basic Operations*.)

- *Set Directory*

- *Exit*
  (See “Exit” on page 15 in chapter 1, *User Interface and Basic Operations*.)

Set Directory

This menu choice opens a dialog where you can enter the path to your current project directory.

View Menu

The *View* menu contains the following menu choices:

- *Find*
- *Find Next* *(F3)*

- *Find Selection*
- *Copy from Text Area*
- *Paste into Command Line*
- *Line Numbering*
- *Clear Text Area*

Find

This menu choice opens a dialog where you can search for text.

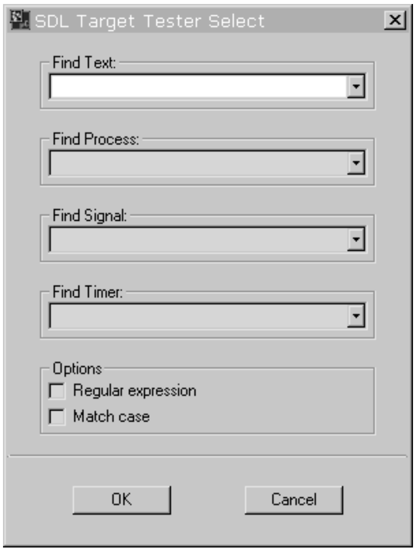


Figure 600: The Find dialog

It is possible to use regular expressions for finding text or to select to find a process, signal or timer belonging to the SDL system.

Find Next

This menu choice will search again for text already found.

Find Selection

This menu choice will search for text currently selected in the SDL Target Tester's text area.

Copy from Text Area

This menu choice will copy the text selected in the SDL Target Tester's text area.

Paste into Command Line

The menu choice will paste the content of the clipboard to the command line.

Line Numbering

This menu choice toggles line numbering in the SDL Target Tester's text area. Line numbering allows a better overview.

Clear Text Area

This menu choice will clear the complete text area of the SDL Target Tester.

Buttons Menu

The *Buttons* menu contains the following menu choices:

- *Load*
- *Append*
- *Save*
- *Save As*
- *Expand Groups*
- *Collapse Groups*
- *Add Group*

For more information on the UI's button definition file, mentioned in the menu commands below, see "Button and Menu Definition File" on page 3579.

Load

This menu choice opens a dialog where you can specify a new button definition file. This will override the current button definitions. All buttons and modules currently in the button area are deleted and replaced by the buttons and modules defined in the new file.

Append

This menu choice opens a dialog where you can specify to append the contents of a new button definition file into the current button definitions. Buttons with new labels are added to the button area, while buttons with already existing labels in the same module will be duplicated (possibly with different definitions).

Save

This menu choice saves the current button and module definitions in the button definition file under its current file name.

Save As

This menu choice opens a dialog where you can save the current button and module definitions in a new button definition file.

Expand Groups

This menu choice expands all modules in the button area.

Collapse Groups

This menu choice collapses all modules in the button area.

Add Group

This menu choice opens a dialog where you can specify the of a new button module to add. The new module will be added after the last module in the button area. It is also possible to add several modules.

Execute Menu

Item	**SDL Target Tester command**
Start Gateway	Start-Gateway
Go-Forever	Go-Forever
Single-Step	Single-Step
Reinit	Reinitialize
Disable-Timer	Disable-Timer
Next-Step	Next-Step

Item	SDL Target Tester command
Shutdown	Shutdown
Enable Timer	Enable-Timer
Exit-Single-Step	Exit-Single-Step
Resume	Resume
Suspend	Suspend

Environment Menu

Item	SDL Target Tester command
Send with params	Output-PAR
Send without params	Output-NPAR

Breakpoint Menu

Item	SDL Target Tester command
Break Input	BPI
View Breakpoints	?Breaklist
Break Nextstate	BPS
Clear all	BA
Break all	BP
Continue	Continue

Examine Menu

Item	SDL Target Tester command
Queue	?Queue
Timers	?Timer-Table
Active	Active-Timer
Process	?Process-State

Item	SDL Target Tester command
Process List	?All-Processes
Profile	?Process-Profile
Last Error	?Errors

Change Menu

Item	SDL Target Tester command
Set Timer	Set-Timer
Stop	Stop
Remove Signal	Remove-Signal
Reset Timer	Reset-Timer
Nextstate	Nextstate
Remove all S	Remove-All-Signals
Create	Create
Remove Q	Remove-Queue
Scale Timers	Scale-Timers

Trace Display Menu

Item	SDL Target Tester command
Display on	Display-On
Display off	Display-Off
Trace Detail	Tr-Detail
Show Params	Tr-Params 1
Hide Params	Tr-Params 0
Start MSC	Start-MSC-Log
Stop MSC	Stop-MSC-Log

Trace File Menu

Item	SDL Target Tester command
Binary Infile	Input-File
ASCII Outfile	Start-Trace-Log trace.asc
Page File	Page-File
Binary Outfile	Output-File
Close ASCII	Stop-Trace-Log
Convert File	Convert-File
Close Binaries	Close-File

Trace Target Menu

Item	SDL Target Tester command
Trace on	Tr-On
Trace off	Tr-Off
Trace Process	Tr-Process
Trace Signal	Tr-Signal

Recorder Menu

Item	SDL Target Tester command
Recorder off	Recorder-Off
Delay	Recorder-Delay
Output recfile	Output-File recfile.trc
Record	Recorder-On
Realtime	Recorder-Realtime
Close Recfile	Close-File recfile.trc
Play	Recorder-Play
Input recfile	Input-File recfile.trc

Config Menu

Item	SDL Target Tester command
Unit scale	Unit-Scale
Unit name	Unit-Name
Target	Get-Configuration
Coder-Config	?Coder
Interface state	Line

Commands Menu

Item	SDL Target Tester command
Quick Info	Help
Info	Help *
Remove cmd	Remove-Command

Button and Menu Definition File

In the SDL Target Tester UI, the button definition information is stored on file (per default this file is named `sdtmt.btn`), i.e. definitions of button groups and button commands in the main window's button area. Furthermore it is possible to define menu entries.

At start-up of the UI, the file is determined in the following way. If the file name does not contain a directory path, the files are searched for in the following directories:

- The current directory
- The Telelogic Tau installation directory

Once the file has been found, it is read and the contents of the corresponding window is set up. If the file cannot be found, the corresponding window area becomes empty and a file selection dialog is opened to enter the required file name.

The text file can contain comment lines starting with the `'#'` character. Empty lines are discarded.

Note:

When a file is read, no checks are made upon the relevance or correctness of the definitions contained in the file.

Button and Menu Definitions

The *button and menu definitions* are stored in a definition file with the default extension `.btn`. The definitions are divided into groups where each group defines a button module in the main window's button area or a menu in the menu bar.

Syntax

In the file, a button group has the following syntax:

```
:[DISPLAY_AS] <entry name>
<item label>
<definition>
<button label>
<definition>
  . . .
```

The [DISPLAY_AS] prefixes the button group's or menu's name <entry name> and can take three different keywords:

- `COLLAPSED`
  A button group is created, initially collapsed.
- No keyword
  A button group is created, initially expanded.
- `MENU`
  A new menu is created.

The <item label> is either the label of the button in a button module or the label of an menu item.

The <definition> is the SDL Target Tester command that will be executed when the button is pressed/the menu item is selected. The syntax of a button definition is the same as when entering a command textually to the monitor.

Missing parameters at the end of the command will open dialogs for those parameters.

After all there is no limit in the amount of buttons in a button group or the amount of items in a menu entry except the visibility and usability of the buttons/items.

Comparing with the Simulator there is a simple rule of ordering the buttons shown in Figure 601. A new button row is added only if the previous row has got 3 buttons.

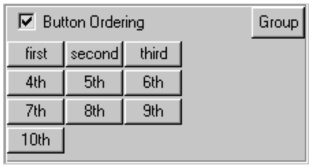


Figure 601: Ordering of the buttons in a button group

The Target Library

General

The target library is a mandatory part of the SDL Target Tester. The SDL Target Tester is of no use without the functions and definitions contained within.

In order to use the options of SDL Target Tester's host site, the target library must be configured accordingly.

A lot of defines make it possible to reduce the SDL Target Tester's functions within the target, so that memory requirements can be reduced when necessary.

In the following sections are explanations of the file structure and the application program interface (API) within the target.

File Structure

The following picture explains the file structure for the SDL Target Testers target library.

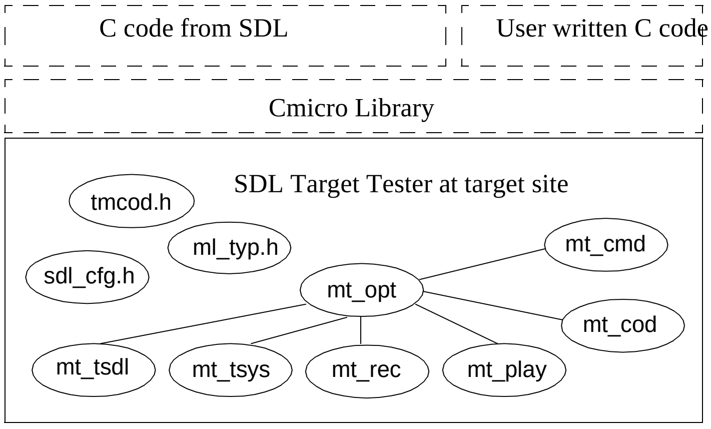


Figure 602: Files of the Cmicro Library

Description of Files

sdl_cfg.h

This file is generated automatically by the Cmicro SDL to C Compiler into the directory which is currently active. It contains compilation flags used for the automatic scaling of the Cmicro Library and the generated C code. The file must not be edited by the user.

Caution!

This file always carries the same name for each SDL system generated and is stored into the currently active directory. A conflict will occur if the user tries to generate several systems into the same directory.

Therefore be sure to use different working directories for each SDL system. Otherwise, unpredictable results during run-time of the generated code will occur, as some automatic scalable features may not re-compiled.

ml_typ.h

This file is the central header file in the Cmicro Package. It contains

- more #includes
- defines which are of global interest and Cmicro Library internal defines
- typedefs which are of global interest and Cmicro Library internal typedefs
- extern-declarations which are of global interest and Cmicro Library internal extern declarations

tmcod.h

`tmcod.h` is an automatically generated file of the Target Message Coder Generator (TMCOD). All the SDL Target Tester command tags and the corresponding parameter structures are defined here. It is strongly recommended not to modify this file.

mt_cmd.c

This file is part of the command interface on the target site. The commands received via the communications link will be performed by this file.

mt_cod.c

This is the module which exports the functions to encode / decode any message from the Cmicro Protocol.

mt_opt.c

This module exports the functions which allow the user to define options regarding the tracer, recorder and player components.

Options may be specified to reduce the traffic load of the system, This is important when a slow communications interface is involved and the user wants to test in real-time.

mt_play.c

This file is part of the Cmicro Recorder and it contains all the functions for the play mode.

mt_rec.c

This file is part of the Cmicro Recorder and it contains all the functions for the record mode.

mt_tsdl.c

This module exports functions to trace SDL actions. These functions are mainly for the use of the Cmicro Kernel and Tester.

Functions to trace either into RAM, into a file or via a V.24 interface are delivered.

The module is common for all these trace methods. The functions exported are called directly by the Cmicro Kernel if the SDL Target Tester or part of the SDL Target Tester is involved when executing the SDL system.

mt_tsys.c

This module contains the functions which handle the trace of system events such as scheduler events or the trace of system errors.

mt_*.h

These header files contain the external definitions of the corresponding `.c` files.

The API on Target

General

The C application program interface (API) can be used both by SDL processes and / or by any other C function on the target.

The simplest method, however, is to set some options before initializing the SDL system and hold these options during the whole run-time. A more comfortable method is to set some options before initializing the SDL system and then change the options in an SDL process, when a specific situation is detected. By using this method it is possible to reduce the amount of information and to make it easier to find unwanted behavior or an erroneous situation in a trace. This can be achieved, for example, by writing the necessary C function call(s) into an SDL task symbol using the #CODE directive.

The following pages introduce the C function interface, which is used in the target to set the trace options.

Caution!

It is important to take care with function parameters. Any function return value should also be checked.

Initialization

```
void xmk_InitOptions ( void )
```

Initialization of the SDL Target Tester is absolutely necessary. It must be done before any other initialization, directly before the C function call `xmk_InitSDL`. The user should not remove this function call in the template `main()` function if one wants to use the SDL Target Tester.

Nothing happens before the C function `xmk_OptionsSymbol` is called, and `xmk_TracerActive` is set to `XMK_TRUE`.

Switch the Cmicro Tracer on / off

In order to get an execution trace it is first necessary to switch it on by assigning

```
xmk_TracerActive = XMK_TRUE
```

This should normally be done after the C function `xmk_InitOptions`.

The settings of functions whose names begin with `xmk_Options*` remains unaffected when changing the above flag.

The flag is **not** set by default.

Selecting SDL Symbols That are to Be Traced

```
xmk_OPT_INT xmk_OptionsSymbol ( unsigned char process-type-id,
                                long bitmask )
```

This function must be called in order to begin the trace. By default, no symbol is selected.

Select the trace for process-type-id and then set or un-set different symbols with the bit mask. If anything goes wrong the function will return `XMK_ERROR`, else `XMK_OKAY`.

It is possible to specify bit mask values for a specific SDL process, for all SDL processes, for the environment and for the kernel part of the Cmicro Library. The last possibility is necessary to allow the optional trace for system events like error trace and trace of scheduling events.

Allowed values for process-type-id:

0	First process-type-id, XPTID_ from `sdl_cfg.h`
1	Second.....
N-1	Last... (N is the maximum number of process types)
xNULLTYPE	specify trace for all processes
any other value	The environment
MICROKERNEL	specify trace for Cmicro Kernel
ENV	Indicates the SDL environment

Allowed values for bit mask:

SDL symbols	System events
TSDL_STATE	TSYS_SCHEDULE
TSDL_INPUT	TSYS_ERROR
TSDL_SAVE	TSYS_SHOWPRIO
TSDL_OUTPUT	
TSDL_CREATE	
TSDL_STOP	
TSDL_STATIC_CREATE	
TSDL_DYNAMIC_CREATE	
TSDL_DECISION	
TSDL_TASK	
TSDL_PROCEDURE	
TSDL_RESULT	
TSDL_TIMER	
TSDL_SET	
TSDL_RESET	
TSDL_ACTUAL_RESET	
TSDL_SIGNALPARAMS	
TSDL_DISCARD	
TSDL_IMPLICIT_CONSUMPTION	

Selecting SDL Signals to Be Traced

```
xmk_OPT_INT  xmk_OptionsSignal ( xmk_T_SIGNAL signal id,
                                 unsigned char bitmask )
```

This function is optionally called because the trace of all SDL signals is switched on per default.

Select the trace for signal id. Then set or un-set different symbols with bit mask. If anything went wrong, the function will return `XMK_ERROR`, else `XMK_OKAY`.

It is possible to specify either “off” for all signals, “on” for all signals, “off” for a specific signal or “on” for a specific signal.

Allowed values for signal-id:

0	First signal-id, #define from generated C code and `<systemname>.ifc`
1	Second.....
N-1	Last.....
XMK_ALLOW_NO	Specify trace for all processes
XMK_ALLOW_ALL	Specify trace for Cmicro Kernel

Allowed values for bitmask:

- Any value in the range of `0x00` to `0xff`.
- A value != `0x00` switches the trace for the signal on.
- A value == `0x00` causes the trace for the given signal(s) to be switched off.

Example to Set Trace Options

A simple method to set the trace options is to set them directly within the C function `main`, before the Cmicro Kernel is initialized. The user will find this example as a template in the file `bare_ex.c` delivered with the Cmicro Package, below the directory `<sdtdir>/cmicro/template`.

Example 591 ––

```
#include “sdl_cfg.h”
#include “ml_typ.h”
    ... main ( ... )    {
  extern int xmk_TracerActive ;
  long       bitmask = 0      ;
  int        result  = XMK_OKAY ;

/* Initialize SDL Target Tester */
xmk_InitOptions ( );

/* Set all options ON */
result = xmk_OptionsSymbol (xNULLPID, 0xffffffff);
if (result != XMK_OKAY) ErrorInUsage ();

/* Set some options ON */
bitmask = TSDL_INPUT  ;
bitmask = TSDL_STATE  ;
bitmask = TSDL_OUTPUT ;

                 /*          +---- Proces-Type - Id (see automatically */
                 /*          |     generated file *.ifc)               */
                 /*          V                                         */
 result = xmk_OptionsSymbol (0, bitmask );
 if (result != XMK_OKAY) ErrorInUsage ();
 result = xmk_OptionsSymbol (1, 0x00000000 );
 if (result != XMK_OKAY) ErrorInUsage ();

/* S w i t c h   o n   t h e   t r a c e */
xmk_TracerActive                 = XMK_TRUE ;
/* Template to set options to filter signals */
                    /*                                           */
                    /* +---- Allow all Signals to be traced      */
                    /* V                                         */
 xmk_OptionsSignal (XMK_ALLOW_NO, 0);
                 /* +---- Signal id is to be set correctly !    */
                 /* |     according to the automatically        */
                 /* |     generated *.ifc - file                */
                 /* V                                           */
 xmk_OptionsSignal (1, 0xff);
 xmk_OptionsSignal (2, 0xff);
 xmk_OptionsSignal (3, 0xff);
/* I n i t i a l i z e   C m i c r o   K e r n e l */
xmk_InitSDL ()

/* R u n   C m i c r o   K e r n e l */
if (xmk_RunSDL (0xff) != XMK_STOP);
exit (0);
}
```

––

Getting a Trace in SDL Tasks with C Code

The Cmicro SDL to C Compiler generates usable traces for SDL tasks only if it is a task containing at least one SDL assignment. Whenever C code is specified within a task symbol (by using the #CODE directive), the Cmicro SDL to C Compiler is not able to generate good trace information.

There are two C functions which enable a work-around:

- `xmk_PrintString (const char *)` or
- `xmk_TSDL_Task (const char *)`

in order to get a trace from C code. The trace output of a call like

```
 xmk_PrintString ('Hello SDL world')
```

looks like:

```
USER:   Hello SDL world
```

The strings have to be unique, in order to distinguish a trace of one process from another. Another difference between the above function is that the C function `xmk_PrintString` does not check if a trace is wanted or not. The trace will be done unconditionally via the SDL Target Tester's communications link.

Of course, the function may also be applied in non SDL program parts.

Using Record or Play Mode of the Cmicro Recorder

```
xmk_OPT_INT xmk_SetRecorderMode ( int mode )
```

This function must be called by the user in order to:

- switch the record mode of the Cmicro Recorder on, or
- switch the play mode of the Cmicro Recorder on, or
- switch any function of the Cmicro Recorder off.

Allowed values for mode:

- `XMK_RECORDER_RECORD`
- `XMK_RECORDER_PLAY`
- `XMK_RECORDER_OFF`

Environment Functions

xInEnv ()

```
void  xInEnv ( void )
```

This function is called each time the Cmicro Kernel's main loop is entered. A more detailed description of the C function `xInEnv()` can be found in "xInEnv()" on page 3440 in chapter 67, *The Cmicro Library*.

xOutEnv ()

```
int  xOutEnv ( xmk_T_SIGNAL sig,
               xmk_T_PRIO prio,
               xmk_T_MESS_LENGTH data_len,
               void *p_data,
               xPID Receiver
```

This function is called each time the SDL system wants to output a signal to the environment. If the environment should be another system (e.g., an MS DOS PC), the signal can be sent directly by calling the C function `xmk_Cod_Encode` (which is described in the following section). A more detailed description of the C function `xOutEnv` can be found in "xOutEnv()" on page 3443 in chapter 67, *The Cmicro Library*.

xmk_Cod_Decode ()

```
int  xmk_Cod_Decode ( char *p_RAWBuffer,      In
                      int *p_MessageClass,    Out
                      int *p_MessageTag,      Out
                      int *p_MessageLength,   Out
                      char *p_dest,           In/Out
                      int dest_len )          In
```

This function is free to be used on the target site. It decodes a given data link frame in the Cmicro Protocol format into the different pieces of data. `MessageClass` may be in the range from `F3h` to `FFh`. The value `Fh` in the higher nibble is used to synchronize the data stream if synchronization fails and `0h` to `2h` in the lower nibble is used by the SDL Target Tester itself. `MessageTag` may be of any value. `p_struct` is a pointer to a C structure which is to be transferred via the communications link. `Struct_length` is the result of the sizeof operator being applied to this structure.

The function is the counterpart to the C function `xmk_Cod_Encode`.

xmk_Cod_Encode ()

```
int  xmk_Cod_Encode ( int MessageClass,
                      int MessageTag,
                      char *p_data,
                      int length )
```

This function is free to be used on the target site: It encodes the given information into a data link frame of the Cmicro Protocol format and sends it via the communications interface. `MessageClass` may be in the range from `F3h` to `FFh`, the value `Fh` in the higher nibble is used to synchronize the data stream if synchronization fails and `0h` to `2h` in the lower nibble is used by the SDL Target Tester itself. `MessageTag` may be of any value. `p_struct` is a pointer to a C structure which is to be transferred via the communications link. `Struct_length` is the result of the sizeof operator being applied to this structure.

The function is the counterpart to the C function `xmk_Cod_Decode`.

Note:

The maximum amount of characters which can be transferred is restricted to 255 per default in this Cmicro Package version.

Compiling and Linking the Target Library

Some defines are to be set in order to compile and link the parts of the SDL Target Tester correctly. These defines can be set either on the command line when invoking make or in the header file `ml_mcf.h`. Be sure that for each configuration possibility all the defines are set appropriately.

The best way to select the compiler options for the SDL Target Tester is to use the Targeting Expert. Please view the section "Configure the SDL Target Tester (Cmicro only)" on page 2873 in chapter 60, *The Targeting Expert*.

Here is a first idea of what has to be to set in the configuration file `ml_mcf.h`:

```
#define XMK_ADD_MICRO_TESTER
#define XMK_ADD_MICRO_TRACER
#define XMK_USE_V24
#define XMK_USE_COMMLINK
```

Scaling of SDL Target Tester Functionality

In order to reduce the amount of memory - e.g. if memory gets scarce within a progressing project - it is possible to reduce the SDL Target Tester's functionality on target site.

Removing the whole SDL Target Tester

It is quite simple to remove any functionality used by the SDL Target Tester by specifying:

```
#undef XMK_ADD_MICRO_TESTER
```

Removing the Cmicro Tracer

This is possible by specifying

```
#define  XMK_ADD_MICRO_TESTER
#undef   XMK_ADD_MICRO_TRACER
```

Removing the Cmicro Recorder (Inclusive Play Mode)

This is possible by specifying

```
#define  XMK_ADD_MICRO_TESTER
#undef   XMK_ADD_MICRO_RECORDER
```

Removing the Command and Debug Interface

This is possible by specifying

```
#define  XMK_ADD_MICRO_TESTER
#undef   XMK_ADD_MICRO_COMMAND
```

Partial functionality may be left out by un-defining one of the defines described under “Message Transfer and Presentation of Messages” on page 3595.

Configuration of Buffers

Users must configure buffers used by the SDL Target Tester.

The reason behind this is that the SDL Target Tester uses an efficient way to handle those buffers. This is of great importance especially for micro controllers.

Additionally, these buffers must be consistently dimensioned. For instance, the message buffer of the communications link must be able to handle the largest parameter list of a signal from SDL. The same situation arises in the trace of SDL tasks and procedure names.

Caution!

If any buffer is dimensioned too small a run-time error may result.

The following defines are used to dimension buffers.

XMK_MAX_PRINT_STRING

This define restricts the amount of characters transferred via the communications link when using the C function `xmk_PrintString`. This function is used only by the user.

XMK_MAX_LEN_SYMBOL_NAME

This define is used to restrict the amount of characters transferred via the communications link for the SDL tasks and SDL procedure names.

The transmit and receive buffers of the default communications link software must be configured in the data link (`dl`) module using the following defines:

Transmitter buffer:

```
   #define XMK_MAX_SEND_ONE_ENTRY          1000
   #define XMK_MAX_SEND_ENTRIES            20
```

Receiver buffer:

```
   #define XMK_MAX_RECEIVE_ONE_ENTRY       1000
   #define XMK_MAX_RECEIVE_ENTRIES         20
```

where `*_ONE_ENTRY` is the dimensioning of one message, and `*_ENTRIES` is the maximum amount of entries in the message FIFO of the data link module. Note, that `*_ONE_ENTRY` must be greater than the greatest of the `XMK_MAX_*` defines above, plus a reserve of 6 bytes.

Connection of Host and Target

General

The following sections will detail the technical requirements and the technical implementation of the connection between host and target. The following items will be outlined:

- “Structure of Communications Link Software” on page 3595
- “Default Implementation of Communications Link Software” on page 3595 as delivered.
- How to implement user host executables, which are able to communicate with the SDL Target Tester’s target library. This is described in the subsection “Open Interface” on page 3632.

Structure of Communications Link Software

This is the structure for the communications link software, which is - in principle - applicable for host, as well as for target site:

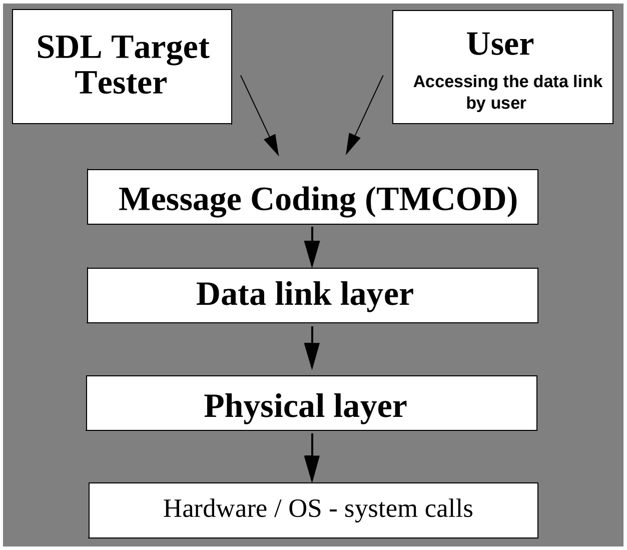


Figure 603: Structure of the communications link software

The following subsections describe the current implementation and how to implement a new communications link software.

Default Implementation of Communications Link Software

Message Transfer and Presentation of Messages

State Machine and Handshake

For all the commands and messages, there is a small state machine to handle the correct transfer of information. All messages, which have the suffix `_REQ` (request) in their name (message tag) are to be confirmed from the target with one of the `_CNF` (confirmation) messages. (See the file `tmcod.h`, where all message tags are defined).

For messages which take parameters, there is always (no exception) a special _CNF message which carries the response parameters back to the requestor. The requestor is usually the host.

For the messages, which do not carry parameters, a positive confirmation is done via the message CMD_OKAY_CNF.

If a message cannot be recognized or processed within the target, a negative confirmation CMD_ERROR_CNF is used in all cases.

Each _REQ message must be confirmed before the next one can be sent.

Messages with an _IND (indication) suffix are mostly sent from target to host in order to indicate a specific situation. For these messages, the sender is always the originator.

Message Formats

The following subsections give a complete list of messages which can be sent to and received from the target.

Each typedef shown below is packed into a data link frame in its target representation.

That means a memcpy (destbuffer,struct,sizeof(struct)) is performed to copy a structure to/from the data link frame.

On host site, it is therefore necessary to perform message encoding and message decoding.

Formats Concerning the Cmicro Tracer

All messages belonging to the Cmicro Tracer are part of the message class XMK_MICRO_TRACER (F1h).

- Message tag: CMD_TTASK

  ```
  Messageclass       :XMK_MICRO_TRACER
  Meaning            :Trace of SDL Task
  Sent when          :Execution of that SDL symbol
  Direction          :T->H
  Typedefinition     :xmk_T_CMD_TTASK
  Conditional compile:XMK_ADD_TTASK
  ```

  Message description: The message is sent when a task symbol is entered in generated C Code and when the trace is set for this symbol. Furthermore, the trace must be active.

- Message tag: CMD_TSET

```
Messageclass        :XMK_MICRO_TRACER
Meaning             :Trace of SDL Set
Sent when           :Execution of that SDL symbol
Direction           :T->H
Typedefinition      :xmk_T_CMD_TSET
Conditional compile:XMK_ADD_TSET
```

Message description: The message is sent when a timer set is performed in generated C Code and when the trace is set for this symbol. Furthermore, the trace must be active.

- Message tag: CMD_TRESET

```
Messageclass        :XMK_MICRO_TRACER
Meaning             :Trace of SDL Reset
Sent when           :Execution of that SDL symbol
Direction           :T->H
Typedefinition      :xmk_T_CMD_TRESET
Conditional compile:XMK_ADD_TRESET
```

Message description: The message is sent when a timer reset is performed in generated C Code and when the trace is set for this symbol. Furthermore, the trace must be active.

- Message tag: CMD_TSTOP

```
Messageclass        :XMK_MICRO_TRACER
Meaning             :Trace of SDL Stop
Sent when           :Execution of that SDL symbol
Direction           :T->H
Typedefinition      :xmk_T_CMD_TSTOP
Conditional compile:XMK_ADD_TSTOP
```

Message description: The message is sent when an SDL process stop is performed and when the trace is set for this symbol. Furthermore, the trace must be active.

- Message tag: CMD_TSTATE

```
Messageclass        :XMK_MICRO_TRACER
Meaning             :Trace of SDL State
Sent when           :Execution of that SDL symbol
Direction           :T->H
Typedefinition      :xmk_T_CMD_TSTATE
Conditional compile:XMK_ADD_TSTATE
```

Message description: The message is sent when an SDL nextstate is performed and when the trace is set for this symbol. Furthermore, the trace must be active.

- Message tag: CMD_TSTATIC_CREATE

```
Messageclass        :XMK_MICRO_TRACER
Meaning             :Trace of SDL Static create
Sent when           :Execution of that SDL symbol
Direction           :T->H
Typedefinition      :xmk_T_CMD_TSTATIC_CREATE
Conditional compile:XMK_ADD_TSTATICCREATE
```

Message description: The message is sent for each statically created SDL process for which the start transition is executed and for which the trace is set for this symbol. Furthermore, the trace must be active.

- Message tag: CMD_TCREATE

```
Messageclass       :XMK_MICRO_TRACER
Meaning            :Trace of SDL Create
Sent when          :Execution of that SDL symbol
Direction          :T->H
Typedefinition     :xmk_T_CMD_TCREATE
Conditional compile:XMK_ADD_TCREATE
```

Message description: The message is sent when the SDL action create is performed and when the trace is set for this symbol. Furthermore, the trace must be active.

- Message tag: CMD_TDYNAMIC_CREATE

```
Messageclass       :XMK_MICRO_TRACER
Meaning            :Trace of SDL Dynamic create
Sent when          :Execution of that SDL symbol
Direction          :T->H
Typedefinition     :xmk_T_CMD_TDYNAMIC_CREATE
Conditional compile:XMK_ADD_TDYNAMICCREATE
```

Message description: The message is sent for each dynamically created SDL process, when its start transition is executed, and when the trace is set for this symbol. Furthermore, the trace must be active.

- Message tag: CMD_TDECISION

```
Messageclass       :XMK_MICRO_TRACER
Meaning            :Trace of SDL Decision
Sent when          :Execution of that SDL symbol
Direction          :T->H
Typedefinition     :xmk_T_CMD_TDECISION
Conditional compile:XMK_ADD_TDecisionValue
```

Message description: The message is sent when an SDL decision is executed and the trace is set for this symbol. Furthermore, the trace must be active.

- Message tag: CMD_TSAVE

```
Messageclass       :XMK_MICRO_TRACER
Meaning            :Trace of SDL Save
Sent when          :Execution of that SDL symbol
Direction          :T->H
Typedefinition     :xmk_T_CMD_TSAVE
Conditional compile:XMK_ADD_TSAVE
```

Message description: The message is sent when a signal save is performed and when the trace is set for this symbol. Furthermore, the trace must be active.

- Message tag: CMD_TTIMER

```
Messageclass        :XMK_MICRO_TRACER
Meaning             :Trace of SDL Timer
Sent when           :Execution of that SDL symbol
Direction           :T->H
Typedefinition      :xmk_T_CMD_TTIMER
Conditional compile:XMK_ADD_TTIMER
```

Message description: The message is sent when an SDL timer has expired and is input into the owner process and the trace is set for this symbol. Furthermore, the trace must be active.

- Message tag: CMD_TINPUT

```
Messageclass        :XMK_MICRO_TRACER
Meaning             :Trace of SDL Input
Sent when           :Execution of that SDL symbol
Direction           :T->H
Typedefinition      :xmk_T_CMD_TINPUT
Conditional compile:XMK_ADD_TINPUT
```

Message description: The message is sent when an SDL input is performed and when the trace is set for this symbol. Furthermore, the trace must be active.

- Message tag: CMD_TPROCEDURE

```
Messageclass        :XMK_MICRO_TRACER
Meaning             :Trace of SDL Procedure
Sent when           :Execution of that SDL symbol
Direction           :T->H
Typedefinition      :xmk_T_CMD_TPROCEDURE
Conditional compile:XMK_ADD_TNAME
```

Message description: The message is sent when an SDL procedure is entered and when the trace is set for this symbol. Furthermore, the trace must be active.

- Message tag: CMD_TOUTPUT

```
Messageclass        :XMK_MICRO_TRACER
Meaning             :Trace of SDL Output
Sent when           :Execution of that SDL symbol
Direction           :T->H
Typedefinition      :xmk_T_CMD_TOUTPUT
Conditional compile:XMK_ADD_TOUTPUT
```

Message description: The message is sent when an SDL output is performed and the trace is set for this symbol. Furthermore, the trace must be active.

- Message tag: CMD_TDISCARD

```
Messageclass        :XMK_MICRO_TRACER
Meaning             :Trace of SDL Discard
Sent when           :Execution of that SDL symbol
Direction           :T->H
```

```
Typdefinition      :xmk_T_CMD_TDISCARD
Conditional compile:XMK_ADD_TDISCARD
```

Message description: The message is sent when an SDL discard has been detected and when the trace is set for this symbol. Furthermore, the trace must be active.

- Message tag: CMD_PRINT_STRING

```
Messageclass       :XMK_MICRO_TRACER
Meaning            :User command Print string
Sent when          :Execution of fct.xmk_PrintString()
Direction          :T->H
Typedefinition     :xmk_T_CMD_PRINT_STRING
Conditional compile:XMK_ADD_CPRINT_STRING
```

Message description: The message is sent unconditionally, when the C function `xmk_PrintString` is called by the user.

- Message tag: CMD_TIMPLICIT_CONSUMPTION

```
Messageclass       :XMK_MICRO_TRACER
Meaning            :Trace of SDL Implicit consumption
Sent when          :Implicit consumption occurs
Direction          :T->H
Typdefinition      :xmk_T_CMD_TIMPLICIT_CONSUMTION
Conditional compile:XMK_ADD_TIMPLICIT_CONSUMPTION
```

Message description: The message is sent when an SDL implicit consumption is performed and when the trace is set for this symbol. Furthermore, the trace must be active.

- Message tag: CMD_TACTUAL_RESET

```
Messageclass       :XMK_MICRO_TRACER
Meaning            :Trace of SDL Actual reset
Sent when          :SDL timer is actually reset
Direction          :T->H
Typedefinition     :xmk_T_CMD_TACTUAL_RESET
Conditional compile:XMK_ADD_TACTUAL_RESET
```

Message description: The message is sent when an SDL timer is actually reset and the trace is set for this symbol. Furthermore, the trace must be active. An actual reset applies only if a timer was previously set and has not expired.

- Message tag: CMD_TSYS_ERROR

```
Messageclass       :XMK_MICRO_TRACER
Meaning            :Trace of system error
Sent when          :Any kind of systemerror was detected
Direction          :T->H
Typedefinition     :xmk_T_CMD_TSYS_ERROR
Conditional compile:XMK_ADD_TERROR
```

Message description: The message is sent when a system error of any type has been detected and the trace is set for this event. Furthermore, the trace must be active.

- Message tag: CMD_TSYS_SCHEDULE

```
Messageclass       :XMK_MICRO_TRACER
Meaning            :Trace of system schedule event
Sent when          :When scheduling is performed
Direction          :T->H
Typedefinition     :xmk_T_CMD_TSYS_SCHEDULE
Conditional compile:XMK_ADD_TSCHEDULE
```

Message description: The message is sent when the Cmicro Kernel begins scheduling and the trace is set for this event. Furthermore, the trace must be active.

- Message tag: CMD_TSYS_SHOWPRIO

```
Messageclass       :XMK_MICRO_TRACER
Meaning               :Trace of system event : change priority
level
Sent when          :When changing to another prioritylevel
Direction          :T->H
Typedefinition     :xmk_T_CMD_TSYS_SHOWPRIO
Conditional compile:XMK_ADD_TSHOWPRIO
```

Message description: The message is sent when the Cmicro Kernel schedules from one priority level to another and the trace is set for this event. Furthermore, the trace must be active.

- Message tag: CMD_TBETWEEN_SYMBOL

```
Messageclass       :XMK_MICRO_TRACER
Meaning            :Another symbol than input is executed
Sent when          :Another symbol than input is executed
Direction          :T->H
Typedefinition     :xmk_T_CMD_TBETWEEN_SYMBOL
Conditional compile:XMK_ADD_SDLE_TRACE
```

Message description: With this command the process ID of the current executed process is sent and a system wide unique identifier. The SDL Target Tester can force the SDL Editor to highlight the current SDL symbol.

- Message tag: CMD_TAT_FIRST_SYMBOL

```
Messageclass       :XMK_MICRO_TRACER
Meaning            :An Input symbol has bee executed
Sent when          :An Input symbol is executed
Direction          :T->H
Typedefinition     :xmk_T_CMD_TAT_FIRST_SYMBOL
Conditional compile:XMK_ADD_SDLE_TRACE
```

Message description: Similar like CMD_TBETWEEN_SYMBOLS the currently executed SDL input symbol will be highlighted in the SDL Editor.

Formats Concerning the Cmicro Recorder

All messages belonging to the Cmicro Recorder are part of the message class `XMK_MICRO_RECORDER (F2h)`.

- Message tag: CMD_RECORD_COUNT_IND

```
Messageclass        :XMK_MICRO_RECORDER
Meaning             :Show recorded event counter to host
Sent when           :When ENV->SDL stimuli detected
Direction           :T->H
Typedefinition      :xmk_CMD_RECORD_COUNT_IND
Conditional compile:XMK_ADD_MICRO_RECORDER
```

  Message description: The message is sent when the Cmicro Recorder is active and is in record mode. It is sent together with, but always before `CMD_RECORD_OUTPUT_ENV2SDL`.

- Message tag: CMD_RECORD_OUTPUT_ENV2SDL

```
Messageclass        :XMK_MICRO_RECORDER
Meaning             :Indicate stimuli ENV->SDL to host
Sent when           :When ENV->SDL stimuli detected
Direction           :T->H
Typedefinition      :xmk_T_CMD_RECORD_OUTPUT_ENV2SDL
Conditional compile:XMK_ADD_MICRO_RECORDER
```

  Message description: The message is sent when the Cmicro Recorder is active and is in record mode. It is sent after `CMD_RECORD_COUNT_IND`.

- Message tag: CMD_RECORD_OUTPUT_SDL2ENV

```
Messageclass        :XMK_MICRO_RECORDER
Meaning             :unused
Sent when           :-
Direction           :T->H
Typedefinition      :-
Conditional compile:XMK_ADD_MICRO_RECORDER
```

  Message description: This message is currently unused and is never sent.

- Message tag: CMD_PLAY_SECTION_REQ

```
Messageclass        :XMK_MICRO_RECORDER
Meaning             :Request next sections from host
Sent when           :When current cnt reaches debit cnt
Direction           :H->T
Typedefinition      :-
Conditional compile:XMK_ADD_MICRO_RECORDER
```

Message description: The message is sent when the Cmicro Recorder is active and is in play mode. It is sent if the target has to put in the next stimuli (an environment signal).

- Message tag: CMD_PLAY_COUNT

```
Messageclass        :XMK_MICRO_RECORDER
Meaning             :Transfer Debitcounter to target
Sent when           :When target requests next sections
Direction           :H->T
Typedefinition      :xmk_T_CMD_PLAY_COUNT
Conditional compile:XMK_ADD_MICRO_RECORDER
```

Message description: The message is sent when the Cmicro Recorder is active and is in play mode. It is sent if the target requests the next stimuli (an environment signal). It is also sent in the initialization phase of the play mode. Outside the initialization phase, it is always sent together with, but after `CMD_PLAY_OUTPUT_ENV2SDL`.

- Message tag: CMD_PLAY_OUTPUT_ENV2SDL

```
Messageclass        :XMK_MICRO_RECORDER
Meaning             :Transfer stimuli ENV->SDL to target
Sent when           :When target requests next sections
Direction           :H->T
Typedefinition      :xmk_T_CMD_PLAY_OUTPUT_ENV2SDL
Conditional compile:XMK_ADD_MICRO_RECORDER
```

Message description: The message is sent when the Cmicro Recorder is active and is in play mode. It is sent together with `CMD_PLAY_COUNT` but always before `CMD_PLAY_COUNT` if the target request the next stimuli (an environment signal).

- Message tag: CMD_PLAY_OUTPUT_SDL2ENV

```
Messageclass        :XMK_MICRO_RECORDER
Meaning             :unused
Sent when           :-
Direction           :H->T
Typedefinition      :-
Conditional compile:XMK_ADD_MICRO_RECORDER
```

Message description: This message is currently unused and is never sent.

- Message tag: CMD_RECORDER_OFF

```
Messageclass        :XMK_MICRO_RECORDER
Meaning             :Switch off recorder
Sent when           :When at end of replay session detected
Direction           :H->T
Typedefinition      :-
Conditional compile:XMK_ADD_MICRO_RECORDER
```

Message description: This message is sent if the host detected an end of file after the target has requested the next stimuli (an environment signal). It switches the Cmicro Recorder within the target off and execution returns from simulation to real-time.

Formats Concerning the Command and Debug Interface

All messages belonging to the Cmicro Commands are part of the message class `XMK_MICRO_COMMAND (F0h)`.

- Message tag: CMD_SUSPEND_REQ

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Suspend Cmicro Kernel
Sent when          :User typed in command
Direction          :H->T
Typedefinition     :-
Conditional compile:XMK_ADD_CSUSPEND
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_RESUME_REQ

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Resume  Cmicro Kernel
Sent when          :User typed in command
Direction          :H->T
Typedefinition     :-
Conditional compile:XMK_ADD_CRESUME
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_SYSTEM_REINIT_REQ

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Reinitialize SDL system
Sent when          :User typed in command
Direction          :H->T
Typedefinition     :-
Conditional compile:XMK_ADD_CREINIT
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_SYSTEM_SHUTDOWN_REQ

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Shutdown SDL system
Sent when          :User typed in command
Direction          :H->T
Typedefinition     :-
Conditional compile:XMK_ADD_CSHUTDOWN
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_SINGLE_STEP_REQ

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Go into single step
Sent when          :User typed in command
Direction          :H->T
Typedefinition     :-
Conditional compile:XMK_ADD_CSINGLE_STEP
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_DISABLE_TIMER_REQ

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Disable processing of timers
Sent when          :User typed in command
Direction          :H->T
Typedefinition     :-
Conditional compile:XMK_ADD_CDISABLE_TIMER
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_ENABLE_TIMER_REQ

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Enable processing of timers
Sent when          :User typed in command
Direction          :H->T
Typedefinition     :-
Conditional compile:XMK_ADD_CENABLE_TIMER
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_NEXT_STEP_REQ

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Execute next step
Sent when          :User typed in command
Direction          :H->T
Typedefinition     :-
Conditional compile:XMK_ADD_CNEXT_STEP
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_SYSTEM_CONTINUE_REQ

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Continue system, if halted
Sent when          :User typed in command
Direction          :H->T
Typedefinition     :-
Conditional compile:XMK_ADD_CCONTINUE
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_SET_BREAKPOINT_REQ

```
Messageclass        :XMK_MICRO_COMMAND
Meaning             :Set breakpoint
Sent when           :User typed in command
Direction           :H->T
Typedefinition      :xmk_T__CMD_SET_BREAKPOINT_REQ
Conditional compile:XMK_ADD_CBREAK_LOGIC
```

Message description: This message is send if the user typed in the appropriate command.

- Message tag: CMD_BREAK_INPUT_REQ

```
Messageclass        :XMK_MICRO_COMMAND
Meaning             :Set breakpoint on SDL input
Sent when           :User typed in command
Direction           :H->T
Typedefinition      :xmk_T_CMD_BREAK_INPUT_REQ
Conditional compile:XMK_ADD_CBREAK_LOGIC
```

Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_BREAK_STATE_REQ

```
Messageclass        :XMK_MICRO_COMMAND
Meaning             :Set breakpoint on nextstate
Sent when           :User typed in command
Direction           :H->T
Typedefinition      :xmk_T_CMD_BREAK_STATE_REQ
Conditional compile:XMK_ADD_CBREAK_STATE
```

Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_CLEAR_BREAKPOINT_REQ

```
Messageclass        :XMK_MICRO_COMMAND
Meaning             :Clear specified breakpoint from list
Sent when           :User typed in command
Direction           :H->T
Typedefinition      :xmk_T_CMD_CLEAR_BREAKPOINT_REQ
Conditional compile:XMK_ADD_CBREAK_LOGIC
```

Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_CLEAR_ALL_BREAKPOINT_REQ

```
Messageclass        :XMK_MICRO_COMMAND
Meaning             :Clear all breakpoints
Sent when           :User typed in command
Direction           :H->T
Typedefinition      :-
Conditional compile:XMK_ADD_CBREAK_LOGIC
```

Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_QUERY_BREAKPOINT_REQ

  ```
  Messageclass       :XMK_MICRO_COMMAND
  Meaning            :Query the breakpoint list
  Sent when          :User typed in command
  Direction          :H->T
  Typedefinition     :-
  Conditional compile:XMK_ADD_CBREAK_LOGIC
  ```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_QUERY_ERROR_REQ

  ```
  Messageclass       :XMK_MICRO_COMMAND
  Meaning            :Query last error occured in system
  Sent when          :User typed in command
  Direction          :H->T
  Typedefinition     :-
  Conditional compile:XMK_ADD_CQUERY_ERROR
  ```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_NEXT_STATE_REQ

  ```
  Messageclass       :XMK_MICRO_COMMAND
  Meaning            :Modify processstate to value
  Sent when          :User typed in command
  Direction          :H->T
  Typedefinition     :xmk_T_CMD_NEXT_STATE_REQ
  Conditional compile:XMK_ADD_CNEXTSTATE
  ```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_CREATE_PROCESS_REQ

  ```
  Messageclass       :XMK_MICRO_COMMAND
  Meaning            :Create SDL process
  Sent when          :User typed in command
  Direction          :H->T
  Typedefinition     :xmk_T_CMD_CREATE_PROCESS_REQ
  Conditional compile:XMK_ADD_CCREATE_PROCESS
  ```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_TBUF_REQ

  ```
  Messageclass       :XMK_MICRO_COMMAND
  Meaning            :Query Tracebuffer
  Sent when          :User typed in command
  Direction          :H->T
  Typedefinition     :-
  Conditional compile:XMK_ADD_CTBUF
  ```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_KILL_PROCESS_REQ

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Kill process
Sent when          :User typed in command
Direction          :H->T
Typedefinition     :xmk_T_CMD_KILL_PROCESS_REQ
Conditional compile:XMK_USE_CMD_KILL&XMK_ADD_CKILL_PROCESS
```

Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_PCO_REQ

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Define PCO
Sent when          :unused
Direction          :H->T
Typedefinition     :-
Conditional compile:-
```

Message description: This message is currently not implemented and is never sent.

- Message tag: CMD_QUERY_PROCESS_REQ

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Query process state
Sent when          :User typed in command
Direction          :H->T
Typedefinition     :xmk_T_CMD_QUERY_PROCESS_REQ
Conditional compile:XMK_ADD_CQUERY_PROCESS
```

Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_QUERY_QUEUE_REQ

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Query all queues
Sent when          :User typed in command
Direction          :H->T
Typedefinition     :-
Conditional compile:XMK_ADD_CQUERY_QUEUE
```

Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_RMQUEUE_REQ

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Flush all queues
Sent when          :User typed in command
Direction          :H->T
Typedefinition     :-
Conditional compile:XMK_ADD_CRM_QUEUE
```

Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_SET_TIMER_REQ

```
Messageclass        :XMK_MICRO_COMMAND
Meaning             :Set SDL timer
Sent when           :User typed in command
Direction           :H->T
Typedefinition      :xmk_T_CMD_SET_TIMER_REQ
Conditional compile:XMK_ADD_CSET_TIMER
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_RESET_TIMER_REQ

```
Messageclass        :XMK_MICRO_COMMAND
Meaning             :Reset SDL timer
Sent when           :User typed in command
Direction           :H->T
Typedefinition      :xmk_T_CMD_RESET_TIMER_REQ
Conditional compile:XMK_ADD_CRESET_TIMER
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_QUERY_TIMER_REQ

```
Messageclass        :XMK_MICRO_COMMAND
Meaning             :Query timertables
Sent when           :User typed in command
Direction           :H->T
Typedefinition      :-
Conditional compile:XMK_ADD_CQUERY_TIMER
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_ACTIVE_TIMER_REQ

```
Messageclass        :XMK_MICRO_COMMAND
Meaning             :Query, if timer active
Sent when           :User typed in command
Direction           :H->T
Typedefinition      :xmk_T_CMD_ACTIVE_TIMER_REQ
Conditional compile:XMK_ADD_CACTIVE_TIMER
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_FLUSH_SIGNAL_BY_SID_REQ

```
Messageclass        :XMK_MICRO_COMMAND
Meaning             :Flush signals with given ID
Sent when           :User typed in command
Direction           :H->T
Typedefinition      :xmk_T_CMD_FLUSH_SIGNAL_BY_SID_REQ
Conditional compile:XMK_ADD_CFLUSH_SIGNAL_BY_SID
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_FLUSH_SIGNALS_BY_PID_REQ

```
Messageclass        :XMK_MICRO_COMMAND
Meaning             :Flush signals of process
Sent when           :User typed in command
Direction           :H->T
Typedefinition      :xmk_T_CMD_FLUSH_SIGNALS_BY_PID_REQ
Conditional compile:XMK_ADD_CFLUSH_SIGNALS_BY_PID
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_RESET_ALL_TIMERS_REQ

```
Messageclass        :XMK_MICRO_COMMAND
Meaning             :Reset all timers
Sent when           :User typed in command
Direction           :H->T
Typedefinition      :xmk_T_CMD_RESET_ALL_TIMERS_REQ
Conditional compile:XMK_ADD_CRESET_ALL_TIMERS
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_SCALE_TIMER_REQ

```
Messageclass        :XMK_MICRO_COMMAND
Meaning             :Scale timers
Sent when           :User typed in command
Direction           :H->T
Typedefinition      :xmk_T_CMD_SCALE_TIMER_REQ
Conditional compile:XMK_ADD_CSCALE_TIMER
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_START_SDL_REQ

```
Messageclass        :XMK_MICRO_COMMAND
Meaning             :Starts the SDL system
Sent when           :User typed in command
Direction           :H->T
Typedefinition      :xmk_T_CMD_START_SDL_REQ
Conditional compile:-
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_OUTPUT_TO_REQ

```
Messageclass        :XMK_MICRO_COMMAND
Meaning             :Sends a signal from the environment
Sent when           :User typed in command
Direction           :H->T
Typedefinition      :xmk_T_CMD_OUTPUT_TO_REQ
Conditional compile:-
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_OUTPUT_INTERNAL_REQ

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Sends an internal signal
Sent when          :User typed in command
Direction          :H->T
Typedefinition     :-
Conditional compile:-
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_GET_CONFIG_REQ

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Gets the target’s configuration
Sent when          :User typed in command
Direction          :H->T
Typedefinition     :-
Conditional compile:-
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_REC_OFF

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Switch recorder off
Sent when          :User typed in command
Direction          :H->T
Typedefinition     :-
Conditional compile:-
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_REC_PLAY

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Switch recorder to play mode
Sent when          :User typed in command
Direction          :H->T
Typedefinition     :-
Conditional compile:-
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_REC_RECORD

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Switch recorder to record mode
Sent when          :User typed in command
Direction          :H->T
Typedefinition     :-
Conditional compile:-
```

  Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_TRACE_ON

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Switch trace on in general
Sent when          :User typed in command
Direction          :H->T
Typedefinition     :-
Conditional compile:-
```

Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_TRACE_OFF

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Switch trace off in general
Sent when          :User typed in command
Direction          :H->T
Typedefinition     :-
Conditional compile:-
```

Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_TRACE_PROCESS

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Modify traceoptions for process
Sent when          :User typed in command
Direction          :H->T
Typedefinition     :xmk_T_CMD_TRACE_PROCESS
Conditional compile:-
```

Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_TRACE_SIGNAL

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Modify traceoptions for signal
Sent when          :User typed in command
Direction          :H->T
Typedefinition     :xmk_T_CMD_TRACE_SIGNAL
Conditional compile:-
```

Message description: This message is sent if the user typed in the appropriate command.

- Message tag: CMD_BREAK_HIT_IND

  ```
  Messageclass       :XMK_MICRO_COMMAND
  Meaning            :Indicate breakpoint hit to host
  Sent when          :Breakpoint hit in target
  Direction          :T->H
  Typedefinition     :xmk_T_CMD_BREAK_HIT_IND
  Conditional compile:XMK_ADD_CBREAK_LOGIC
  ```

  Message description: This message is sent if one of the breakpoints defined in the breaklist has been hit. The SDL system is halted and waits on the next commands from that point on.

- Message tag: CMD_ERROR_CNF

  ```
  Messageclass       :XMK_MICRO_COMMAND
  Meaning            :Negative acknowledge on command
  Sent when          :A command couldn’t be executed
  Direction          :T->H
  Typedefinition     :-
  Conditional compile:-
  ```

  Message description: This message is sent if the target was not able to respond to a command from the user.

- Message tag: CMD_TBUF_CNF

  ```
  Messageclass       :XMK_MICRO_COMMAND
  Meaning            :Indicate Tracebuffercontents
  Sent when          :Response to user command
  Direction          :T->H
  Typedefinition     :-
  Conditional compile:-
  ```

  Message description: This message is currently not implemented and will never be sent.

- Message tag: CMD_OKAY_CNF

  ```
  Messageclass       :XMK_MICRO_COMMAND
  Meaning            :Command acknowledge
  Sent when          :Command correctly executed
  Direction          :T->H
  Typedefinition     :-
  Conditional compile:unconditional
  ```

  Message description: This message is sent from target to host as a response for all commands which do not carry parameters of any kind. It is used within the host library as a general confirmation of most messages.

- Message tag: CMD_QUERY_PROCESS_CNF

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Indicate process state
Sent when          :Response to user command
Direction          :T->H
Typedefinition     :xmk_T_CMD_QUERY_PROCESS_CNF
Conditional compile:XMK_ADD_CQUERY_PROCESS
```

  Message description: This message is sent as a reaction to the message `CMD_QUERY_PROCESS_REQ`. It carries information about the process state.

- Message tag: CMD_QUERY_QUEUE_CNF

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Indicate queue states
Sent when          :Response to user command
Direction          :T->H
Typedefinition     :xmk_T_CMD_QUERY_QUEUE_CNF
Conditional compile:XMK_ADD_CQUERY_QUEUE
```

  Message description: This message is sent as a reaction to the message `CMD_QUERY_QUEUE_REQ`. It carries information about the queue state.

- Message tag: CMD_QUERY_TIMER_CNF

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Indicate timer table
Sent when          :Response to user command
Direction          :T->H
Typedefinition     :xmk_T_CMD_QUERY_TIMER_CNF
Conditional compile:XMK_ADD_CQUERY_TIMER
```

  Message description: This message is sent as a reaction to the message `CMD_QUERY_TIMER_REQ`. It carries information about the timer tables.

- Message tag: CMD_ACTIVE_TIMER_CNF

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Indicate, if timer is active
Sent when          :Response to user command
Direction          :T->H
Typedefinition     :xmk_T_CMD_ACTIVE_TIMER_CNF
Conditional compile:XMK_ADD_CACTIVE_TIMER
```

  Message description: This message is sent as a reaction to the message `CMD_ACTIVE_PROCESS_REQ`. It carries information about the state of a timer according to SDL semantics.

- Message tag: CMD_QUERY_BREAKPOINT_CNF

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Indicate breaklist to host
Sent when          :Response to user command
Direction          :T->H
Typedefinition     :xmk_T_CMD_QUERY_BREAKPOINT_CNF
Conditional compile:-
```

  Message description: This message is sent as a reaction to the command `CMD_QUERY_BREAKPOINT_REQ`. It carries the breaklist from target to host.

- Message tag: CMD_QUERY_ERROR_CNF

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Indicate last error occured
Sent when          :Response to user command
Direction          :T->H
Typedefinition     :xmk_T_CMD_QUERY_ERROR_CNF
Conditional compile:-
```

  Message description: This message is sent as a reaction to the message `CMD_QUERY_ERROR_REQ`. It carries information about the last error which has occurred in the target.

- Message tag: CMD_START_SDL_IND

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Indicate ready for configuration
Sent when          :Response to user command
Direction          :T->H
Typedefinition     :-
Conditional compile:-
```

  Message description: This message is sent when the target is ready for configuration from the host. In this state the target's SDL system is not initialized.

- Message tag: CMD_START_SDL_CNF

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Confirm SDL system start
Sent when          :Response to user command
Direction          :T->H
Typedefinition     :-
Conditional compile:-
```

  Message description: This message is sent as a reaction to the message `CMD_START_SDL_REQ`.

- Message tag: CMD_GET_CONFIG_IND

```
Messageclass       :XMK_MICRO_COMMAND
Meaning            :Send target's configuration
Sent when          :Automatic at startup
Direction          :T->H
Typedefinition     :xmk_GET_CONFIG_IND
Conditional compile:-
```

Message description: This is the second message which shows the target configuration. This means the scaling configuration (`ml_mcf.h`) as well as run-time configuration.

- Message tag: CMD_GET_DATA_CONFIG_IND

```
Messageclass         :XMK_MICRO_COMMAND
Meaning              :see description below
Sent when            :Automatic at startup
Direction            :T->H
Typedefinition       :xmk_GET_DATA_CONFIG_IND
Conditional compile:XMK_USE_AUTO_MCOD
```

  Message description: This is the first message which indicates the target's length, alignment and endians of the basic c-data types.

- Message tag: CMD_GET_CONFIG_CNF

```
Messageclass         :XMK_MICRO_COMMAND
Meaning              :
Sent when            :Response to user command
Direction            :T->H
Typedefinition       :xmk_T_CMD_GET_CONFIG_CNF
Conditional compile:-
```

  Message description: This message is the reply message to the host's `CMD_GET_CONFIG_REQ`. It sends the same information as `CMD_GET_CONFIG_IND`.

Used Default Protocol

The data link part of the communications link is represented by the data link module and the coder module. The coder module ensures that each data link frame is encoded / decoded according to definitions of the default protocol.

- The protocol delivered with the Cmicro Package is easy to modify.
- It allows transfer of all the data necessary for each part of the SDL Target Tester
- It allows the transfer of user defined data to be integrated.
- No handshake procedure or retransmission request is implemented

Each frame of the Cmicro Protocol is seen from the physical layer as binary data.

Each frame consists of the following items:

- **`Message Class`**
  Used to classify the information (is it SDL Target Tester or User de-

fined data?). The codes F3h to FFh are free to use by the user, the other ones are reserved for the SDL Target Tester.

- **`Message Tag`**
  Used to differentiate the different information either of the SDL Target Tester or the user defined data. For user defined data, the message tag is free to use. If more than 255 bytes are to be transferred, the most significant bit of the message tag is set, otherwise it is cleared. This means there are up to 128 free values for the message tag.

- **`SYNC`**
  Fixed marker 0xaa is used to check if the receiver is synchronized with the sender. This is to detect problems in the start-up phase of host and target. This field of the message header is used in combination with the message class (>= F0h) to re-synchronize the data stream in the host's data link layer (sdtgate).

- **`Message Length`**
  Used to store the information about how many bytes are to follow after the length information, excluding the CRC information. The message length is normally encoded in one octet, which means the following data has a maximum length of 255 octets. If there is more data the most significant bit of the message tag is used to indicate that the message length exceeds 255 octets. In that case the message length is encoded in 2 octets which makes it possible to transfer 65536 octets of data.

- **`Data`**
  Free formatted data (user data or data for the SDL Target Tester).

- **`Checksum`**
  Used to check if the frame is OK on the receiving site. The checksum is built beginning from tag1 to the last data byte by a modulo-8 operation. The sync byte is included in the CRC.

The following is an example of a coded Cmicro Protocol frame for a MS DOS PC, where the message tag is coded as `CMD_TSTATIC_CREATE`. Note this is only an example and the real coding may be different.

Example 592: Coding for MS-DOS PC

Message-class	Message-tag	SYNC	Message-length	-any kind of data-	Checksum
0xF1	0x06	0xAA	0x02		0x??

The functions described in "The API on Target" on page 3585 can be used to encode / decode these frames.

The Communications Link's Target Site

The Data Link Layer

General

The data link of the communications link usually handles the separately message transfer of information from host and to host.

The data link uses the services and functions of the physical layer in order to transmit and receive information.

The data link offers functionality that makes it possible to send and to receive messages via the physical layer assigned to the data link. This functionality can be used by all programs (if implemented so). Mainly it is the SDL Target Tester which uses the data link layer services in order to send and to receive messages. But the users may want to send and receive messages, too. This is possible by using the data link layer functions and by regarding the defined protocol used for each message.

The data link layer offers the following functionality:

- Receive a message from any part of the system like kernel SDL processes or user defined C functions.
- Store messages in a buffer.
- If the physical layer is ready with the transfer of the last message, send the next message to the physical layer.
- If the physical layer indicates a complete message, store it into the receiving buffer.

- Deliver the messages, which have been stored within the receiving buffer to the requesting program parts.
- Poll the physical layer.
- Handle both directions in parallel.
- CRC procedures (if necessary).

The data link layer is represented by the following functions, which are exported by the data link (`mg_dl`) module.

Data Link Functions Used by SDL Target Tester

The SDL Target Tester's target library uses the following C functions of the link and buffering file `mg_dl.c`. The body of each C function may be rewritten according to the users needs.

`xmk_DLInit`	Initialization of data link
`xmk_DLQuery`	Polling the receiver of data link
`xmk_DLRead`	Reading a message from data link
`xmk_DLWrite`	Writing a message to data link
`xmk_DLDeinit`	Deinitialization of data link

These functions are fixed into the SDL Target Tester's data link layer and they handle the mapping and the access to the physical layer.

The Physical Layer

General

Any type of physical communications link can be used to connect host and target, e.g. it could be a V.24, a TCP/IP or a IEEE 488 interface or it could be any other serial or parallel interface.

The physical layer of the communications link normally handles the byte-wise transfer of messages from host to target and the other way around.

It uses the services of the underlaying hardware like interrupts, CPU registers or memory mapped I/O.

The physical layer offers functionality to the data link layer in order to make it possible to transfer any kind of information, including SDL Target Tester data and / or user data.

The physical layer offers the following functionality:

- Receive a buffer which is to be transferred from the data link layer.
- Send this buffer byte-wise via special hardware (or OS - system call) to the receiving system (either host or target).
- If all characters of this buffer have been transmitted, request next buffer.
- Receive characters from the sending system via special hardware (or OS - system call).
- Collect characters until a message is complete; characters are stored within a temporary buffer.
- If message is complete in temporary buffer, transfer it to the data link layer.
- Handle both directions simultaneously.
- CRC procedures (if necessary).

Physical Layer Functions Used by the Data Link Layer (Default)

As described above the data link layer of the SDL Target Tester's target library is shown in the files `mt_cod.c` and `mg_dl.c`. Please note that `mg_dl.c` is only a template with a default implementation for the connection between data link and physical layer.

Distributed with the SDL Target Tester's target library, there are several template files for the physical layer (TCP/IP and V.24 interfaces) contained in the `cmicro/template/commlink` directory in the installation directory. These template files are tested only for the specific hardware in the Cmicro Package test environments. An error free compilation or execution cannot be guaranteed for the user's target hardware.

There are physical layer templates for the following target hardware:

- Intel 8051 derivatives -> 8051_v24.[ch]
  Franklin/Keil C51 and IAR C51 compilers

- Siemens 80C166 -> 80166v24.[ch]
  BSO/Tasking 80C166 compiler
- MS Windows NT/95 -> win32v24.[ch]
  Borland C++ 5.x compiler, Microsoft VC++ 6.00
- MS Windows NT/95 -> win_sock.[ch]
  Borland C++ 5.x compiler, Microsoft VC++ 6.00
- SunOS 5.x -> ux_sock.[ch]
  GNU gcc, Sun WorkShop compilers CC and cc
- HP-UX 10.xx -> ux_sock.[ch]
  GNU gcc, HP compiler aCC and cc

For use in the target executable, one of the modules listed above is automatically included in `mg_dl.c`. Which physical layer module is included depends on the compiler selected and the flags set in `ml_mcf.h` (e.g. `XMK_USE_V24 or XMK_USE_SOCKETS`).

The template module `mg_dl.c` works as follows:

The function `xmk_DLInit` initializes the buffers used for the data link and calls the initialization of the physical layer (by default this is `xmk_V24Init`).

`xmk_DLDeinit` is the counterpart of `xmk_DLInit` and frees the interface device. This function is normally empty in microcontroller applications but has to be used on PCs, under Microsoft Windows or UNIX.

`xmk_DLWrite` is called when a complete frame should be sent to the host. There are two ways the connection to the physical layer can be handled.

1. Blocked

   If a frame is to be sent it is necessary that the previous frame has already been completely sent. If not, the data link has to wait until the previous frame is sent. See the module `mg_dl.c` (for the compilers KEIL_C51 and IARC51) to get an idea of this connection.

2. Unblocked

   If the frame should be sent, it is handed over to a ring buffer. Although the previous frame has not been sent completely, SDL can execute because the data link does not have to wait.

> One exception must be mentioned: If the transmitter's ring buffer (used for V.24 communication) is full (macro `XMK_MAX_SEND_ENTRIES`) the data link will still have to wait here. Also the size allowed for the ring buffer's entries must be great enough (to verify this see the macro `XMK_MAX_SEND_ONE_ENTRY`.

`xmk_DLRead` will be polled with each cycle of the SDL loop `xmk_RunSDL` (see module `mk_main.c`). If the transmitter and the receiver of the physical layer are not implemented as interrupt service routines (which is strongly recommended) the transmit and receive functions have to be called here (`xmk_V24Receive` and `xmk_V24Send` for the distributed V.24 templates). In addition for the V.24 interface the XON handshake has to be called here doing `xmk_V24Handshake`.

As in `xmk_DLWrite` there are two ways the receiver buffers can be handled in `xmk_DLRead`. It is also possible to use a ring buffer for the receiver's side. Similar as in `xmk_DLWrite` it is necessary to fill the macros `XMK_MAX_RECEIVE_ONE_ENTRY` and `XMK_MAX_RECEIVE_ENTRIES` with useful values.

Steps to Implement a Communications Link

Extension to the Template Makefile

In the distribution of the Cmicro Package there are several template makefiles.

On UNIX the existing makefile templates are called `makeoptions.no_tester` and `makeoptions.with_tester`.

In Windows the existing makefile templates are called `makeno_t.opt` and `makew_t.opt`.

The intention is to give one template makefile where the SDL Target Tester is not included (which means that there is no compilation of any SDL Target Tester C code), and to give one template makefile including the SDL Target Tester.

In the template makefiles that include the SDL Target Tester, it is necessary to define the communications link software by adding the name of the C module(s) that contain(s) this. This can be achieved by changing the list of the object files given in `sctLINKTARGET_WITH_TESTER` (e.g. `8051_v24`.suffix). In addition, the compilation rules for this/these additional module(s) must be specified as well in the makefile.

Finally, the makefile name must be changed into `makeoptions` on **UNIX**, and into `make.opt` in **Windows**.

> **Note:**
> The `makeoptions` and `make.opt` files below the predefined kernel directories in `<sdtdir>/SCMA*` are not intended to be used for target or target debug applications.

Adapting the Data Link Layer

The data link layer for the SDL Target Tester on target site is represented by the module `dl.c` with the functions

```
-  xmk_DLInit()
-  xmk_DLRead()
-  xmk_DLWrite()
-  xmk_DLQuery()
-  xmk_DLDeinit()
```

Several ways to fill these functions are already given in `mg_dl.c`.

The data link layer can be build using a ring buffer for all the trace message or writing each message directly to the physical layer.

Furthermore the data link layer must be build to work together with the physical layers (E.g. ISR driven or not).

It is recommended to have a look into the distributed template `mg_dl.c` which shows different ways.

- For using the ring buffer, please follow the flag `TCC80166`.
- For using no ring buffer, please follow the flags `IARC51` or `KEIL_C51`.
- For using an ISR driven physical layer, please follow the flags `IARC51` and `KEIL_C51`.
- For using a not ISR driven physical layer, please follow the flag `TCC80166`.

Adapting the Physical Layer

There are given tested physical layers for the micro controllers 8051 and 80166.

The structure of these physical layers should be taken and filled with the functions for the needed communications link. E.g. the function to write to a serial interface must be replaced by a function to write to an Ethernet interface.

The Communications Link's Host Site

The sdtgate is one executable of the SDL Target Tester's tool chain on host site. It is built to simplify the communication between the host and the target.

In this release of the SDL Target Tester, the sdtgate is built using a V.24 interface and the sockgate is build using a TCP/IP interface. But as you can see in this subsection, it will be very simple to implement any other communication interface as a new executable.

Communication between the SDL Target Tester Executables

The SDL Target Tester executables `sdtmtui`, `sdtmt` and `sdtgate` communicate with the help of the Cmicro Postmaster. The Cmicro Postmaster handles all the communication between the SDL Target Tester tool chain and is started automatically when the SDL Target Tester is invoked.

The Default V.24 Connection: Sdtgate

Fork Parameters for the Sdtgate

There are two groups of fork parameters handed over to the sdtgate.

The Sdtgate Device Specifications

Inside the SDL Target Tester configuration file `sdtmt.opt` there are 7 entries for the gateway executable.

Example 593: On UNIX ––––––––––––––––––––––––––––––––––––––

```
USE_GATE                <pathname>/sdtgate
GATE_CHAR_PARAM_1       /dev/ttya
GATE_CHAR_PARAM_2       0
GATE_CHAR_PARAM_3       0
GATE_INT_PARAM_1        9600
GATE_INT_PARAM_2        0
GATE_INT_PARAM_3        0
```

–––

On UNIX, it muse be ensured that the device (`/dev/ttya` above) is accessible. Ask the system administrator if there are problems to open this device.

Example 594: In Windows ––––––––––––––––––––––––––––––––––––

```
USE_GATE                <pathname>/sdtgate.exe
GATE_CHAR_PARAM_1       COM2
GATE_CHAR_PARAM_2       0
GATE_CHAR_PARAM_3       0
GATE_INT_PARAM_1        9600
GATE_INT_PARAM_2        0
GATE_INT_PARAM_3        0
```

–––

USE_GATE gives the name of the used gate only.

Inside the delivered sdtgate for V.24 interface only the GATE_CHAR_PARAM_1 (a string) and the GATE_INT_PARAM_1 (an integer) are used. For future developments there are 2 further string parameters and two further integer parameters which are handed over when forking the gate.

The delivered version of the sdtgate uses fixed values for parity, start bits, stop bits...

These are: databits = 8, startbits = 1, stopbits = 1, parity = none

Coding Rules

There are two coding rule pieces of information the gate needs to work.

These are the length of a character on the target in octets (normally 1) and the position of the character. This value is necessary for special micro controller memory layouts e.g. the MSP58C80 gets the character length of 2 octets but the used character is the lower octet only. In this case the character position is 1. Normally if the character size is 1 the character position is 0.

The TCP/IP Connection: Sockgate

The Sockgate Device Specifications

Like the sdtgate, the sockgate is using only 3 of the 7 entries for the gateway executable inside the SDL Target Tester configuration file `sdtmt.opt`.

Example 595: ---

```
USE_GATE               <pathname>/sockgate
GATE_CHAR_PARAM_1      207.46.138.0
GATE_CHAR_PARAM_2      0
GATE_CHAR_PARAM_3      0
GATE_INT_PARAM_1       9000
GATE_INT_PARAM_2       0
GATE_INT_PARAM_3       0
```

USE_GATE gives the name of the used gate only.

It must be ensured that the IP-address (`207.46.138.0` above) is accessible.

Inside the delivered sockgate only the `GATE_CHAR_PARAM_1` (a string) and the `GATE_INT_PARAM_1` (an integer) are used. The delivered version of the sockgate uses an address string (a valid IP-address) and a port number (valid between 1 and 65535).

Building an Custom Made Communications Link

All the C files and libraries necessary to re-build the default gateway sdtgate are delivered with the SDL Target Tester. The files for a custom built gateway are named usergate to distinguish between the custom and the delivered sdt- and sockgate.

The Modules Building a Usergate

The following files are part of the usergate:

```
cmicro
   + sdtmpm
   |    + hppa
   |    |    + sdtmlib.a (HP-UX version)
   |    + sunos5
   |    |    + sdtmlib.a (SunOS 5 version)
   |    + wini386
   |         + sdtmpmbcb.lib (Borland C++ Builder version)
   |         + sdtmpmcl.lib (Microsoft C++ 6.0 version)
   |         + sdtmpmcl5.lib (Microsoft C++ 5.0 version)
   + mcod
   |    + mcodfnc.[ch]pp
   |    + mcodlib.[ch]pp
   |    + mcod.h
   + sdtgate
   |    + mg_ctrl.[ch]
   |    + usergate.rc (Windows only)
   |    + hppa
   |    |    + usergate.[ch] (HP-UX version)
   |    + sunos5
   |    |    + usergate.[ch] (SunOS 5 version)
   |    + wini386
   |         + usergate.[ch] (Windows version)
   + template
        + mg_dl.[ch]
        + commlink
             + <interface>.[ch]
```

Files with suffix `.h` and `.hpp` are not listed but are still being used.

<interface> stands for the selected interface type.

On UNIX the module `ux_v24.c` is used (`ux_sock.c` for sockets).

In Windows the module `win32v24.c` is used (`win_sock.c` for sockets).

The purpose of each module is as follows:

- `mcodfnc.cpp`

  This module delivers all functions to decode and encode the messages sent between the different executables of the SDL Target Tester tool chain. All functions are contained within a C++ class called ClassHostMessage.

  As these functions have to be suitable for the executables sdtmt and sdtmtui, no changes need be made by the user.

- mcodlib.cpp

  This module is automatically generated by the Message Coding Generator (MCOD) and delivers a decode or encode strategy for each message that can be exchanged between the parts of the SDL Target Tester tool chain.

No modifications can be done inside this module.

- mcod.h

  Like `mcodlib.cpp` this file is generated by MCOD too. It contains all MessageTags for the communication between sdtmtui, sdtmt and sdtgate and no modifications are to be done here.

- mg_ctrl.c

  In this module the reactions of the messages exchanged between the SDL Target Tester tool chain are invoked. Like `mcodlib.cpp` and `mcodfnc.cpp` there is no need to make any modification inside this module because the communication between the different parts of the tool chain depends on it.

- usergate.rc **(Windows only)**

  This resource file is for use in MS Windows executables only. It just needs to be compiled and to be linked together with the other object files.

- usergate.c

  This file handles all the initialization and communication of the sdtgate with the Cmicro Postmaster.

  This file is platform specific and needs to be modified in some cases, e.g. when using the V.24 or socket interface on MS Windows a Windows Message is generated when a character is received or completely sent. The reaction on the WM_COMMNOTIFY message can be removed here if another interface is used.

- sdtmlib.a/sdtmpmbc.lib/sdtmpmcl.lib

  This library presents the connection to the Cmicro Postmaster.

- <Interface>.c

  The last of the listed files needs some modifications if the communication interface is to be changed.

  There are eight functions inside this module

  – `xmk_V24Init`() and `xmk_SocketInit()`
    These functions are called only once after the sdtgate has been forked. All initialization of the communication device have to be done here.

Errors should be given as plain text and will be sent to the user interface.

- `xmk_V24Deinit()` and `xmk_SocketDeinit()`
  These functions may not need to be filled for the wanted communication interface. If it has to, all deinitialization can be done here. This function is called directly before exiting the gate.
- `xmk_V24Handshake()`
  The V.24 RS-232 protocol describes the use of the XON - XOFF communication. In the SDL Target Tester's host - target communication XON characters sent only every second. If using another communication protocol (like RS-422/RS-485) this function must be left empty.
- **`xmk_V24Receive()`** and `xmk_SocketReceive()`
  This function does several things.
  At first the communication interface is polled and all available characters are taken from the interface. Then the gate checks the received character stream for the correct sync field position, the length field is read and the amount of characters used to build one complete target message is sent to sdtmt, unless there are no complete messages left.
  The re-synchronization of the data stream is also done in this function if the sync field is not in the correct position.
- **`xmk_V24Send()`** and `xmk_SocketSend()`
  If the gate gets a message from sdtmt which is to be transmitted to the target, this function is called.
  There are several ways to transmit the whole message. For instance, on Windows NT it is possible to hand over a character buffer to the serial interface. The rest of the transmission is being done by the interface itself.
- `xmk_ExtractControl()` / `xmk_MaskControl()`
  These two functions are for use with an XON-XOFF protocol. If a message contains an XON or XOFF character, in front of this XON or XOFF an ESC is inserted in `xmk_MaskControl()` to differ this (XON/XOFF-) character from the control XON/XOFF.
  On the other hand if a message containing XON/XOFF characters is received, the function `xmk_ExtractControl()` removes all ESC from the data stream.

These functions are called from `xmk_V24Send()` and `xmk_V24Receive()` only and the function calls should be removed if the new communication interface does not support the XON-XOFF protocol.

Compiler Settings

- **On UNIX:**

  The following defines have to be made. The user can use the makefile `makefile.template` in the directory `$sdtdir/cmicro/sdtgate` as a template to compile and link his own gateway.

  - `_GCC_`
  - `XMK_UNIX`
  - `SDTGATE`
  - `XMK_USE_V24` (for the default gateway sdtgate)
  - `XMK_USE_SOCKETS` (for the socket gateway)

- **In Windows:**

  The source code for the sdtgate is tested only in combination with the Borland C++ 5.x compiler and the Microsoft C++ compiler 6.x on a Microsoft Windows NT/95 installation.

  The following defines have to be made. There is a Borland project file called `usergate.ide` that the user may use to compile and link his own gateway with the Borland compiler.

  - `BORLAND_C` or `MICROSOFT_C`
  - `XMK_WINDOWS`
  - `SDTGATE`
  - `XMK_USE_V24` (for the default gateway sdtgate)
  - `XMK_USE_SOCKETS` (for the socket gateway)

Current Restrictions for Communications Link Software

SDL Restrictions

It is impossible to transfer pointer values via the communications link. An example may be the SDL predefined sort charstring which is by default implemented as a pointer in C.

To avoid problems, users must not use SDL sorts that are implemented as pointers (like charstring), whenever it is required to send / receive them via the communications link (e.g. Cmicro Tracer or Cmicro Recorder functionality). This means, that this kind of sorts should only be used in signals inside the SDL system but not at it's environment.

Message Length

The maximum length of the message length is restricted.

An explanation follows:

Each message is transferred as:

```
Header + Data + CRC
```

The length field contained in the header may represent the values from 0 to 255. If the MSB in the message tag is set to "1", then the length field is expected to be a 2 octet values, which makes it possible to represent the value from 256 to 65535.

However, the length field consisting of 2 octets is currently not implemented.

In fact, no message may be greater than 4 + 255 + 1 = 260 characters in total (1 character normally is equal to 1 Byte, which consists of 8 Bits).

Connection of a Newly Implemented Communications Link

The SDL Target Tester uses only a few C function calls in order to send and receive information via the communications link. From the C syntax point of view, users only have to implement the body of these functions and to link them together with the SDL Target Tester's target library into one executable program.

However, when considering timing constraints, critical paths, interrupts and similar things, users have to note the following.

Critical Paths

Critical paths occur in any type of software which uses interrupts. Especially the transfer of information between ordinary C code and interrupt service routines must be considered, e.g. for the communication from the physical layer (which may be implemented as an ISR) to the data link layer (which may be implemented as ordinary C functions).

Critical paths may occur at any place in the software which accesses global C variables.

The beginning of each critical path must be blocked with something like “disable interrupts” (see `XMK_BEGIN_CRITICAL_PATH` in the compiler section of `ml_typ.h`). The end of a critical path must be released with something like “enable interrupts” (see `XMK_END_CRITICAL_PATH`).

Timing Constraints

Often there are timing constraints to be considered within the target system. Timing problems must be prevented. Each ISR must be implemented with a few statements only, in order to make the execution time as short as possible.

Interrupts

C functions which should be compiled as ISRs by the C compiler in use are mostly

marked by a preprocessor directive or by a special C compiler keyword. The C compilers reference manual must be consulted in order to implement ISRs.

Open Interface

The open interface can be used to implement user host executables which are able to communicate with the SDL Target Tester at the target site. The SDL Target Tester at the target site is represented by the target library.

Using the Open Interface

In order to use the open interface in this way, the following are of interest:

- Which messages can be sent to the target. This is described within the earlier subsection “Message Formats” on page 3596.
- Which messages are to be expected from the target. This is also described within “Message Formats” on page 3596.

- The format of these messages. Also described within the earlier subsection “Message Formats” on page 3596.
- The situations in which the messages may be sent. Please refer to the subsection “State Machine and Handshake” on page 3595.
- The situations in which a particular message may be received. Again, please refer to “State Machine and Handshake” on page 3595.

It is not of great interest what the format of the data link frames is because it is possible either to use the existing protocol or to re-implement the protocol (e.g. it may be necessary in order to integrate the SDL Target Tester into an existing communication link’s software).

The host site must send each message in target representation.

For instance, if the target system uses integers as 2 octet values with the ordering higher octet first, then the host must send an integer value with higher octet and then lower octet, irrespective of its own layout.

For the other direction (reception of messages at host site), the host must also map the information according to the relation between the target memory layout and its own internal representation.

Accessing the Data Link by User

The user can send information via the communication link by calling the C function `xmk_DLWrite` with appropriate parameters. The following example shows one possibility.

Example 596 ––

```
#define HelloWorldTag <AnyValueAllowed>

char buffer     [20];
char helloworld [15];
int  length     = 0;
int  write_result ;
strcpy (helloworld, 'Hello World');


buffer [0] = 0x11;
length ++;
/* any value for Messageclass, except      */
/* reserved values for SDL Target Teste    */
/* reserved values begin from 250 to 255   */

buffer [1] = HelloWorldTag;
length ++;
/* any value for Messagetag                */
/* Messagetag is made unique together      */
/* with Messageclass, so that no           */
/* conflict occurs.                        */

buffer [2] = strlen (helloworld);
length ++;
/* length of data for this message         */
/* length is given without calculating     */
/* the CRC field                           */

memcpy (buffer [3], helloworld, strlen (helloworld);
length += strlen (helloworld);
/* It is assumed, that no stringterminator */
/* is included in the message (the length  */
/* can be calculated from Messagelength)   */

buffer [length] = xmk_EvalChkSum (buffer, length);
/* Evaluate the checksum of current buffer */

write_result = xmk_DLWrite (buffer, 5 + strlen (helloworld));
/* transmit buffer via communications link */
```

The user can receive information from the communication link by calling the C function `xmk_DLRead` with appropriate parameters.

Example 597: `xmk_RunSDL ()` ––––––––––––––––––––––––––––––

The following section is partially implemented within the module `mk_main.c`

```
p_data = xmk_DLRead () ;
if (p_data != (char *) NULL)
{
  /* if anything received, decode it    */
  /* according to Cmicro Protocol       */
  result = xmk_Cod_Decode (p_data,
                           &MessageClass,
                           &Messagetag,
                           &MessageLength,
                           buffer,
                           sizeof (buffer));
if (MessageClass == XMK_MICRO_COMMAND)
{
  result = xmk_HandleCmd ( Messagetag, (xmk_U_CMDS*) buffer );
  return;
}

/*
** Received user message
*/
if (MessageClass != 0xF8)
{
  ErrorHandler (-7777);
  return ;
}

if (HelloWorldTag == Messagetag)
{
  /* HelloWorldTag is used in this example to */
  /* identify the hello world message         */
  /* The user can choose any value for the    */
  /* Message tag                               */
  char helloworld [15];

  memset (helloworld, 0, sizeof (helloworld));
  memcpy (helloworld, buffer, MessageLength);
  printf ('%s', helloworld);
}
```

––

The above examples assume that the Cmicro protocol (which is described in “Used Default Protocol” on page 3616) is used.

More Technical Descriptions

The File Formats of Sdtmt

General

This section deals with the general file format of the different files which are written to or read by the SDL Target Tester's tool chain. The section is not of general interest, and need be read only when implementing and connecting other tools to the host. The format of the file containing symbols is described in the subsection "The Symbol Tables" on page 3636.

<infile> and <outfile>

Physically, there is the same file format behind the input files and output files written by sdtmt. These files store the information as received via the communications interface in a 1:1 format. The trace of the SDL execution flow and the information from the record mode is simply copied into the file on the hard disk. The same header information is used as on the communications link. This has some advantages.

Without exception the format of the information stored in <infile> or <outfile> is exactly the same as the format of the Cmicro Protocol. Therefore these files are using the memory layout of the target system.

The reason for this is that binary files are always more compact than readable ASCII files.

The message decoder of the host must be active only when information is to be converted, for example if they are to be displayed. This can decrease the performance a little.

The Symbol Tables

The symbol tables which are used by the SDL Target Tester are automatically generated by the Cmicro SDL to C Compiler. Most of the intelligence to interpret SDL traces and recorded events is implemented on the host system. Normally, the host system has enough memory and performance to do the interpretation, whereas the target has not. This is the reason, why the symbol tables used by the Cmicro Package mainly reside on the host. However, some parts of the symbol table reside where the Cmicro Kernel physically is executed, i.e. within the target system.

The Target Symbol Table

The following table is generated by the Cmicro SDL to C Compiler. For each SDL process, there is one entry in the table representing a pointer to the different trace options for that process.

```
/*****************************************************
**  Symbol trace table
*****************************************************/
#ifdef XMK_ADD_TEST_OPTIONS
XSYMTRACETBL *xSYMTRACETBL[MAX_SDL_PROCESS_TYPES+1] =
{
  (XSYMTRACETBL_ENTRY *)  NULL,
...........
..........
  (XSYMTRACETBL_ENTRY *)  NULL,
  X_END_SYMTRACE_TABLE
};
#endif
```

The Host Symbol Table

Generated File <systemname>.sym

The host symbol table is generated by the Cmicro SDL to C Compiler into a file called `<systemname>.sym`. This file is to be specified when the host site is invoked (see subsection “Invoking the SDL Target Tester’s Host” on page 3525).

In the following, the file format is described with EBNF like syntax:

```
SymbolFile          ::= Header SymbolDescriptions
Header              ::= HeaderComment Timestamp
HeaderComment       ::= “Cmicro .sym file, version x.y”
Timestamp           ::= “TIMESTAMP <GenerationTime>”
SymbolDescriptions  ::= ( DescriptionLine ) ...
DescriptionLine     ::= Depth Subject SDTReference
Depth               ::= [0-9]+
Subject             ::=  SYSTEM IDs
                       | BLOCK IDs
                       | SIGNAL IDs
                       | PROCESS IDs
                       | START IDs
                       | INPUT IDs
                       | OUTPUT IDs
                       | NEXTSTATE IDs
                       | <to be continued>
SDTReference        ::= “#SDTREF(<sdt-reference>)” LF
LF                  ::= “\n”
```

In the following an example is given that was produced with a simple “ping pong” SDL system:

Example 598

```
Cmicro .sym file, version 2.0
TIMESTAMP <GenerationTime>

1 SYSTEM <SystemName>  #SDTREF(sdtref)
2 BLOCK <BlockName>  #SDTREF(sdtref)
3 PROCESS <ProcessName> 0 1  #SDTREF(sdtref)
4 START 0  #SDTREF(sdtref)
4 INPUT 1  #SDTREF(sdtref)
4 INPUT 2  #SDTREF(sdtref)
4 OUTPUT 3  #SDTREF(sdtref)
4 NEXTSTATE State1 1 4  #SDTREF(sdtref)
4 OUTPUT 5  #SDTREF(sdtref)
4 NEXTSTATE - 0 6  #SDTREF(sdtref)
4 TASK 7  #SDTREF(sdtref)
4 STOP 8  #SDTREF(sdtref)
3 PROCESS <ProcessName> 1 1  #SDTREF(sdtref)
4 START 9  #SDTREF(sdtref)
4 INPUT 10  #SDTREF(sdtref)
4 ASSIGNMENT 11  #SDTREF(sdtref)
4 NEXTSTATE State1 1 12  #SDTREF(sdtref)
4 ASSIGNMENT 13  #SDTREF(sdtref)
4 DECISION 14  #SDTREF(sdtref)
4 OUTPUT 15  #SDTREF(sdtref)
4 STOP 16  #SDTREF(sdtref)
4 OUTPUT 17  #SDTREF(sdtref)
4 NEXTSTATE - 0 18  #SDTREF(sdtref)
4 DCL counter Integer  #SDTREF(sdtref)
3 SIGNAL PING 1  #SDTREF(sdtref)
3 SIGNAL PONG 2  #SDTREF(sdtref)
3 SIGNAL PONG_SAY_BYE_BYE 3  #SDTREF(sdtref)
3 SIGNALROUTE SR1  #SDTREF(sdtref)
3 SIGNALROUTE SR2  #SDTREF(sdtref)
```

The first line, `Cmicro .sym file, version x.y`, gives some comment about the Cmicro symbol file version.

The `TIMESTAMP <GenerationTime>` value is used internally for consistency checks with the SDL Target Tester. With this time-stamp, a rough consistency check for the generated files is performed and warning messages are printed out in the case of an inconsistency.

The lines following the time-stamp information, contain in the first position an integer value indicating the depth of the structuring level in which the information is found in SDL. For example, the SDL system is in level 1, a block in the SDL system level is in level 2, and so on.

The next column in these lines describe the subject, e.g. if it is a system, block, channel, signal route, process, procedure, signal, start, input or output, or whatever.

For each subject there are IDs generated, these are used in the target executable program for generating trace information. The IDs begin with

0 (in the case above it is the start symbol of the first process) and end with the number of the last SDL symbol in the system.

The last column in the lines contains a `#SDTREF` reference, which points to the location of the subject in either SDL/GR or SDL/PR.

Internal Symbol Table Structure

The symbols of a trace can be found during the conversion of internal information into a displayable format.

The following three SDL objects are treated:

- Processes
- Signals/Timers
- States

They are handled in the same way. Namely by building a linked list of symbols for processes, a linked list for signals and a linked list for states.

Cmicro Recorder

The information given in the following sections are valid both for host and target and there is no differentiation made between these.

Note:

The SDL Target Tester's Record and Play functions are only available if a Cmicro Recorder license is available.

Type and Amount of Stored Information

As already mentioned in earlier sections, using the Cmicro Recorder makes more sense in those cases where an SDL system is executed in real-time in the target.

That's why the amount of stored information is to be kept very small. Of course, this leads to the problem that not all types of errors can be found with the Cmicro Recorder but there is a good chance to find most of them. The information which the Cmicro Recorder produces at target site and which the Cmicro Recorder uses at host site cannot be configured in their dimension. The packets which are to be transmitted via the communications link are always the same. A technical description follows in the next section.

Record Mode

First, the SDL system executes the start transitions of all statically created processes. At the occurrence of a “counted” symbol, an internal counter variable is incremented (which was set to zero before going into the SDL start-up phase). The counter will simply be incremented until the first environment signal is sent from the environment to a process in the SDL system, or until the first timer expires within the SDL system.

These are special events for the Cmicro Recorder.

In that case, the Cmicro Recorder on the target site sends the following messages to the host:

```
Host <-------- <Counter value> ------------ Target
Host <-------- <Event> -------------------- Target
```

Both messages are stored in the <outfile> on the host.

The counter is reset to zero again and the procedure restarts by counting the “counted” symbols and sending the above messages to the host, either when an environment signal is sent from the environment to a process in the SDL system, or when the first timer expires within the SDL system.

The <outfile> is closed when the user terminates the record mode on the host site, i.e. it is not possible to close the file by ending the record mode on the target site. The record mode can be finished at any time.

After the record mode has been switched off, it is not allowed to switch the record mode or the play mode on again.

The <outfile> on the host now contains messages like this:

```
<trace event 1>
....
<trace event N>
<Counter value>
<Event>
<trace event N+1>
....
<trace event N+1+M>
<Counter value>
<Event>
```

As can be seen, the following sections are stored in the file whereas each section is optional:

```
For all events in the target
           [Store trace section]
           [Store Counter value section]
           [Store Event section]
```

Counted Symbols

The occurrence of the following SDL symbols will be “counted” by an internal counter (a C integer variable):

STATE	The SDL nextstate operation
INPUT	The SDL input operation
SAVE	The SDL save operation
CREATE	The SDL create operation
STOP	The SDL stop operation
STATIC CREATE	Used at system start, appears for each statically created process
DYNAMIC CREATE	Used at start of a dynamically created process
DECISION	SDL decision
TASK	SDL task (not C Code)
OUTPUT	The SDL output operation (see remarks below)
PROCEDURE	SDL procedure call
SET	SDL action: Set timer
RESET	SDL action: Reset timer
DISCARD	SDL Discard
IMPLICIT CONSUMPTION -	SDL implicit transition

A special case is the SDL output operation:

If the output is from one process in the SDL system to another process in the SDL system or if the output is from one SDL process in the SDL system to the environment, the occurrence of this symbol is counted.

Otherwise if the output is from the environment to a process within the SDL system, the event is relevant for the Cmicro Recorder (as the timer expiry is).

C code in an SDL task is normally not counted when recording. The users are free to count C code by calling:

`xmk_Record (CMD_TTASK)` for the record mode and

C code written into an SDL Task (i.e. by using the #CODE directive)

`xmk_Play (CMD_TTASK)` for the play mode.

Play Mode

In play mode, the target accepts signals coming from the SDL environment only from the Cmicro Recorder. These signals are read from the file which was previously written to in record mode.

Caution!

No environment signals may be handled by the user in the C function `xInEnv`! At the beginning of `xInEnv`, there should be a statement like

```
if (xmk_RecorderMode == XMK_RECORDER_PLAY)
   return (XMK_OKAY));
```

In addition, no signal may be sent by any other environment function (like ISRs) to the SDL system.

At the beginning of the play mode, the host reads the first <counter value> from the <infile> and sends it to the target.

```
Host -------- <Counter value> --------------> Target
```

The meaning of this is: Please execute the amount of “counted” symbols as indicated in the <counter value>. The target executes the start transitions of all statically created processes (start-up-phase). When preemption is used, it is possible that a transition which is not a start transition is executed. Each “counted” symbol increments a counter value which is compared against the debit, which was indicated by <counter value>.

If the current counter value reaches the debit counter value, then an environment signal (or timer expiry) plus the next <counter value> is requested by the Cmicro Recorder.

```
Host <------- <Request> -------------------- Target
Host ---------------- <Event> --------------> Target
Host -------- <Counter value> --------------> Target
```

If the target has requested the next <Event> and <Counter value>, the target is waiting for these messages only and is not able to handle other messages. This is not a restriction because the environment is disabled in play mode in the target.

If the end of file of <infile> has been reached, the host indicates the end of the play mode to the target. <Counter value> and <event> are always stored together in record mode. Therefore the end of file can only be reached after reading an EnvSig or Timer expiry.

```
Host -------- <Play mode off> --------------> Target
```

Note:

After the play mode has been switched off, it is not allowed to switch the record mode or the play mode on again.

General Restrictions on Record and Play

Of course, there are restrictions in the use of the Cmicro Recorder, especially for the play mode. However, there may be situations where the Cmicro Recorder does not help in finding the source of the problem. The user must decide when the Cmicro Recorder can be properly utilized.

Firstly, the user has to ensure the same configurations are used for hardware and software, in order to allow comparison between the results of a recorded session with the ones of the replayed session. This is only a general recommendation. It is possible that he even would like to compare two different hardware configurations, and the Cmicro Recorder should help him finding the differences between these. As a more general recommendation we would say: Compare only those configurations that are definitely comparable.

Secondly, starting and ending the record and the play mode must be as follows. Host and target are to be synchronized.

Restrictions when Starting Record

The target may not transfer information to the host until the host has become active.

Therefore, the SDL Target Tester's host site should be started first and the file to write in should be opened.

If the target is started (e.g. reset-button), it has to wait on sdtmt before transmitting information (see "XMK_WAIT_ON_HOST" on page 3412).

Restrictions when Ending Record

Ending the record session is to be done from the GUI (sdtmtui) by closing the <outfile>. The user can enter any exit commands at any time. He should pay attention to the exact time when he exits. If exit is performed in the middle of a running SDL transition in the target, then the traces following will not be stored! The last information may then be unpredictable.

Restrictions when Starting Replay

The target may not transfer information to the host, until the host has become active. The host may not transfer information to the target, before the target has become active.

Therefore the SDL Target Tester's host site should be started first and the file containing the recorded session should be opened.

If the target is started (e.g. reset-button) it has to wait for the host before transmitting information.

Restrictions when Terminating Replay

In principle, the play mode reaches the end when the last stimuli stored in the recorded file was read and sent to the target. Actually, the play mode should end when all the transitions which follow that stimuli have ended. This is a discrepancy which is solved as follows:

When the target has received the last stimuli from the recorded session, it enters the "recorder off" mode. The problem with this is, that timers are simulated during the play mode but are no longer simulated if changing to "recorder off" mode. Timers will expire as they have been set and it is the real-time that will be used to allow them expire instead of the simulated timer expiry in the Cmicro Recorder.

Real-time Play Mode

The host tries to replay the SDL environment events at the same absolute time values as they occurred during the record session, if so required by the user (real-time-play is switched on).

Internally, a time stamp is stored for each environment signal, which is compared within the target with the current value of SDL now. SDL now and the time stamps in the signals are calculated with the value 0 from system reset on. The target generates the SDL now values in any case.

Of course, the processing power of host and target together must be large enough to handle this. As a rule of thumb, the recorded environment signals may not be sent to the SDL system more often than one every few hundred microseconds. But that - of course - also depends on the communications link in use.

Restrictions when Using Preemption

There are some restrictions when the Cmicro Recorder is used in combination with preemption. The problem is that in a real-time execution, signals coming from the environment (this may be a hardware interface) can cause a preemption of a running SDL process by a new process with a higher priority. The preemption may occur at any time, which means at any machine instruction (see notes below). On the other hand, only SDL symbols are registered by the Cmicro Recorder.

This means that the play mode may produce a different system behavior other than the recorded one.

The only alternative to implement an exact reproduction of the recorded session would be the use of special debugger hardware (not a general solution).

Some Additional Comments

Real-time behavior cannot be expected in play mode. When considering the other direction, from SDL to the environment, it is not absolutely necessary to cut this connection off (C function `xOutEnv`). It may be a good idea to let this connection be as it is in record mode. For example, if there is a display driver in the environment, which is controlled by SDL, it would then be possible to see the reactions on the display in the same way as they have been observed in record mode.

Utility Functions

File Structure

Description of Files

The modules and functions described in this subsection are used to handle first in-first out buffering, ring buffering, message coding and other possibilities.

ml_buf.c

This module contains functions to handle first-in first-out buffers and ring buffers.

It is mainly used internally by the SDL Target Tester to write and read trace information into/from a first-in first-out buffer within the RAM.

These functions are also free for users.

Three functions are implemented representing the low level functions. Each entry has to be of the same size, because the handling is in principle similar to an array. The type of each entry is not relevant. In addition, the caller of the functions contained in this module defines the memory for the trace buffer as well as a control structure representing all the internal data and variables necessary to administrate the trace buffer.

As a result, users are free to have more than 1 instance of this buffer type.

ml_buf.h

This is the header file containing the external definitions of `ml_buf.c`.

Trouble-Shooting

What to Do if the SDL Editor Trace Does Not Work?

The SDL Target Tester gets the SDL Editor references from the generated file `<systemname>.sym` in the known form `#SDTREF( ... )`.

As it is possible to generate an SDL system and to test it on another platform it is possible that the paths contained in each `#SDTREF()` are not up to date if using the SDL Target Tester.

All the paths in `<systemname>.sym` must simply be updated.

What to Do if There Is a Warning of the Information/Message Decoder?

If the message decoder detects an error there will be a warning like the one shown in Example 599.

Example 599: warning of the message decoder ––––––––––––––––––––––––––––––––

```
**WARNING: There is an inconsitency in the information decoder.
This message will be suppressed form now on.
Message Class: 241,d
Message Tag  : 5,d
Number of data bytes received=6
Number od data bytes decoded =5

  ***Received buffer (32 octets):
  ***f1 05 aa 06 00 00 00 00 - 01 04 ab ee ee ee ee ee
  ***ee ee ee ee ee ee ee ee - ee ee ee ee ee ee ee ee
```

––

This warning must be interpreted as follows:

- In the Cmicro Package there are the following message classes:
  - Cmicro Tester commands = 0xf0 = 240
  - Cmicro Tester trace = 0xf1 = 241 (see Example 599)
  - Cmicro Recorder = 0xf2 = 242

- The message tag can be found in the file “tmcod.h” on page 3583

  In Example 599 the message tag 5 (decimal) is the message `CMD_TSTATE`. Furthermore there is a structure called `xmk_T_CMD_TSTATE`. This structure looks like

```
typedef struct
{
  xPID        process;
  xmk_T_STATE state;
}
```

- The shown buffer from above can be interpreted as:
  f1 -> Cmicro Tester trace
  05 -> `CMD_TSTATE`
  aa -> sync byte
  06 -> message data length 6 octets.

- The message data contains the values `xPID` and `xmk_T_STATE`. In the Example 599 the `xPID` value is an `unsigned int` and the `xmk_T_STATE` is an `unsigned char`.

Note:

The kind of C basic type the `xPID` value is represented by, depends on the SDL system in use (single instance processes only or multiple instance processes). Please view the typedef of `xPID` in the file `ml_typ.h`.

The size of the C basic type depends on the microcontroller and compiler that is used.

By having a look to the C struct (`xmk_T_CMD_TSTATE`) which is represented by the 6 data octets received a value for `xPID` is found like `0x00000000` as an integer is given with the length 4 octets in this example.

Note:

The kind of C basic type the `xmk_T_STATE` value is represented by, depends on the SDL system in use, please view the flag “XMK_USE_HUGE_TRANSITIONTABLES” on page 3397.

The size of the C basic type depends on the microcontroller and compiler in use.

Having a look to the fifth octet of the message data the value for `xmk_T_STATE` is `0x01`, as the length of character is 1 and the alignment of character is 8. Please view “Preparing the Host to Target Communication” on page 3520.

After all in this example there are 5 bytes (octets) decoded (4+1) and 6 received which leads to the inconsistency in the information decoder.

The user is asked to correct the settings in the file `sdtmt.opt` to support the SDL Target Tester’s host site with the correct information about the target’s memory layout.

Caution!

The Example 599 is an exception concerning the inconsistency in the information decoder.

The compiler used for this example only knows C structures with a size dividable by two which leads to a message data length of 6.

The value 0x04 in the example only represents a filler octet which does not need to be noticed.

Chapter 68 The SDL Target Tester