MLIB/MTRACE

MATLAB-dSPACE Interface Libraries

Release 5.3 – March 2007

dSPACE

How to Contact dSPACE

Mail:	dSPACE GmbH Technologiepark 25 33100 Paderborn Germany
Tel.:	+49 52 51 16 38-0
Fax:	+49 52 51 6 65 29
E-mail:	info@dspace.de
Web:	http://www.dspace.com
Technical Support:	support@dspace.de +49 52 51 16 38-941 http://www.dspace.com/goto?support

How to Contact dSPACE Support

dSPACE recommends that you use dSPACE Support Wizard to contact dSPACE support. It is available:

- On your dSPACE DVD at `\Diag\Tools\dSPACESupportWizard.exe`
- Via Start – Programs – dSPACE Tools (after installation of the dSPACE software)
- At http://www.dspace.com/goto?supportwizard
  You can always find the latest version of dSPACE Support Wizard here.

Software Updates and Patches

dSPACE strongly recommends that you download and install the most recent patches for your current dSPACE installation. Visit http://www.dspace.com/goto?support for software updates and patches.

Important Notice

Contents

About This Document

This document contains detailed information on the MATLAB-dSPACE Interface Libraries.

Where to go from here

Information in this section

Document Symbols and Conventions

Symbols

The following symbols are used in this document:

⚠	Indicates a general hazard that may cause personal injury of any kind if you do not avoid it by following the instructions given.
⚡	Indicates the danger of electric shock which may cause death or serious injury if you do not avoid it by following the instructions given.
⚠	Indicates a hazard that may cause material damage if you do not avoid it by following the instructions given.
⯃	Indicates important information that you must note to avoid malfunctions.

Indicates tips containing useful information to make your work easier.

Naming conventions

The following abbreviations and formats are used in this document:

%name% Names enclosed in percent signs refer to environment variables for file and path names, for example, `%DSPACE_ROOT%` specifies the location of your dSPACE installation in the file system.

< > Angle brackets contain wildcard characters or placeholders for variable file and path names, etc.

📖 Precedes the document title in a link that refers to another document.

❓ Indicates that a link refers to another document, which is available in dSPACE HelpDesk.

Accessing Online Help and PDF Files

Objective

After you install your dSPACE software, the documentation for the installed products is available as online help and Adobe® PDF files.

Online help

You can access the online help – dSPACE HelpDesk – as follows:

Windows Start menu Click Start – Programs – dSPACE Tools – dSPACE HelpDesk.

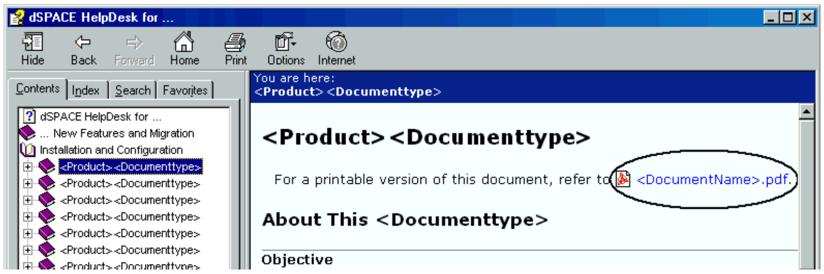
Context-sensitive Press the **F1** key or click the **Help** button in the dSPACE software.

Local installation on your host PC Double-click the `dSPACEHelpDesk.chm` file in `%DSPACE_ROOT%\Doc\Online`.

PDF Files

You can access the PDF files as follows:

dSPACE HelpDesk Click the PDF link at the beginning of a document:



<Product> <Documenttype>

For a printable version of this document, refer to <DocumentName>.pdf.

About This <Documenttype>

Objective

Local installation on your host PC Double-click the PDF file in `%DSPACE_ROOT%\Doc\Print`.

Related Documents

Below is a list of documents that you are recommended to also read when working with MLIB/MTRACE.

Information in other documents

Interface libraries

MLIB/MTRACE MATLAB-dSPACE Interface Libraries 1
Contains detailed reference information and instructions on the experiment features of MLIB/MTRACE.

CLIB C Interface Library
Contains detailed reference information on the C Interface Library CLIB, which contains C functions to access the dSPACE processor and controller boards from the host PC.

ControlDesk Standard

ControlDesk Experiment Guide
Introduces you to the experiment features provided by ControlDesk Standard.

ControlDesk Automation Guide
Shows you how to automate the features provided by ControlDesk Standard.

ControlDesk Reference ***ControlDesk Instrument Reference*** ***ControlDesk Automation Reference***
AutomationDesk ***AutomationDesk Guide*** Introduces you to the automation features provided by AutomationDesk. It provides detailed information on the GUI components, basic concepts, and instructions using AutomationDesk and AutomationDesk Automation Server. ***AutomationDesk Tutorial*** Has several lessons that guide you through using AutomationDesk. It shows you concrete procedures by means of a demo project. ***AutomationDesk Reference*** Provides detailed information on the menus, context menus, and dialogs contained in AutomationDesk. ***AutomationDesk Library Reference*** Provides detailed information on the libraries supported by AutomationDesk.
RTI and RTI-MP, RTLib ***Implementation*** Provides you with information on the features, the hardware and the software support for your specific system.

Introduction to MLIB/MTRACE

Basics of MLIB/MTRACE

The MATLAB-dSPACE Interface Libraries give you access to dSPACE real-time processor hardware from the MATLAB workspace.

MLIB/MTRACE functions can be called from the MATLAB Command Window or from MATLAB M files. Thus the powerful numerical tools running under MATLAB can be used in combination with MLIB/MTRACE for:

- Analyzing real-time data
- Test automation
- Optimizing control algorithms

MLIB/MTRACE provides basic functions for reading and writing data to dSPACE processor boards and other functions for generating interrupts, setting the processor state, and getting processor status information.

MLIB/MTRACE provides real-time data capture capabilities, including the following features:

- Free-running or level-triggered mode
- Pre- and post-trigger
- Selection between single_shot and continuous mode
- Simultaneous start of multiple data captures
- Distinction between double, float and integer (signed and unsigned) variables
- Adjustable trace capture sampling rate (downsampling)
- Direct data transfer to the MATLAB workspace

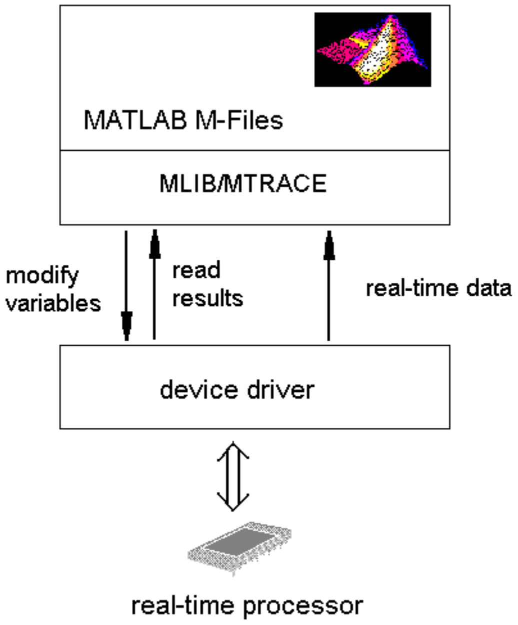
- Data storage on the hard disk drive in continuous mode (optional)
- Specification of data capture parameters by property/values pairs.

MLIB/MTRACE functions are used to modify parameters online and to send sequences of test data to real-time applications. To get access to any variable on a real-time processor with MLIB/ MTRACE, you do not need to know its address because there are several functions that return the required addresses when the symbolic name of a variable is specified.

MLIB/MTRACE ideally complements the software environment, which consists of MATLAB, Simulink, Real-Time Workshop, and the dSPACE tools RTI and ControlDesk.

MLIB/MTRACE is not only used for programs generated by RTI, but also for any other application program written in C.

The following illustration shows how a real-time processor is accessed from MATLAB M files via MLIB/MTRACE.



Installation

To get help on the installation process of MLIB/MTRACE, refer to the list below:

- To check the required dSPACE boards and the versions currently available, refer to *System Requirements* on page 13
- For information about the created directories, refer to *Installation Process* on page 14
- For M files information about the copied demo files, refer to *Demo Files* on page 15

System Requirements

The functions of the MATLAB-dSPACE Interface and Trace Libraries are available for the MATLAB versions that are supported by the current dSPACE Release. MLIB/MTRACE runs on IBM-PC/AT or compatible machines under Windows NT, Windows 2000 and Windows XP. It currently supports the following dSPACE processor boards:

- DS1103
- DS1104
- DS1401
- DS1005
- DS1005 multiprocessor systems
- DS1006
- DS1006 multiprocessor systems

MLIB/MTRACE requires the following dSPACE software products:

Board	RTLIB
DS1103	RTLIB1103
DS1104	RTLIB1104
DS1401	RTLIB1401
DS1005	RTLIB1005
DS1006	RTLIB1006

- When using RTI or RTI-MP for multiprocessor systems, you need the RTI version from the related dSPACE release.
- When MLIB/MTRACE is used to access a dSPACE processor board via network, make sure that the TCP/IP Network software is installed (delivered with Windows NT, Windows 2000 and Windows XP).
- It is recommended to install all the software related to MLIB/MTRACE from the same distribution CD/DVD.

Related topics

Basics

• *Installation* on page 13

Installation Process

The installation program performs all necessary steps automatically. During installation you will only be asked for the MATLAB root folder.

When MLIB/MTRACE is installed, files are copied to `%DSPACE_ROOT%\Matlab\Mlib`.

In addition, the file `%DSPACE_ROOT%\Matlab\Local\Dspacerc.m` is copied to `\Matlab\Toolbox\Local`.

A call to `dspacerc.m` is then appended to the MATLAB startup file `Matlabrc.m`. As a result, the dSPACE configuration file `dspace_setup.m` residing in `%DSPACE_ROOT%\Matlab\Local` is executed each time MATLAB is started.

All necessary configurations for dSPACE products related to MATLAB are set automatically by `dspace_setup.m`.

Related topics

Basics

- *Installation* on page 13

Demo Files

The files necessary to run the MLIB/MTRACE demos supplied by dSPACE are installed in the following folders:

- `%DSPACE_ROOT%\demos\dsXXXX\mlib`,
- `%DSPACE_ROOT%\demos\dsXXXXMP\dualXXXX\mlib`.

`dsXXXX` denotes the respective processor board name, for example, DS1103, DS1104, DS1401, DS1005 or DS1006.

Related topics

Basics

- *Installation* on page 13

Getting Started

A good way to become familiar with MLIB/MTRACE is to take a look at the demo examples installed in the following folders

- `%DSPACE_ROOT%\demos\dsXXXX\mlib` for MLIB/MTRACE
- `%DSPACE_ROOT%\demos\dsXXXXMP\dualXXXX\mlib` for MLIB/MTRACE-MP.

The sample M files can also be used as templates for new applications.

- Before using MLIB/MTRACE functions, you must select a real-time processor board for use with MLIB/MTRACE. The processor board must be already registered in ControlDesk's Platform Manager.

Board info

To obtain a list of the boards currently registered in ControlDesk's Platform Manager, include the command **`boards_info = mlib('GetBoardInfo')`** in the MATLAB window. The boards_info will then contain basic information about registered boards. If boards_info is empty, no boards are registered.

mlibini.m

It is recommended to use the MATLAB M file `mlibini.m` supplied with MLIB/MTRACE to select the first processor board registered at ControlDesk's Platform Manager. For most MLIB/MTRACE applications, it is sufficient to include the line

```
% select the first processor board registered
% at ControlDesk's Platform Manager for use with
% MLIB/MTRACE
mlibini
```

at the beginning of M files that call MLIB/MTRACE functions.

Connection via bus or network

MLIB/MTRACE can access locally installed boards via a bus connection, and other boards via a network connection. The choice of bus or network connection is automatic. The first time an MLIB/MTRACE function is called, the last connection that was set by Control Desk's Platform Manager is used.

MLIB/MTRACE DLL file

MLIB/MTRACE is implemented as a MATLAB MEX DLL file, which is loaded into the PC memory the first time it is called during a MATLAB session. MLIB/MTRACE remains loaded until a **`clear mex`**, **`clear functions`**, **`clear dsmlib`**, or **`clear all`** command is invoked, or until MATLAB is closed. If you clear MLIB/MTRACE, you must call SelectBoard before proceeding with other MLIB/MTRACE functions.

The following topics contain further information on:

- *Preparing Real-Time Applications* on page 18
- *Typical Calling Sequences of MLIB/MTRACE Commands* on page 19

Preparing Real-Time Applications

MLIB/MTRACE needs special services to be executed in the real-time application, to perform Read/Write operations and carry out real-time data acquisition:

- host_service(hostsvc_no, ts): where the hostsvc_no can be an integer number between 1 and 4
- master_cmd_server()

host_service (hostsvc_no, ts)

This service, called the data capture service in this manual, is responsible for real-time data acquisition. It collects one set of data per call and writes it to a buffer reserved in the real-time processor memory, called the trace buffer below. Thus, the rate at which the data is sampled depends on the frequency at which this service is executed.

For applications generated by Real-Time Interface, data capture service number 1 is called in the task with the fastest timer-based sample rate, by default. To add an additional service to the real-time application, you must use the Data Capture block from the RTI library. For handcoded applications, it is your responsibility to place the call to the data capture service, host_service(hostsvc_no, ts), in the desired task.

The second parameter, ts, is a pointer to the time stamp structure that represents the absolute time when the set of trace data was collected.

master_cmd_server()

This service reads and evaluates commands sent by MLIB/MTRACE to the real-time program. It has to be called in the background loop.

You can only read and write data from the local memory of the DS1103 and the DS1401 boards, local variables residing on the DS1104 board, and the cached memory of DS1005 and DS1006, if this service is executed.

RTI inserts this service automatically.

Example

```
while(1)
{
  ...
  master_cmd_server();
  ...
}
```

Related topics

Basics

- *Getting Started* on page 17

Typical Calling Sequences of MLIB/MTRACE Commands

MLIB/MTRACE functions always relate to the most recently selected board. Repeated selection of the same board has no effect.

For example, if a DS1005 board is used, the typical sequence of MLIB/MTRACE commands is as follows:

Example

This example shows how to modify parameters online:

```
% select DS1005 for use with MLIB/MTRACE
mlib('SelectBoard', 'DS1005');
% obtain the descriptors of selected variables
var_names = {'x_par'; 'y_par'};
xy_desc = mlib('GetMapVar', var_names);
% read data from DS1005
[x_val, y_val] = mlib('Read', xy_desc);
% display the results
fprintf("x_par = %f\ny_par = %f\n", x_val, y_val);
```

```
% write new value to DS1005
data = {0.125; 4.5};
mlib('Write', xy_desc, 'Data', data);
```

This example shows the online real-time data acquisition:

```
% connect to processor board DS1005
mlib('SelectBoard', 'DS1005');
% find address of variable in processor memory
x_var = mlib('GetTrcVar', 'User Variables/x');
% configure MLIB for data capture:
% select variable for trace capture,
% configure the time frame for data acquisition
mlib('Set', ...
   'TraceVars', x_var, ...
   'Delay', 0, ...
   'NumSamples', 500);
% start data acquisition;
% default service number for DS1005 is 1
mlib('StartCapture');
% wait until capture is finished
while mlib('CaptureState')~=0, end;
% transfer real-time data to MATLAB workspace
x_data = mlib('FetchData');
% plot results
plot(x_data');
```

For further information, refer to the demo applications in `%DSPACE_ROOT%/demos/dsxxxx/mlib`.

For a detailed description of the MLIB/MTRACE commands, refer to *MLIB/MTRACE Functions* on page *29*.

Related topics

Basics

- *Getting Started* on page 17

Using MLIB/MTRACE for Multiprocessor Systems

MLIB/MTRACE supports access to variables of real-time applications running on dSPACE single-processor and multiprocessor systems.

Typical dSPACE multiprocessor systems are:

- Two or more DS1005 boards linked by the GigaLink connection
- Two or more DS1006 boards linked by the GigaLink connection

For basic information, refer to *Basic Principles of Using Multiprocessor Systems* on page 22.

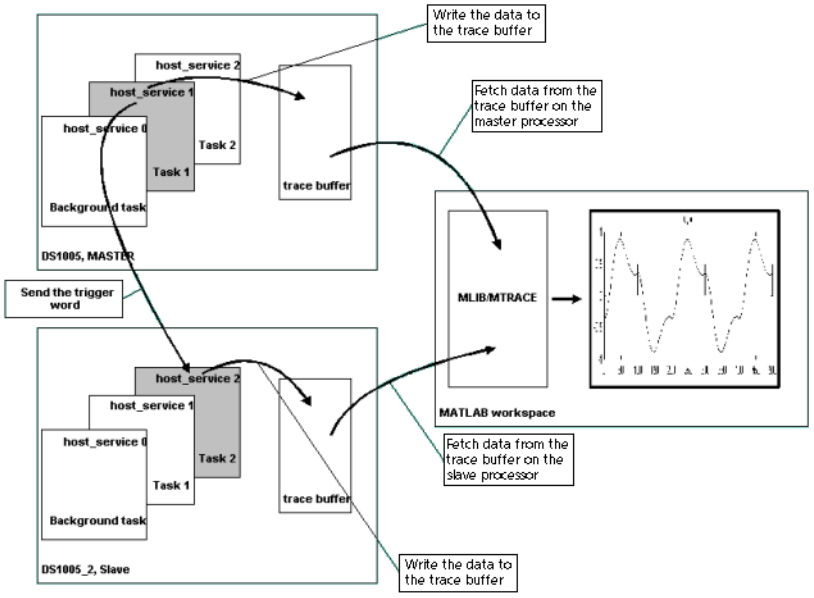
For information on data acquisition on multiprocessor systems, refer to *Data Capture With Different Sampling Periods or Asynchronous* on page 25.

For further information about dSPACE multiprocessor systems, refer to the RTI and RTI-MP Implementation Guide, DS1005 RTLib Reference and DS1006 RTLib Reference.

Basic Principles of Using Multiprocessor Systems

Each member board of a multiprocessor system based on DS1005 or DS1006 is equipped with a host interface and can be accessed directly by MLIB/MTRACE in single-processor mode. However, a single MLIB/MTRACE instance can operate only with one currently selected board. To access all boards simultaneously in single-processor mode, additional instances of MLIB/MTRACE must be created.

The other way is to use MLIB/MTRACE in multiprocessor mode. After the Multiprocessing function is invoked, all global variables residing on any processor that is part of the selected multiprocessor system are accessible with MLIB/MTRACE. Read/write access is similar to read/write access in single-processor mode. Appreciable differences occur in the data capturing (tracing) mechanism. A distributed tracing method is used for multiprocessor systems. The graphic below explains this idea:



In this method, variables are traced locally on each processor board. MLIB/MTRACE accesses them separately and tries to give the data a common time frame. This strategy has the following requirements:

- Information on the time stamp when the data set was collected must be available.

- Data acquisition on the single processors must be started synchronously.

The first condition is fulfilled if the time stamps at which the data has been collected are also traced. The second condition is met by using a trigger mechanism: One of the processors is used to synchronize the start of data acquisition on all the other processors. This processor is called the synchronizing processor below. If triggered mode has been selected, the synchronizing processor is the processor on which the trigger variable resides. For non triggered mode, any processor can be the synchronizing processor. In this case MLIB/MTRACE chooses the synchronizing processor according to certain rules, which are not of importance for an understanding of the distributed tracing method. The synchronized processors always run in triggered mode.

The trigger signal which starts data acquisition on these processors is a 32-bit word sent by the synchronizing processor.

Assuming triggered mode has been selected, as soon as the trigger event is recognized by the synchronizing processor, a trigger word is changed according to predefined rules and sent to the synchronized processors. The synchronized processors react to the change in the trigger word and start data acquisition.

Data Capture in Continuous Mode

If the data is captured in continuous mode, the start of data acquisition on the single processors is not synchronized. In this case MLIB/MTRACE returns the data to the MATLAB workspace in the form of a vector of structures: one structure for each processor. Since the time stamps are traced throughout the capture process, it is possible to give the data a common time frame.

Service Group

As shown in the example, data acquisition on the master processor is carried out by host service number 1. On the slave processor, it is carried out by host service number 2. These two services form the **service group**. It is possible to build several service groups for the multiprocessor system, for example: CPU master, service number 1; CPU slave, service number 1; or CPU master, service number 2; CPU slave, service number 2, etc.

To carry out data acquisition in single processor mode, you must specify a service number; to carry out data acquisition in MP1005 or MP1006 mode, you must specify a service group.

The service group is identified by its name. To specify the service group as above, the following MLIB/MTRACE functions must be called (assuming Task1 on the master is called with 0.00012 sampling period, and Task2 on the slave is called with a sample period of 0.00036)

```
mlib('Set', 'ServiceGroup','Master1_Slave2_Group',....
   'CPU','Master',...
   'Service', 1,....
   'StepSize', 0.00012);
mlib('Set','ServiceGroup','Master1_Slave2_Group',....
   'CPU','Slave',....
   'Service',2,....
   'StepSize', 0.00036);
```

Each time the StartCapture function is called with this name as an input parameter: `mlib('StartCapture','Master1_Slave2_Group);`, the data capture is carried out by service 1 on the master processor and by service 2 on the slave processor.

The TRC files generated by RTI-MP contain entries describing the available host services in the following form:

```
sampling_period[1]
{
   value:      0.0
   alias:      "AsynchronousTask"
}
```

The index of the sampling_period vector denotes the host_service number, the service name is specified under alias and the step size (0.0 = asynchronous task) under value. If TRC files exist, MLIB/MTRACE combines services that are on different processors but have the same name to form one service group. The name of the service group corresponds to the name of the services. You do not have to specify the service group as shown above. However, you can use this service group, for example, with StartCapture function.

Related topics

Basics

- *Data Capture With Different Sampling Periods or Asynchronous* on page 25

Data Capture With Different Sampling Periods or Asynchronous

If the services carrying out data acquisition on the individual member processors are called with the same sampling period, in most cases MLIB/MTRACE is able to find a common time axis for data collected on different CPUs. Therefore it makes no difference whether the trace interval has been specified by the number of samples to be collected or by the duration of data acquisition. Since the sampling period is the same for all processors, the N samples on processors P1, P2, P3 etc., corresponds to (N * sampling period) T seconds on processor P1, P2, P3 etc. and T seconds on processor P1 correspond to (T/sampling_period) N samples on processor P2, P3.

However, if the services carrying out data acquisition on the individual member processors are called with different sampling periods or asynchronously, MLIB/MTRACE returns different results depending on how the trace interval is defined. The examples below explain the differences.

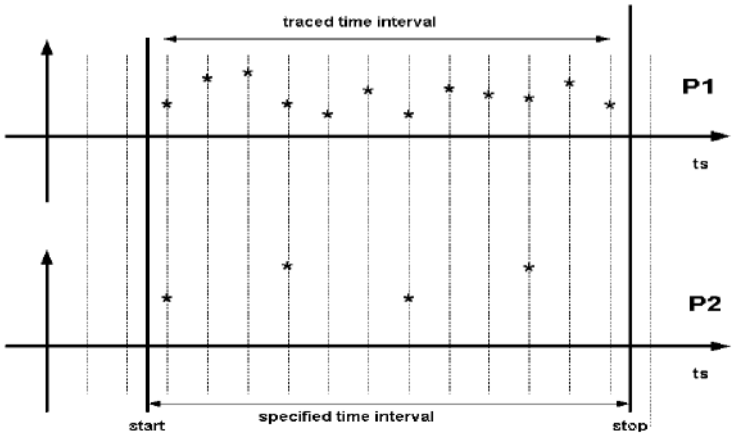
The following assumptions apply to both examples:

- Sampling period on processor P1 = tp
- Sampling period on processor P2 = 3 * tp
- Synchronizing processor: P1

1. The trace interval is specified by the duration of the data acquisition:

   ```
   mlib('Set',...
        'Start', t0,....
        'Stop',tN,...
   );
   ```

The graph below shows the result of the data acquisition:



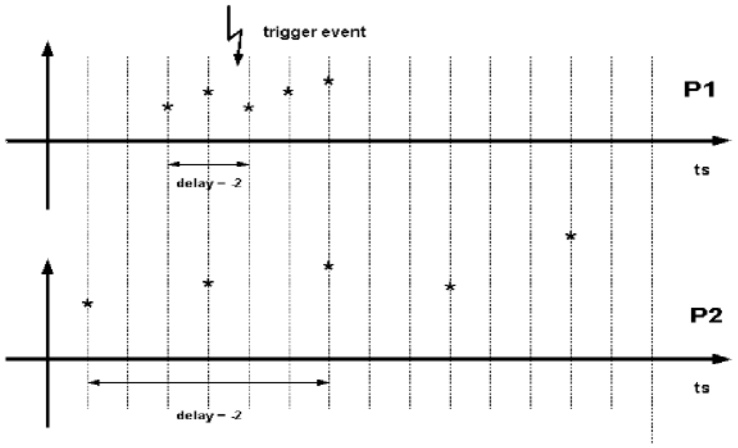
Only points which lie exactly on or after the lower time limit, and points which lie before or exactly on the upper time limit, are collected.

The number of samples collected with processor P1 differs from the number of samples collected with processor P2.

2. The trace interval is specified by the number of samples to be collected:

```
mlib('Set',...
     'Delay', -2,....
     'NumSamples',5,...
);
```

The graph below shows the result of data acquisition:



Since the samples are collected with a different sampling period on each processor, the data is spread over different time intervals.

Related topics

Basics

- *Basic Principles of Using Multiprocessor Systems* on page 22

MLIB/MTRACE Functions

This topic presents a brief overview of the MLIB/MTRACE functions. All functions are listed alphabetically and described in detail with relevant examples:

- *General Information on the Function Reference* on page 30
- *MLIB/MTRACE Function Reference* on page 31

General Information on the Function Reference

Naming Conventions

In the chapter *MLIB/MTRACE Function Reference* on page 31, the following conventions for parameter names are used:

var Variable descriptor – a structure containing information about a variable as returned by the following address evaluation functions:

Function	Description
var.name	Name of the variable
var.cpu	Name of the CPU on which the variable resides
var.type	Data type
var.addr	Variable base address
var.offset	Offset of the first element from the base address
var.length	Number of elements

var_vec A vector of structures described above

ival Matrix of integer values in the MATLAB workspace

uival Matrix of unsigned integer values in the MATLAB workspace

fval Matrix of single-precision floating-point values in the MATLAB workspace

dval Matrix of double-precision floating-point values in the MATLAB workspace

data A cell array in the MATLAB workspace, where each cell can be a scalar, a vector, or a matrix of any data type, supported by a selected processor board

MLIB/MTRACE Function Reference

The following table gives an overview of the implemented MLIB/MTRACE functions.

Purpose	Refer to
Administrative Functions	
To get information about a real-time application	*GetApplInfo* on page 40
To get information about boards currently registered at ControlDesk's Platform Manager	*GetBoardInfo* on page 42
To select a real-time processor board for use with MLIB/MTRACE	*SelectBoard* on page 60
To check if a real-time application is running	*IsApplRunning* on page 50
To set a lock on the real-time program	*LockProgram* on page 54
To select single-processor or multiprocessor mode	*Multiprocessing* on page 54
To release a lock on the real-time program	*UnlockProgram* on page 71
To load a variable description file	*LoadVarFile* on page 53
Address Evaluation Functions	
To obtain the descriptor for a global variable from the linker map file	*GetMapVar* on page 43
To obtain the descriptor for a global variable specified within the TRC file	*GetTrcVar* on page 45
To search the linker map file for variables whose names match the specified regular expression	*SearchMap* on page 58
To search the TRC file for variables whose names match the specified regular expression	*SearchTrc* on page 59
To set the properties stored in a variable descriptor	*SetVarProperty* on page 65
Read/Write Functions	
To read data from a real-time processor board	*Read* on page 55
To read data directly from a real-time processor board	*ReadDirect* on page 57
To write date to a real-time processor board	*Write* on page 71
To write data directly to a real-time processor board	*WriteDirect* on page 73
Interrupt Functions	
To generate an interrupt on a real-time processor	*Intrpt* on page 49
Data Acquisition Functions	
To start a data capture on the processor board	*StartCapture* on page 67
To stop a data capture on the processor board	*StopCapture* on page 68
To get trigger status information	*TriggerState* on page 70
To get capture status information	*CaptureState* on page 32
To transfer data from the processor board memory to the MATLAB workspace	*FetchData* on page 34
To set data acquisition options	*Set* on page 60
To get data acquisition options	*Get* on page 37
Functions Supporting MLIB/MTRACE–Generated MAT Files	
To load selected data from a MAT file into the MATLAB workspace	*LoadFile* on page 51

Processing mode

If the syntax or description depends on the processing mode, the mode is indicated:

- *SP = single-processor mode*
- *MP = multiprocessor mode with DS1005-based or DS1006-based multiprocessor systems*

Information on how to select a processing mode is given in the description of the Multiprocessing function.

- To get online help for MLIB/MTRACE functions, type **`help mlib`** or **`help <function_name>`** in the MATLAB Command Window.
- To get HTML online help, open the dSPACE HelpDesk or type **`mlib_help`** in the MATLAB Command Window.

If you are using MLIB/MTRACE with a multiprocessor system based on DS1005 or DS1006, we strongly recommend that you familiarize yourself with the topic *Using MLIB/MTRACE for Multiprocessor Systems* on page 21 before reading the reference part for the first time.

CaptureState

Purpose

To get capture status information.

Syntax SP

```
state = mlib('CaptureState')
state = mlib('CaptureState', services)
```

Syntax MP

```
state = mlib('CaptureState')
state = mlib('CaptureState', serviceGroups)
```

Description

When used without an input parameter, this function returns the current state of the data capture process performed by the default service for SP or the service group for MP1005 or MP1006.

In single-processor mode, the default service is service number 1.

In multiprocessor mode, the defaults are as follows:

- If the TRC file exists:
  - The Service group collecting services with the same name as the service called with the fastest rate on the master processor is used as the default.
  - If all services are called asynchronously, the service group collecting services with the same name as the service first found on the master processor is used as the default.
- If the TRC file does not exist:
  - The service group named ServiceName1 is used as the default. This collects all services with the number 1 on member processors.

Otherwise, you must specify the service/service group `services/serviceGroups` in the parameter.

The return values indicate the current state of the data capture:

Value	Description
2	The buffer is empty.
1	The buffer is neither empty nor full (data acquisition is in progress).
0	The buffer is full (data acquisition is finished).

If multiple data captures performed by different data capture services/services groups were started simultaneously, the states of all of them by calling the CaptureState function with the additional input parameter `services/serviceGroups`. This must be a vector containing the numbers of services or a cell array containing the names of service groups carrying out data acquisition. In this case, `state` is a vector whose elements indicate the state of the corresponding capture process.

Result

The state of the capture process is returned.

Example

Example 1:

```
% wait for a data acquisition to finish
while mlib('CaptureState')~=0, end
disp('Capture finished.');
```

Example 2:

```
% wait for a data acquisition performed
% by services 1 and 2 to be finished
while any(mlib ('CaptureState', [1 2])) ~= 0, end
disp('Both captures finished.');
```

Example 3:

```
% wait for a data acquisition performed
% by the service groups HostService1 and HostService2
while any(mlib ('CaptureState', ...
   {'HostService1'; 'HostService2'})) ~= 0, end
disp('Both captures finished.');
```

Related topics

References

- *StartCapture* on page 67
- *TriggerState* on page 70

FetchData

Purpose

To fetch data from the real-time processor buffer and transfer it to the MATLAB workspace.

Syntax

```
traced_data = mlib('FetchData')
traced_data = mlib('FetchData', [count ,], ...
   property_name, property_value, ...
```

Description

Depending on the settings made in the MLIB/MTRACE function Set, the FetchData function returns the traced data in one of the following formats:

- Matrix
- Matrix with time
- Structure
- Structure with time

Matrix The number of rows corresponds to the number and length of variables that have been traced. Each row represents a time series of a variable (or of one vector element if the variable is a vector). Each column represents all variables at the same specific time. The rows in the output matrix are assigned to the traced variables in the order in which the variables were selected with the TraceVars property.

Matrix with time The format is the same as the matrix format, except that the first row represents time stamps at which the data was collected. The number of rows therefore corresponds to the number and length of variables that have been traced + 1.

Structure with time This structure has the following fields:

- .time
- .signals

The .time field contains a [N x 1] vector of the time stamps, where N is the number of collected samples (a sample contains a set of values of all the selected variables collected in one sampling step). The .signals field contains a vector of substructures, each of which corresponds to one trace variable.

Each substructure has two fields:

- .values
- .label

The .label field specifies the variable name, the .values field contains a [N x variable length] matrix, with values of the corresponding variable at time stamps specified in the .time field.

Structure The format is the same as the structure with time, except the .time field remains empty.

Structures for multiprocessor systems

If the structure format has been selected in the Set function, the FetchData function returns a vector of structures as described above, each of which corresponds to one processor with traced variables. To indicate which structure corresponds to which processor, an additional .name field is added to hold the processor name.

Note that each processor has its own time vector.

The data can be fetched from the processor memory even if the capturing process is still in progress. If the `count` parameter is omitted, all the data already captured and written to the trace buffer is returned. It is recommended to call the CaptureState function beforehand to ensure that the data capturing process is finished. The count parameter can be used to fetch the exact number of samples. In this case the number of columns in the output matrix and the number of rows in the .values matrices in the output structure equals the count. If there is not enough data to complete the output matrix/structure (for example, the FetchData function was already called several times and the remaining number of samples to be read from the trace buffer is smaller than `count`), zeros are written instead.

By default, MLIB/MTRACE assumes that the data to be fetched was captured by default capture service (SP) or service group (MP1005 or MP1006).

If the data capture was started in the continuous mode of MLIB/MTRACE with the StreamToDisk option set to ON (see the Set function), each call of the FetchData function also streams the data into a MAT file on the hard disk.

The data is always stored in the MAT file (matrix/matrix with time), independently of the output format of the FetchData function. The structure and structure with time formats are not supported.

The possible properties and their descriptions are as follows:

SP

Service If a service other than the default is used, you must specify its number via this property. This property can be used only in single-processor mode.

MP

ServiceGroup If a service group other than the default is used, its

name must be explicitly specified by via this property.

This property is used only in multiprocessing mode with multiprocessor systems based on DS1005 or DS1006.

StartSample By default, MLIB/MTRACE reads the data from the trace buffer, beginning with the current MLIB/MTRACE read pointer. The MLIB/MTRACE read pointer is the position in the trace buffer up to which data was fetched into the MATLAB workspace. To change this order, the StartSample property.

Result

The data captured for each variable selected with the Set function is transferred into the MATLAB workspace.

Example

Example 1:

```
uy_var = {'User Variables/u'; 'User Variables/y'}
uy_addr = mlib('GetTrcVar',uy_var)
% capture 2000 samples of each variable by service 2
mlib('Set', 'Service', 2, ...
   'TraceVars', uy_addr, ...
   'NumSamples', 2000);
mlib('StartCapture', 2);
% wait for a data acquisition to finish
while mlib('CaptureState', 2) ~= 0, end
% fetch all data for variables u and y
matdata = mlib('FetchData' , 2);
plot(matdata');
```

Example 2:

```
uy_var = {'User Variables/u'; 'User Variables/y'}
uy_addr = mlib('GetTrcVar',uy_var)
% capture 2000 samples of each variable by
% service 1(default)
mlib('Set', 'TraceVars', uy_addr, ...
   'NumSamples', 2000);
mlib('StartCapture');
% fetch data while capture process is still running
while mlib('CaptureState') ~= 2
   data = mlib('FetchData', 200)
   % edit data etc.
   ...
end
```

Related topics

References
- *Set* on page 60

Get

Purpose

To get current data acquisition options.

Syntax SP

```
info = mlib('Get')
info = mlib('Get', property_name)
info = mlib('Get', 'Service', service_number)
info = mlib('Get', 'Service', service_number, ...
   property_name)
```

Syntax MP

```
info = mlib('Get')
info = mlib('Get', property_name)
info = mlib('Get','ServiceGroup', service_group_name)
info = mlib('Get','ServiceGroup', service_group_name, ...
   property_name)
```

Result

The current MLIB/MTRACE settings, such as the time frame, trigger options, and variables to be traced, are returned. By default, if no service number/service group name or property name is specified, the function returns information about all the settings of the data capture process performed by the default service (SP) or service group (MP1005 or MP1006).

Another service number/service group name can be selected with the `service_number/service_group_name` parameter.

If any property is explicitly specified by its name, the info structure contains this single field only.

Depending on the selected processing mode, the returned info structure can have the following fields.

SP Mode	MP Mode
.Service	.ServiceGroup
.Trigger	.Trigger
.TriggerVariable	.TriggerVariable
.TriggerLevel	.TriggerLevel
.TriggerEdge	.TriggerEdge
.StepSize	.Downsampling
.Downsampling	.TimeStamping
.TimeStamping	.UseAbsoluteTime
.UseAbsoluteTime	.TraceVars
.Start	.AcquisitionMode
.Stop	.StreamToDisk
.Delay	.Output format
.NumSamples	.AutoClearBuffer

SP Mode	MP Mode
.TraceVars	.InterpolateBoundaryPoints
.AcquisitionMode	.CPU
.StreamToDisk	.TimeOutValue
.FileName	
.OutputFormat	
.AutoClearBuffer	
.TimeOutValue	

.CPU The .CPU field is a vector of structures, where each structure contains information about settings specified for one CPU. Each structure has the following subfields:

- .name (name of the CPU)
- .service (number of the data acquisition service on the current CPU, which belongs to the specified service group)
- .stepSize (the sampling period of the task where the data acquisition service is called)
- .start (the start of the trace interval for which the data is to be collected, stated in the units of the step size)
- .stop (the end of the trace interval for which the data is to be collected, stated in the units of the step size)
- .numSamples (length of the trace interval specified in the number of samples)
- .delay (delay related to the trigger event specified in the number of samples)
- .fileName (name of the MAT file to store the data collected in StreamToDisk mode)

For further information on the fields and the returned info structure, refer to the Set function.

Example

```
% display the settings of the data capture process
% performed by service 2
mlib('Get', 'Service', 2)
...
% display the value of the NumSamples property of the data
% capture process performed by service 1(default)
mlib('Get', 'NumSamples')
```

Related topics

References

- *Set* on page 60

GetApplInfo

Purpose

To get information about the real-time application.

Syntax

```
info = mlib('GetApplInfo')
info = mlib('GetApplInfo', 'property_name', property_value...)
```

Result

By default, if SP mode is selected, GetApplInfo returns information about the real-time application currently running on the selected processor board.

Description

The returned information is stored as a structure containing the following fields:

Field	Description
.name	Name of the real-time program
.date	Creation date of the real-time program
.board	Name of the board on which the program is running
.type	Type of the board on which the program is running

If the MP mode is selected, the `info` parameter is an M-by-1 vector of structures, where M equals the number of CPUs in the currently loaded MP application. Each structure contains information about the program running on the corresponding CPU as described above plus the following fields:

Field	Description
.master_cpu	Name of the master CPU
.master_board	Name of the board on which the master CPU resides
.reachable	An integer value that indicates if the global variables residing on the CPU are accessible via MLIB/MTRACE or not 0: the variables are not accessible 1: the variables are accessible

The permissible property/value pairs are listed below:

'SDFFile' Name of the SDF file created by RTI-MP. If this property is defined (SP or MP mode), the GetApplInfo function returns information about the MP application specified in the SDF file, independently of the application currently loaded on the processor board(s).

'CPU' Name of the CPU for which the info structure is to be returned. In this case, info is a simple structure.

Example This example refers to single-processor systems (SP)

```
% which application is running on the selected board?
appl = mlib('GetApplInfo');
disp(['The current real-time application is' appl.name]);
```

This example refers to multiprocessor systems (MP)

```
% Get the information which program is running on the
% CPU "MASTER"
appl = mlib('GetApplInfo', 'CPU', 'Master');
fprintf('The program running on the CPU MASTER is: %s', ...
   appl.name);
```

Related topics

References

- *IsApplRunning* on page 50

GetBoardInfo

Purpose

To get information about boards currently registered in ControlDesk's Platform Manager.

Syntax

```
board_info = mlib('GetBoardInfo')
```

Description

This result is returned as an M-by-1 vector of structures, where M is equal to the number of boards currently registered in ControlDesk's Platform Manager and where each structure contains the following fields:

Field	Description
.name	Name of the board
.type	Type of the board
.index	Board's registration index at the device driver

This function can be invoked at any time, which means without calling the SelectBoard function first.

Example

```
% Get the board info
board_info    = mlib('GetBoardInfo');
% get the number of boards currently registered
num_of_board = size(board_info,1);
% display info
printf('Registered processor boards\ttype\tindex\n');
for i = 0:num_of_board,
   fprintf('\t\t%s\t%s\t%d\n', board_info(i).name, ...
      board_info(i).type, board_info(i).index);
end
% select first registered processor board
mlib('SelectBoard', baord_info(1).name);
```

Related topics

References

- *SelectBoard* on page 60

GetMapVar

Purpose

To obtain the descriptor for a global variable.

Syntax

```
var_vec = mlib('GetMapVar', variable_names)
var_vec = mlib('GetMapVar', variable_names, ...
   'property_name', property_value, ...)
[var1, ..., varN] = mlib('GetMapVar', variable_names)
[var1, ..., varN]= mlib('GetMapVar', variable_names, ...
   'property_name', property_value, ...)
```

Description

This result returns descriptors for the variables specified in `variable_names`. The returned variables must be declared as global symbols in the currently running real-time application.

The `variable_names` parameter must be an M-by-1 cell array, where M is the number of variables whose descriptor are to be obtained and where each cell contains a symbolic name for one variable. `var_vec` is therefor an M-by-1 vector of descriptors. If M=1 (a descriptor for only one variable is to be obtained), the function also accepts a string as an input parameter. `var_vec` is then a simple structure.

If multiple output arguments are specified (`var1,..., varN`), the descriptors corresponding to the specified variables are assigned to the separate output parameters.

The number of parameters on the left-hand side must match the number of variable names on the right-hand side (N = M).

The property/value pairs apply to all specified variables. If the properties are not explicitly specified in the function call, MLIB/MTRACE uses default values:

CPU	Name of the CPU on which the variables reside.

If you select MP mode, you must specify this property. Otherwise, an error message is displayed.

Type Data type of the specified variable(s). Use the abbreviations from the following table to specify the data type:

Abbreviation	Description
`Int8`	8-bit signed char
`UInt8`	8-bit unsigned char
`Int16`	16-bit signed short
`UInt16`	16-bit unsigned short
`Int32`	32-bit signed integer
`UInt32`	32-bit unsigned integer
`FloatIEEE32`	32-bit single-precision floating-point type in IEEE 754 format
`FloatIEEE64`	64-bit double-precision floating-point type in IEEE 754 format, default for DS1103, DS1104, DS1401, DS1005 and DS1006 boards
`Int8Ptr`	Pointer to 8-bit signed char
`UInt8Ptr`	Pointer to 8-bit unsigned char
`Int16Ptr`	Pointer to 16-bit signed short
`UInt16Ptr`	16-bit unsigned short
`Int32Ptr`	Pointer to 32-bit signed integer
`UInt32Ptr`	Pointer to 32-bit unsigned integer
`FloatIEEE32Ptr`	Pointer to 32-bit single-precision floating-point type in IEEE 754 format
`FloatIEEE64Ptr`	Pointer to 64-bit double-precision floating-point type in IEEE 754 format

'Offset' Offset of the first element relative to the basic address. The default value is 0.

The calculation of the actual memory address depends on the data type of the variable:

- For directly addressed variables:

  `addr = var.addr + sizeof(var.type)*offset`

- For variables referenced by pointers:

  `addr = *var.addr + sizeof(referenced.type)*offset`

'Length' Number of elements as an integer number. The default value is 1.

You can change the default properties of each variable descriptor by using the SetVarProperty function.

The variable addresses are retrieved from the linker map file generated by the linker. The MAP file must reside in the same folder as the associated object files (`PPC` or `X86`). A variable declared as static, or local variables within the function body, cannot be accessed.

Example

Example 1:

```
% obtain the descriptors for variables declared
% in the real-time application running on DS1005 board:
% int simState;
% double * currentTime;
% double finalTime;
var_names = {'simState'; 'currentTime'; 'finalTime'};
var       = mlib('GetMapVar',vars_names);
% set the appropriate data types
var(1)    = mlib('SetVarProperty', var(1), 'type', 'Int32');
var(2)    = mlib('SetVarProperty', var(2), 'type', ...
   'FloatIeee64Ptr');
% var(3)    = mlib('SetVarProperty', 'var(3)', 'type', ...
%   'FloatIeee64'); not necessary, FloatIeee64 is default type
% for DS1005 board
```

Example 2:

```
% obtain the descriptors for variables declared
% in the real-time application running on DS1005 board:
% float mass, damping, spring_coefficient;
var_names = {'mass'; 'damping'; 'spring_coefficient'};
var       = mlib('GetMapVar', var_names, 'type', ...
   'FloatIeee32');
```

Related topics

References

- *GetTrcVar* on page 45
- *SetVarProperty* on page 65

GetTrcVar

Purpose

To obtain the descriptor for a variable specified within the TRC file.

Syntax SP

```
var_vec           = mlib('GetTrcVar', variable_names)
[var1, ..., varN] = mlib('GetTrcVar', variable_names)
```

Syntax MP

```
var_vec           = mlib('GetTrcVar', variable_names, 'CPU', cpu_name)
[var1, ..., varN] = mlib('GetTrcVar', variable_names, 'CPU', cpu_name)
```

Description

If a real-time application is generated from a Simulink model using the Real-Time Workshop and Real-Time Interface, a TRC file (TRC) is generated automatically. The TRC file maps Simulink variable names to the corresponding variables in the generated code (source code variables). Variables in real-time applications can therefore be accessed via the names of Simulink blocks, or via the labels of the corresponding signal lines.

To describe accessible variables within an application program, you can also create the TRC file manually. You can structure the information hierarchically by assigning variables to groups and subgroups.

GetTrcVar can be used to find the descriptors of one or more variables renamed in the TRC file (referred to as trace variables below). Specify the names of the variables in the `variable_names` input parameter.

The `variable_names` parameter must be an M-by-1 cell array, where M is equal to the number of trace variables whose descriptor are to be obtained and where each cell contains the name of one variable. `var_vec` is an M-by-1 vector of structures described in the chapter *Naming Conventions* on page 30. If M=1, which means that the descriptor of only one trace variable is to be obtained, the function also accepts a simple string as an input parameter. `var_vec` is then a simple structure.

If multiple output arguments are specified, `var1,...,varN`, the variable descriptors corresponding to the specified variables are assigned separately to each output argument.

The number of variable descriptors on the left-hand side must match the number of variable names on the right hand-side (N = M).

The properties of each variable descriptor (such as the address, data type, length) are retrieved from the TRC file.

You can also change the properties by using the SetVarProperty function.

If the trace variables are identical to the Simulink variables, the syntax for their names is as follows:

Syntax	Description
`Model Root/<subsystem>/<blockName>/<parameterName>`	For block parameters
`Model Root/<subsystem>/<blockName>/Out<i>`	For i-th block outputs
`Model Root/<subsystem>/<blockName>/ContState`	For continuous states of a block
`Model Root/<subsystem>/<blockName>/DiscState`	For discrete states of a block

Syntax	Description
`Model Root/<subsystem>/<blockName>/ContStateDeriv`	For state derivatives of a block
`Model Root/<subsystem>/<outportName>`	For outputs of a subsystem
`Labels/<label>`	For labeled signals

For the parameters of the Look-Up tables the following syntax applies:

Syntax	Description
`Model Root/<subsystem>/<blockName>/InputValues`	For the vector with input values of the 1-D Look-Up table block
`Model Root/<subsystem>/<blockName>/OutputValues`	For the vector with output values of the 1-D Look-Up table block
`Model Root/<subsystem>/<blockName>/Look-Up\nTable (2D)/Row Index`	For the row vector of the 2-D Look-Up table block
`Model Root/<subsystem>/<blockName>/Look-Up\nTable (2D)/ColumnIndex`	For the column vector of the 2-D Look-Up table block
`Model Root/<subsystem>/<blockName>/Look-Up\nTable (2D)/OutputValues`	For the table matrix of the 2-D Look-Up table block

`subsystem`is the path name of the Simulink system where the corresponding block resides, but with the model name omitted. Refer to the *Simulink User's Guide* by The MathWorks for Simulink system identifiers. You can specify output of the block gain block `Kp`, which resides in the subsystem controller of the model vcfp, with `Model Root/controller/Kp/Out1`. You can specify `Gain` parameter with `Model Root/controller/Kp/Gain`.

Labeled signals are block outputs in the Simulink model which are represented by labeled signal lines. `label` denotes the label of the signal to be specified.

In many cases, the subsystem, block or label string does not exactly match the name (with model name omitted) of the associated Simulink block or signal label, for example, if carriage returns or slashes are involved, or if labels are identical. In ambiguous cases, it is recommended to use the TrcView utility to obtain variable names.

Description of vectorized variables in TRC file

The vectorized variables can be described in the TRC file by declaring of the first and last index:

```
myVector[10..99]
{
   type: flt(64,IEEE)
   alias: "myVector"
   flags: PARAM
}
```

The vectorized variables can also be described in the TRC file by specifying the vectorized data type:

```
typedef ArrayFlt_90 flt(64,IEEE)[10..99]
myVector[10]
{
   type: ArrayFlt_90
   alias: “myVector”
   flags: PARAM
}
```

In the first case, the GetTrcVar function can only access single vector elements in which the vector is disassembled by the TRC file parser. Assuming that this vector is placed in the User Variables group, the call to the GetTrcVar function looks like this:

```
elem_10 = mlib('GetTrcVar', 'User Variables/myVector[10]')
```

to get the descriptor for the 10th element, or

```
elem_34 = mlib('GetTrcVar', 'User Variables/myVector[34]')
```

to get the description for the 34th element, etc.

To get the descriptor for the whole vector, use the following calling sequence:

```
myVectorDesc = mlib('GetTrcVar','UserVariables/myVector[10]');
myVectorDesc = mlib('SetVarProperty', myVectorDesc, 'Length', 90);
```

In the second case, the GetTrcVar function returns the descriptor for the whole vector. Since the vectorized parameters, outputs, states and state derivatives of the Simulink block are generated in the TRC file in this manner, the retrieved descriptor always describes all of the vector elements.

If you create TRC files manually, the following general syntax applies to variable names:

`group/subgroup1/.../subgroupN/name`.

For example, variable Kp in the subsystem controller of the top level group ABS system can be accessed with `ABS system/controller/Kp`.

Example

```
% set the parameter 'Amplitude' and 'Frequency'
% of the Simulink block named 'Signal Generator'
par_names    = { ...
      Model Root/Signal Generator/'Amplitude' ; ...
      Model Root/Signal Generator/'Frequency' ; ...
   };
param        = mlib('GetTrcVar', par_names);
% read the value for amplitude and frequency
param_val    = mlib('Read', param);
```

```
% change the value of amplitude
param_val{1} = param_val{1} + 0.1;
% change the value of frequency
param_val{2} = param_val{2} + 10;
% write new value into real-time processor
mlib('Write', param, 'Data', param_val);
```

Related topics

Basics

- *Accessing Matrices with MLIB/MTRACE* on page 78

References

- *GetMapVar* on page 43
- *SetVarProperty* on page 65

Intrpt

Purpose

To generate an interrupt on the real-time processor.

Syntax

`mlib('Intrpt')`

Result

A hardware interrupt is generated on the real-time processor. The names of the interrupt service routines in C code are:

Board	Name
DS1103	Name as registered by `DS1103_set_interrupt_vector()`
DS1104	Name as registered by `DS1104_set_interrupt_vector()`
DS1401	Name as registered by `DS1401_set_interrupt_vector()`
DS1005	Name as registered by `DS1005_set_interrupt_vector()`
DS1006	Name as registered by `DS1006_set_interrupt_vector()`

The interrupt service routine must be present in the real-time program. The respective interrupt vector must be initialized and the interrupt enabled (see the demo application program `adc_XXXX_hc.c` in the folder `%DSPACE_ROOT%\demos\dsXXXX\mlib`).

Example

```
% request a hardware interrupt on a DS1005
% processor board
mlib('SelectBoard', 'DS1005');
...
mlib('Intrpt');
```

IsApplRunning

Purpose

To check if an application is currently running on the real-processor board(s).

Syntax

```
state = mlib('IsApplRunning')
```

Result

`state` will show if an application is running

Description

If the SP mode is selected (default), `state` is an integer value that equals zero or 1. If the MP mode is selected, state is an M-by-1 vector of structures, where M is equal to the number of CPUs in an MP application and where each structure contains the following fields:

Name	Description
.board	Board name
.cpu	CPU name
.appl_state	State of the program loaded on the CPU. This is an integer value equal to 0 or 1.

In both SP and MP mode, the meanings of the state values are:

- 1 – application is running
- 0 – application is not running

Example

```
if mlib('IsApplRunning')
   str  = 'Currently running real-time application is:';
   appl = mlib('GetApplInfo');
   disp([str appl.name]);
else
   disp('No real-time application running\n');
end
```

Related topics	References • *GetApplInfo* on page 40

LoadFile

Purpose

To load selected portions of data from the MAT file generated by MLIB/MTRACE during continuous data acquisition with the **StreamToDisk** option.

Syntax

```
data = mlib('LoadFile', mat_file_name)
data = mlib('LoadFile', mat_file_name, ...
   property_name, property_value)
```

Result

The data you selected is loaded.

Description

`mat_file_name`is a string containing the name of the file generated by MLIB/MTRACE in MAT format. The file name extension `.mat` can be omitted. By default, all data stored in the specified MAT file is returned to the `data` output matrix. Each row of this matrix represents a time series of one variable. If the data stored in the MAT file was collected without time stamps, its number of rows corresponds to the number of selected trace variables; otherwise, the number of rows corresponds to the number of selected trace variables +1 (the first row is for the time stamps). The rows in `data` are assigned to the trace variables in the order in which the variables were selected with the TraceVars property.

In many cases the amount of collected data can be too large to load it all into the MATLAB workspace. Here you can specify a portion of data to be loaded by setting appropriate property/value pairs:

Variables An M-by-1 cell array of variable names specifying the variables for which the data will be loaded. The number of rows of the data matrix corresponds to M, or to M +1 if the time stamps were stored in the MAT file. The rows in the data are assigned to the specified variables in the order in which the variables were selected. If data for only one variable is to be loaded, the function also accepts a string as a parameter. By default, the data for all variables is returned.

CPUs An M-by-1 cell array of CPU names. You must set this property if data acquisition was performed in the multiprocessor mode and some of the variables residing on the different CPUs have the same name. To distinguish between variables, the CPUs must be specified in addition to variable names (property Variables). By default, MLIB/MTRACE assumes that all variable names are unique and loads the data corresponding to the first variable found with the specified name.

Samples A2-by-1 vector of integer values specifying the first and last samples of the interval for which the data is to be loaded. The number of columns of the data matrix results from the interval length. By default, the whole time series is loaded.

Interval A2-by-1 vector of double values specifying the interval (in the units of the step size) from which the data is to be loaded. The number of columns of the data matrix results from the interval length divided by the step size and downsampling factor. By default, the whole time series is loaded.

Example

```
% The data acquisition was performed in SP mode
% for variable 'Model Root/Signal Generator/Out1',
% 'Model Root/Plant/y/Out1' and 'Model Root/Controller/out/out1'
% and stored into MAT file exp_data.mat
% load the whole time series for variables 'Model Root/Plant/y/Out1'
% and 'Model Root/Signal Generator/Out1'
variables = {'Model Root/Plant/y/Out1'; ...
   'Model Root/Signal Generator/Out1'}; ...
data = mlib('LoadFile', 'exp_data', ...
   'Variables', variables);
...
% load the data for the selected variable in the
% specified interval
data = mlib('LoadFile', 'exp_data', ...
   'Variables', variables, ...
   'Samples', [1000; 2000]);
% data is a 2 x 1001 matrix
...
```

Related topics

Basics

- *MLIB/MTRACE Benchmarks* on page 125

LoadVarFile

Purpose	To load the variable description file.
Syntax	`mlib('LoadVarFile', var_file_name)`

Description

For single-processor environments, the variable description file is identical to the TRC file. For multiprocessor environments, the variable description file is identical to the SDF file. For further information on the TRC and SDF files, refer to *Creating System Description Files* (📖 *ControlDesk Experiment Guide*).

By default, MLIB/MTRACE assumes single-processor mode and that the name of the TRC file matches the name of the application currently running on selected board. If the name of the TRC file differs from the name of the application program or if there are several variable description files, you must use this function to specify the one required for MLIB/ MTRACE. Only variables available in a currently loaded TRC file can be accessed by the GetTrcVar function. Loading a TRC file while the multiprocessor mode is active switches MLIB/MTRACE to single-processor mode.

If an SDF file is loaded, the multiprocessor mode is switched on. In this case the function is similar to the Multiprocessing function.

The `var_file_name` must be a string with the file name extension TRC or SDF. If the variable file resides in the MATLAB working folder, only the file name of the relative path is sufficient, otherwise the absolute path must be specified.

Example

```
% Select DS1005 board for use with MLIB/MTRACE.
% Note, that the application smdtf_1005_sl.ppc
% is currently running on selected board.
mlib('SelectBoard','DS1005');
% load the variable file smdtf_1005_sl_set1.trc
mlib('LoadVarFile','smdtf_1005_sl_set1.trc');
```

LockProgram

Purpose

To set a lock on an application program.

Syntax

```
mlib('LockProgram')
```

Result

The application program currently running on the selected processor board is locked. The application program lock ensures that the real-time program cannot be mistakenly aborted or restarted by other host applications while MLIB/MTRACE functions are in progress.

The real-time program remains locked until the MLIB function UnlockProgram is explicitly called.

Example

```
% lock real-time processor
% to prevent other host applications
% from interfering
mlib('LockProgram');
% some calls to MLIB/MTRACE functions
...
% MLIB/MTRACE has finished, give way to other
% host applications
mlib('UnlockProgram');
```

Related topics

References
- *UnlockProgram* on page 71

Multiprocessing

Purpose

To set MLIB/MTRACE to multiprocessor mode.

Syntax

```
mlib('Multiprocessing', 'sdf_file_name')
mlib('Multiprocessing', 'off')
```

Result

MLIB/MTRACE is in the multiprocessor mode.

Description

Because MLIB/MTRACE is in single-processor mode by default, you must activate the multiprocessor mode if MLIB/MTRACE is used in a multiprocessor environment. `Sdf_file_name` is the name of the system description file (SDF) of the currently running multiprocessor application. YOu must specify the full path name if the M file calling MLIB/MTRACE resides in a folder other than the files associated with the application (PPC, X86, MAP, etc.). Invoking `Multiprocessing` with the keyword off returns MLIB/MTRACE to the single-processor mode.

Example

```
% select board
mlib('SelectBoard', 'DS1005');
mlib('Multiprocessing', 'pipt1_dual1005_sl.sdf');
% access variables in the multiprocessor system
...
mlib('Multiprocessing', 'off');
% single-processor on DS1005...
```

Read

Purpose

To read data from the real-time processor.

Syntax

```
data                = mlib('Read', var_vec)
[data1, ..., dataN] = mlib('Read', var_vec)
```

Result

The data is read from the real-time processor.

Description

The `var_vec` parameter is an M-by-1 vector of variable descriptors as returned by the GetTrcVar or GetMapVar function (see *Naming Conventions* on page 30). A vector of data is read for each variable descriptor. The data type and the number of elements are contained in the variable descriptor (`var_vec(i).type` and `var_vec(i).length`, respectively). The data vector is returned in the i^{th} cell of the M-by-1 cell array data.

If you specify multiple output arguments (`data1, ..., dataN`), each data vector read from the real-time hardware is assigned to a separate output.

The number of arguments on the left-hand side must match the number of variable descriptors on the right-hand side (N = M).

Example

Example 1:

```
% get the descriptor of the variables simState,
% currentTime and read their values
% obtain the addresses
variables   = {'simState'; 'currenTime'}
vars        = mlib('GetTrcVar', variables);
% read the values
data        = mlib('Read', vars);
simState    = data{1};
currentTime = data{2};
```

The last 3 statements can optionally be replaced by

```
[simState, currentTime] = mlib('Read', vars);
```

Example 2:

```
% read the 3 values of the variable int_vector and 5
% values of the variable dbl_vector
% get the variable descriptors
int_vector_desc = mlib('GetMapVar', 'int_vector', 'type', ...
   'Int32', 'length', 3);
dbl_vector_desc = mlib('GetMapVar', 'dbl_vector', 'type', ...
   'FloatIeee64', 'length', 5);
[int_vector, dbl_vector] = mlib('Read', ...
   [int_vector_desc; dbl_vector_desc]);
```

Related topics

References

- *ReadDirect* on page 57
- *Write* on page 71

ReadDirect

Purpose

To read data directly from the real-time processor.

Syntax

```
data                 = mlib('ReadDirect', var_vec)
[data1, ..., dataN] = mlib('ReadDirect', var_vec)
```

Result

The data is read directly from the real-time processor.

Description

The `var_vec` parameter is an M-by-1 vector of variable descriptors as returned by the GetTrcVar or GetMapVar function (see *Naming Conventions* on page 30 for further information). A vector of data is read or each variable descriptor. The data type and the number of elements are contained in the variable descriptors (`var_vec(i).type` and `var_vec(i).length`, respectively). The data vector is returned in the `i`th cell of the M-by-1 cell array data.

If you specify multiple output arguments (`data1,..., dataN`), each data vector read from the real-time hardware is assigned to a separate output.

- The number of arguments on the left-hand side must match the number of variable descriptors on the right-hand side (N=M).
- Only global variables residing on DS1104, the global memory of DS1103 and DS1401 boards and non cached global memory of DS1005 or DS1006 can be accessed. These limitations do not apply to the MLIB/MTRACE function Read. However, the Read function requires that the application is running and the special services `host_service` and `master_cmd_server` are called. If the application has crashed and it is necessary to read the current memory state (for example, to find the reason for the crash), only the ReadDirect function can be used.

Example

```
% Get descriptor of test variable NoOfIteration
% and read its value. Note: the variable resides in
% noncached global memory of DS1005 board.
NoOfIter_desc =
   mlib('GetMapVar','NoOfIteration','Type','Int32');
NoOfIter_val = mlib('ReadDirect',NoOfIter_desc);
```

SearchMap

Purpose	To search the linker map file for variables whose names match the specified regular expression.
Syntax SP	`vars = mlib('SearchMap', regexp)`
Syntax MP	`vars = mlib('SearchMap', regexp, 'CPU', cpu_name)`

Result

The linker map file corresponding to the currently running real-time application is searched for variables whose names match the specified pattern `regexp`. These names are stored in a column cell array `vars`, where each cell contains one variable name as a string.

The pattern `regexp` must be a regular expression. The regular expression allowed by MLIB/MTRACE is constructed as follows:

- A period (.) matches any character.
- An asterik (*) matches zero or more occurrences.
- A plus (+) matches one or more occurrences.
- A question mark (?) matches zero or one occurrence of the previous pattern.
- A caret (^) indicates the beginning of a line.
- A dollar sign ($) indicates the end of a line.
- A non empty string enclosed in ([]) brackets matches any single character in that string. ASCII intervals such as [A-Z] are allowed. The class of characters that are not expected is indicated by a caret after a left bracket, for example, [^0-9].
- Any special character such as ., *, [,] etc. must be preceded by a backslash (\) if you want it match itself.

Example

```
% Assumed the user specific global variables are defined
% in the real-time application by
% float usr_var1, usr_var2, usr_var2 etc.
% get the names of all user specific variables
usr_vars     = mlib('SearchMap', '^usr_.*');
% obtain the descriptors for each of them
usr_var_desc = mlib('GetMapVar', usr_vars);
```

Related topics

References

- *SearchTrc* on page 59

SearchTrc

Purpose

To search the TRC file for variables whose names match the specified regular expression.

Syntax SP

`var_names = mlib('SearchTrc', regexp)`

Syntax MP

`var_names = mlib('SearchTrc', regexp, 'CPU', cpu_name)`

Result

The TRC file corresponding to the currently running real-time application is searched for variables whose names match the specified pattern `regexp`. These names are stored in a column cell array `var_names`, where each cell contains one variable name as a string.

The pattern `regexp` must be a regular expression. The regular expression allowed by MLIB/MTRACE is constructed as follows:

- A period (.) matches any character.
- An asterik (*) matches zero or more occurrences.
- A plus (+) matches one or more occurrences.
- A question mark (?) matches zero or one occurrence of the previous pattern.
- A caret (^) indicates the beginning of a line.
- A dollar sign ($) indicates the end of a line.
- A non empty string enclosed in ([]) brackets matches any single character in that string. ASCII intervals as [A-Z] are allowed. The class of characters that are not expected is indicated by a caret after a left bracket: [^0–9].
- The special character (as ., *, [,] etc.) must be preceded by a backslash (\) if you want it match itself.

Example

```
% Example
% Get the names of all Simulink block outputs
% contained in the subsystem 'PID-Controller'
output_names   = mlib('SearchTRC','.*PID-Controller/.*/Out1');
% Obtain the descriptors for found block outputs
output_desc   = mlib('GetTrcVar',output_names);
```

Related topics

References

- *SearchMap* on page 58

SelectBoard

Purpose

To select a processor board for use with MLIB/MTRACE.

Syntax

```
mlib(´SelectBoard´)
mlib('SelectBoard', 'board')
```

Description

If the parameter board is omitted, the working board set by ControlDesk's Platform Manager is used.

The optional parameter board can be either an ASCII string or an integer value. If it is an ASCII string, the board that is registered with this name in ControlDesk's Platform Manager is selected. If an integer value is specified, the board registered with that index is selected. The selected real-time processor board is used for subsequent MLIB/MTRACE calls.

To obtain a list of the boards which are currently registered in ControlDesk's Platform Manager, invoke in the MATLAB Command Window:

```
boards = mlib('GetBoardInfo')
```

Example

```
% select first registered board
boards = mlib('GetBoardInfo');
mlib('SelectBoard', boards(1).name);
...
% switch to second one
mlib('SelectBoard', boards(2).name);
...
```

Set

Purpose

To set the data acquisition options.

Syntax

```
mlib('Set', property_name, property_value, ...)
```

Result

The data acquisition options, such as the time frame, triggering and variables to be traced, are set by defining the appropriate property/value pairs.

The permissible property/value pairs for data acquisition are listed below:

ServiceGroup (MP1005 or MP1006) Name of the service group used for data acquisition on multiprocessor systems based on DS1005 or DS1006.

Service An integer number (1 … 31) specifying which data capture service in the real-time program is to perform data acquisition (MLIB/MTRACE can distinguish between up to 31 different services). The default is service number 1.

Trigger The trigger mode is turned on or off.

- 'ON' – trigger mode enabled
- 'OFF' – free-running mode (default value)

TriggerVariable Descriptor of the trigger variable

TriggerLevel Value of the trigger level. Default value: 0.0

TriggerEdge The trigger edge is specified:

- 'rising' – trigger on rising edge (default value)
- 'falling' – trigger on falling edge

TraceVars Variables to be traced specified as an M-by-1 vector of variable descriptors

StepSize The step size of the task containing the corresponding host_service macro. For example, if the selected (or default) service is contained in the timer task routine, the step size is identical to the sampling period.

If the step size is defined, you can specify the trace interval either by the **Start/Stop** properties or by **Delay/NumSamples**, otherwise only by **Delay/NumSamples**.

Downsampling A positive integer value used to reduce the amount of data per capture. A downsampling factor of n forces MLIB/MTRACE to store a set of data only every n-th time the service macro is called in the real-time program. Default value: 1.

TimeStamping Specifies if the time stamps at which the data was collected are to be traced or not

- 'ON' - time stamps are traced
- 'OFF'- time stamps are not traced (default)

In MP1005 or MP1006 mode, this property is set to 'ON', and you cannot modify it.

UseAbsoluteTime Specifies if the time stamps are to be stated as absolute times or relative to the absolute zero point of the trace shot. In triggered mode, the time at which the trigger event occurred is taken as absolute zero, while in the free-running mode, the time stamp of the data set first collected is absolute zero.

'ON' - use the absolute time

'OFF'- use the time relative to absolute zero (default value)

Start The start of the trace interval for which data is to be collected, specified in units of the step size. If the trigger mode is set to 'ON', a negative value indicates the pre-trigger mode, and a positive value indicates the post-trigger mode (you must define the step size beforehand).

Stop The end of the trace interval for which the data is to be collected, specified in the units of the step size (you must define the step size beforehand).

Delay Delay related to the trigger event specified in the number of samples. A negative value indicates the pre-trigger mode, and a positive value indicates the post-trigger mode (default value: 0).

NumSamples Length of the trace interval specified in the number of samples. Default value: 200.

AcquisitionMode The mode of the data capture is specified.

- 'continuous' – the data is collected until StopCapture is called. The trigger must be set to 'OFF'. This mode is useful if you want to acquire large amounts of data over a long time period.
- 'single_shot' – a single data capture whose length is specified by the **Start/Stop** or **Delay/NumSamples** properties (default value)

StreamToDisk In 'continuous' mode the collected data can optionally be saved on the hard disk:

- 'ON' – data is saved in a MAT file on the hard disk
- 'OFF'– data is not saved in a MAT file (default value) on the hard disk

OutputFormat Specifies the format in which the traced data is available in the MATLAB workspace.

- 'matrix' - the data format is a matrix (default value). If TimeStamping has been set to 'ON', the format is 'matrix with time'.
- 'structure' - the data format is a structure. If TimeStamping has been set to 'ON', the format is 'structure with time'.

For further information about the format, refer to function *FetchData* on page 34.

- The data is always stored in the MAT file as a matrix or a matrix with time, independently of the output format selected with this property.
- In MP1005 or MP1006 mode, the 'matrix' format means that the matrix with time is to be returned (because of the time stamps). However, the common time axis for the data collected on different processors can be found only if the data was traced with the same sampling period. If the data on different processors is traced by services called with different sampling periods or asynchronously, the output format must be set to 'structure'

FileName If StreamToDisk is set to 'ON', you must specify the name of the MAT file where the collected data is to be stored. You do not need to type the file name extension MAT. For more information, see *MLIB/MTRACE Benchmarks* on page 125.

MP1005 and MP1006 only:

The data collected on different processors contained in the multiprocessor system is saved to separate MAT files whose names must be unique. You can explicitly specify a MAT file name for each processor by using this property together with the CPU property. If the CPU property is not specified, the names of the MAT files are built according to the rule: <specifiedFileName>_<processorName>.mat

AutoClearBuffer Specifies if the trace buffer is to be cleared and freed after the last portion of collected data is fetched into the MATLAB workspace:

- 'ON' – the buffer is automatically cleared and freed (default value)
- 'OFF' – the buffer is not cleared and the data can be read multiple times.

TimeOutValue Integer value specifying the time in seconds after which a timeout is reported during execution of the FetchData function. TimeOut can occur if MLIB/MTRACE has to wait until the desired is reached (default value: 5).

InterpolateBoundaryPoints (MP1005 and MP1006 only)

Specifies if points are to be added (if necessary) on the boundaries of the trace interval. If yes, the values for the boundary points are obtained using the interpolation method. This property can be useful if data is captured asynchronously, for example, using the data acquisition service from an asynchronous task, during the specified time interval <t1,t2>. In this case only points with time stamps t>=t1 or t<=t2 are used. Because data acquisition is asynchronous, it can happen that there are no points exactly on the boundaries.

'ON' - add points to the boundaries of the trace interval

'OFF'- do not add boundaries to the trace interval (default value)

This property takes effect only if you specify the trace interval in the time unit, and not in samples.

MaxSamplesPerSecond Specifies the estimated number of calls per second to the asynchronous task containing the corresponding host_service macro.

CPU Name of the CPU. If you specify this property, the following properties (if specified) apply only to the selected CPU:

- Service
- FileName
- MaxSamplesPerSecond
- StepSize

Otherwise the above properties apply to all processors in the DS1005 or DS1006 multiprocessor application.

Example

Example 1:

```
% obtain the variable address
variables = {'Model Root/Signal Generator/Out1'; ...
   'Model Root/Plant/out'};
vars_desc = mlib('GetTrcVar', variables);
% specify the data capture options:
% triggering on the variable Model Root/Signal Generator/out1,
% trigger level 0.0, rising edge;
% 500 samples with 10 samples pre-trigger;
% variables to be traced:
% 'Model Root/Signal Generator/Out1' and,
% 'Model Root/Plant/out';
% single_shot mode, downsampling 1
% service number: 1
```

```
mlib('Set', 'Trigger', 'ON', ...
   'TriggerVariable', vars_desc(1), ...
   'TraceVars', vars_desc, ...
   'Delay', -10, ...
   'NumSamples', 500);
```

Example 2:

```
% MP1005 mode has been activated
% get the addresses of the trace variables
sigGen = mlib('GetTrcVar','Model Root/Signal Generator/Out1',...
   'CPU','Master') ;
enablsysOut = mlib('GetTrcVar','Model Root/enablsys/output',...
   'CPU','Slave') ;
% For RTI-MP-generated applications, the available service groups
% are automatically obtained from TRC files.
% If TRC files do not exist, or any other service group is desired as
% described in TRC files, the service group
% can be specified by the user.
% in the example below, a new service group is created.
% It allows that data can be traced simultaneously with
% synchronous (CPU Master) and asynchronous (CPU Slave) tasks
mlib('Set','ServiceGroup','myNewServiceGroup',...
   'CPU','Master',...
   'Service', 1,...
   'StepSize', 0.00012);
mlib('Set','ServiceGroup','myNewServiceGroup',...
   'CPU','Slave',...
   'Service', 2,...
   'MaxSamplesPerSecond', 100);
% This new service group can now be used for data acquisition
% The data is to be captured during the time interval <-0.00036; 0.048>
% The values for the asynchronously
% traced variable 'Model Root/enablesys/output
% are to be interpolated on boundaries of this trace interval
mlib('Set','ServiceGroup','myNewServiceGroup',...
   'Trigger','on',...
   'TriggerVariable', sigGen,...
   'Start', -0.00036,...
   'Stop', 0.048,...
   'TraceVariables',[sigGen; enablsysOut];
   'InterpolateBoundaryPoints','on');
```

Related topics

References

- *Get* on page 37

SetVarProperty

Purpose

To set the properties stored in the variable descriptor.

Syntax

```
new_var_vec = mlib('SetVarProperty', old_var_vec, ...
   'property_name', property_value, ...)
```

Result

The new property values are stored in the variable descriptor.

Description

The permissible properties are listed in the table below:

Name	Description
Name	Name of the variable
CPU	Name of the CPU on which the variable resides. In MP mode, this property must not be changed after the variable address was obtained with GetTrcVar or GetMapVar.
Type	Data type of the variable. For possible data type specifications, see the description of the GetMapVar function.
Addr	The variable's base address
Offset	Offset of the first element from the base address as an integer number
RelativeOffset	Offset of the first element from the current memory address
Length	Number of elements as an integer number

The calculation of the current memory address depends on the data type of the variable:

- For directly addressed variables:

```
addr = var.addr * sizeof(var.type) * offset
```

- For variables referenced by pointers:

```
addr = *var.addr + sizeof(referenced.type) *
offset
```

The specified property/value pairs apply to all variable descriptors stored in the column vector `old_var_vec`. The modified variable descriptors are returned to the column vector `new_var_vec`. If `old_var_vec` is empty, a variable descriptor with specified properties is created. To create a descriptor, you must define the following properties Name, Addr, Type and CPU (for MP mode). The defaults are used for the others: Offset = 0, Length = 1.

Example

```
% obtain the descriptors of variables declared
% in the DS1005 real-time application by:
% int    int_vect[5];
% float  flt_vect[10];
% double dbl_vect[10];
var_names    = {'int_vect'; 'flt_vect'; 'dbl_vect'};
[int_vect_var, flt_vect_var, dbl_vect_var] = ...
   mlib('GetMapVar', var_names);
% set the appropriate data types and lengths
int_vect_var = mlib('SetVarProperty', int_vect_var, ...
   'type', 'Int32', 'length', 5);
flt_vect_var = mlib('SetVarProperty', flt_vect_var, ...
   'type', 'FloatIeee32', 'length', 10);
dbl_vect_var = mlib('SetVarProperty', dbl_vect_var, ...
   'type', 'FloatIeee64', 'length', 10);
% read the values of the variable float_vect
float_vect   = mlib('Read', flt_vect_var);
```

Related topics

References

- *GetMapVar* on page 43
- *GetTrcVar* on page 45

StartCapture

Purpose

To start a new capture on the real-time processor board

Syntax SP

```
mlib('StartCapture')
mlib('StartCapture', services)
```

Syntax MP

```
mlib('StartCapture')

mlib('StartCapture',serviceGroups)
```

Result

The data capture process is initialized.

Description

If there is an appropriate service in the real-time application, a set of data (sample) is collected once per call of that service (host_service: see the chapter *Preparing Real-Time Applications* on page 18). Multiple services can be used simultaneously in one real-time application, for example, in different tasks.

With the `services/serviceGroups` parameter, you can specify which of the services will perform the data acquisition process. If the `services/serviceGroups` parameter is omitted, the default service or service group is used. Otherwise, MLIB takes the service number or the name of the service group which is specified by the `services/serviceGroups` parameter.

In some cases it is desirable to have multiple data captures performed by different data capture services or service groups that start simultaneously. The options of the data capture process – traced variable, trigger, time frame, etc. - for each of the services or service groups must already be defined. For further information, refer to *Multiple MLIB/MTRACE Data Captures in One Real-Time Application* on page 76. In such cases

- The `services` parameter is a vector containing the service numbers.
- The `serviceGroups` parameter is a cell array containing the names of the service groups.

Example

```
% start a capture for an application generated by the RTI
mlib('StartCapture');
while mlib('CaptureState')~=0, end
mat_data = mlib('FetchData');
...
% start capturing data using the data capture service
% number 2
mlib('StartCapture', 2);
```

Related topics

References
- *CaptureState* on page 32
- *StopCapture* on page 68

StopCapture

Purpose

To stop the data capture on the selected processor board.

Syntax SP

```
mlib('StopCapture')
mlib('StopCapture', services)
```

Syntax MP

```
mlib('StopCapture')
mlib('StopCapture', serviceGroups)
```

Result

The data capture process started with the function StartCapture is stopped.

Description

If the additional `service/serviceGroups` parameter is not specified, MLIB stops the data acquisition performed by the default service or service group. Otherwise, you must specify the service number or the name of the service group with the `services/serviceGroups` parameter.

If multiple data captures performed by different services or service groups were started simultaneously, the `services/serviceGroups` parameter must be:

- A vector containing the numbers of services (SP) or
- A cell array containing the names of the service groups (MP1005 and MP1006)

that are to be stopped.

If the data capture was started in *single_shot* mode, it will be stopped automatically after the desired number of samples is collected. If the data capture was started in *continuous* mode, you must invoke StopCapture to stop it.

Example

```
% start data capture in 'continuous' mode
mlib('StartCapture');
TIC;
while TOC < 600
   data = mlib('FetchData', 300);
   plot(data')
   ...
end
% stop the capture after 10 minutes (600 sec)
mlib('StopCapture');
```

Related topics

References

- *CaptureState* on page 32
- *StartCapture* on page 67

TriggerState

Purpose

To get trigger status information.

Syntax SP

```
status = mlib('TriggerState')
status = mlib('TriggerState', services)
```

Syntax MP

```
status = mlib('TriggerState')
status = mlib('TriggerState', serviceGroups)
```

Result

The trigger status information is obtained.

Description

When used without an input parameter, this function returns the trigger status information for the capture process performed by the default service or the service group. Otherwise, you must specify the service or service group via the `services/serviceGroup` parameter.

The possible values for status are:

Value	Description
2	Trigger is disabled
1	Trigger armed, waiting for a trigger event
0	Trigger event has occurred
-1	Trigger not armed

If multiple data captures performed by different services or service groups are started simultaneously, you can check the trigger states of all of them by calling the TriggerState function with the additional `services/serviceGroups` input parameter, which is then

- A vector containing the service numbers `(services)`
- A cell array containing the names of the service groups (`serviceGroups`)

carrying out the data acquisition.

In this case, `state` is a vector whose elements indicate the trigger state of the corresponding capture process.

Example

```
% waiting for a trigger event
while mlib('TriggerState', 2) ~= 0, end;
disp('Trigger event occurred');
```

Related topics	Basics • *Additional MLIB/MTRACE Programming Examples* on page 94

UnlockProgram

Purpose

To release a lock on the application program.

Syntax

```
mlib('UnlockProgram')
mlibn('UnlockProgram')
```

Result

A previously set lock on the real-time program is released.

Example

```
% lock application program to prevent other host
% applications from interfering
mlib('LockProgram');
% some calls to MLIB/MTRACE functions
...
% MLIB/MTRACE has finished, give way to other
% host applications
mlib('UnlockProgram');
```

Related topics

References
• *LockProgram* on page 54

Write

Purpose

To write data to the real-time processor.

Syntax

```
mlib('Write', var_vec, 'Data', data)
```

Result

The data is written to the real-time processor.

Description

The `var_vec` parameter is an M-by-1 vector of variable descriptors as returned by GetTrcVar or GetMapVar (see *Naming Conventions* on page 30). The `data` parameter must be an M-by-1 cell array. A vector matrix of data contained in the corresponding cell of the `data` parameter is written for each variable descriptor. The data type and the number of elements are specified in the variable descriptor (`var_vec(i).type` and `var_vec(i).length`, respectively). The data contained in a matrix is written column-wise.

If M = 1, `data` can be a simple vector or matrix. It does not need to be a 1-by-1 cell array in this case.

To reduce execution times, MLIB/MTRACE does not provide any range checking (depending on the data type) for specified values. It is your responsibility to check variable ranges, for example, for integer data types.

Example

```
% write a vector of parameters in double precision format
% and a vector of parameters in single precision format
% to memory of the DS1005 board
dbl_data    = [0.543212 pi];
flt_data    = [12.12 8.21 0.12 0.36];
% get the variable descriptors; note that the default
% value of the data type is FloatIeee64 for DS1005 board
[dbl_vec_var, flt_vec_var] = mlib('GetMapVar', ...
   {'dbl_vector'; 'flt_vector'});
% set the proper data type
flt_vec_var = mlib('SetVarProperty', flt_vec_var, ...
   'Type', 'FloatIeee32', 'Length', size(flt_data, 1));
dbl_vec_var = mlib('SetVarProperty', dbl_vec_var, ...
   ''Length', size(dbl_data, 1));
% write the data into real-time processor,
mlib('Write', [dbl_vec_var; flt_vec_var], 'Data', ...
   {dbl_data; flt_data});
```

Related topics

References

- *Read* on page 55
- *ReadDirect* on page 57
- *WriteDirect* on page 73

WriteDirect

Purpose

To write data directly to the real-time processor.

Syntax

```
mlib('WriteDirect', var_vec, 'Data', data)
```

Result

The data is written directly to the real-time processor.

Description

The var_vec parameter is an M-by-1 vector of variable descriptors as returned by GetTrcVar or GetMapVar. (see *Naming Conventions* on page 30 for further information). The `data` parameter must then be an M-by-1 cell array. A vector matrix of data contained in the corresponding cell of the `data` parameter is written for each variable descriptor. The data type and the number of elements are specified in the variable descriptor (`var_vec(i).type` and `var_vec(i).length`, respectively). The data contained in a matrix is written column-wise.

If M = 1, `data` can be a simple vector or matrix. It does not need to be a 1-by-1 cell array in this case.

- To reduce execution times, MLIB/MTRACE does not provide any range checking (depending on the data type) for specified values. It is your responsibility to check variable ranges, for example for integer data types.
- Only global variables residing on a DS1104, the global memory of a DS1103 or DS1401, and the non-cached memory of DS1005 or DS1006 can be accessed. These limitations do not affect the MLIB/MTRACE Write function. However, if the Write function is used for PPC boards, the special services host_service and master_cmd are required. This function is therefore slower than the WriteDirect function, which only uses the device driver.

Example

```
% write a vector of parameters in double precision format
% to memory of the DS1005 board
dbl_data = [12.12 8.21 0.12 0.36];
% get the variable descriptors, note that the default
% value of the data type is 'FloatIeee64' for DS1005 board
dbl_vec_desc = mlib('GetMapVar','dbl_vector',...
   'Length', length(dbl_data));
```

```
% write the data into real-time processor,
mlib('WriteDirect', dbl_vec_desc, 'Data', dbl_data);
```

Related topics

References

- *Read* on page 55
- *ReadDirect* on page 57
- *Write* on page 71

MLIB/MTRACE Programming

This topic contains the following sections:

- For some instructions on how to program MLIB/MTRACE, refer to *Programming Instructions* on page 76
- For information on accessing matrices with MLIB/MTRACE, refer to *Accessing Matrices with MLIB/MTRACE* on page 78
- For detailed demonstrations of programming with MLIB/MTRACE, refer to *MLIB/MTRACE Programming Examples* on page 88

Programming Instructions

For details on how to program multiple MLIB/MTRACE captures in one real-time application and how to lock the real-time application, refer to:

- *Multiple MLIB/MTRACE Data Captures in One Real-Time Application* on page 76
- *Locking the Real-Time Program* on page 77

Multiple MLIB/MTRACE Data Captures in One Real-Time Application

MLIB/MTRACE needs special service code to be executed in the real-time application to enable real-time data capture. This code collects one set of data for each sampling step.

You can call multiple service codes in different tasks of the real-time application to trace variables of periodic timer tasks and asynchronous interrupt-driven tasks simultaneously. MLIB/MTRACE can handle up to 10 services per real-time application.

The real-time program communicates with MLIB/MTRACE via the `host_service(nr, 0)` macro. To distinguish between multiple service routines, this macro must be called in the real-time program:

```
host_service(1,0);
host_service(2,0);
```

etc.

In the real-time applications generated by RTI, the service macros are automatically inserted into desired tasks if a DataCapture block is used.

Each service routine requires additional execution time in the application program. If many service macros are placed in a timer task on a processor that is already very busy, an overload may occur. The execution time needed by a service routine depends mainly on the number of variables and pointers to be traced.

With multiple MLIB/MTRACE captures, each service routine in a real-time program requires one separate set of data acquisition options.

Example

```
mlib('SelectBoard');
mlib('Set', 'Service', 1, ...
   'Trigger', 'ON', ...
   'TriggerVariable', trg_addr, ...
   'NumSamples', 1000, ...
   'TraceVars', trc_vars_0);
mlib('Set', 'Service', 2, ...
   'Trigger', 'ON', ...
   'TriggerVariable', trg_addr, ...
   'NumSamples', 1000, ...
   'Downsampling', 10, ...
   'TraceVars', trc_vars_1);
mlib('StartCapture', [1; 2]);
```

Related topics

Basics

- *Impact of MLIB/MTRACE on Execution Time* on page 128

Locking the Real-Time Program

The currently running real-time program can be locked with the MLIB/MTRACE function LockProgram to protect it from being restarted or corrupted by other host applications. All dSPACE PC software relies on the device driver which provides this locking mechanism. A real-time program can be locked by one or more host programs at a time. To unlock the real-time program, each host program must release its lock. MLIB/MTRACE releases a previously set lock when **`mlib('UnlockProgram')`** is invoked.

If a program is downloaded to a processor board by means of ControlDesk's Platform Manager after an MLIB/MTRACE instance locked the currently running real-time program, the error message 'Another program has locked the real-time program.' will appear. Return to MATLAB's Command Window and type **`mlib('UnlockProgram')`** to make the MLIB/MTRACE instance unlock the real-time program. If no other host programs have locked the program, the next download by ControlDesk's Platform Manager should be successful.

Accessing Matrices with MLIB/MTRACE

Objective

In the C code generated by the MATLAB's Real-Time Workshop and dSPACE's RTI, parameter matrices are represented by simple double-precision arrays. The index for finding a specific matrix element mainly depends on whether the matrix was written out row-wise or column-wise. In some cases, there are additional rules that you have to comply with if you wish to modify online transfer function parameters or look-up table data with MLIB/MTRACE. A good way to become familiar with the rules that apply when MATLAB's Real-Time Workshop maps parameter matrices to C code is to see the associated `model.prm` file.

For detailed information about the rules, refer to:

- *Built-in Simulink Blocks* on page 79
- *2-D Look-Up Tables* on page 79
- *S-Functions* on page 79

MLIB/MTRACE provides some M files that perform read and write operations on the matrices' and look-up tables' parameters. For details, see below.

Tools

MLIB/MTRACE supplies tools that let you perform Read/Write operations on the matrices' and lookup-tables' parameters. These tools are written as the M files `mlib_1dlookup.m`, `mlib_2dlookup.m`, and `mlib_matrix.m`, and reside in `%DSPACE_ROOT%\matlab\mlib`. They can be called from the MATLAB Command Window or embedded in other M files.

Refer to:

- *mlib_1dlookup* on page 80
- *mlib_2dlookup* on page 83
- *mlib_matrix* on page 86

Built-in Simulink Blocks

Matrices are written out row-wise (C-style). The index of the (p, q) element of an (m, n) matrix is thus (p-1)·n+q. In linear time-invariant (LTI) systems, optimization algorithms omit matrix elements equal to 0 or 1 in the generated code. Transfer functions must be transformed into state space systems before code can be generated, so polynomial coefficients (or poles and zeros) cannot be represented directly.

2-D Look-Up Tables

The following rules apply when parameters of 2-D look-up tables are stored:

The n-vector elements are stored first, followed by the m-elements of the y-axis vector.

The elements of the (m,n)-table data matrix are stored

- Row-wise up to MATLAB 5.3.x (R11)
- Column-wise since MATLAB 6.0 (R12)

S-Functions

Matrices are written out column-wise (MATLAB or Fortran-style). The index of the (p, q) element of an (m, n) matrix is thus (q-1)·m+p.

For more information, refer to the Real-Time Workshop's Guide by The Mathworks.

MLIB/MTRACE always writes data vectors to the associated real-time memory column-wise. For parameter matrices of built-in Simulink blocks, the MATLAB transpose operator can be used to make sure that matrices are written in the right order.

The following code shows how the parameters that represent state space matrices of the Simulink compensator, observer or State Space block can be modified online by using MLIB/MTRACE.

Example

This example works only if the matrices do not contain zeros or ones, and if the sizes of the matrices equal the sizes of the matrices in the Simulink block that the code was generated from.

```
...
% Design the estimator (as an example for a LTI system).
% a, b, c, d are the resulting state space matrices.
[a,b, c, d] = my_estimator_design_routine;
% Select and lock DSP board
mlib('SelectBoard', 'DS1005');
mlib('LockProgram');
% Obtain addresses
var_names = { ...
      ['Model Root/compensator/estimator/' ...
        'State Space/A']; ...
      ['Model Root/compensator/estimator/ ...
        'State Space/B']; ...
      ['Model Root/compensator/estimator/ ...
        'State Space/C']; ...
      ['Model Root/compensator/estimator/ ...
        'State Space/D']; ...
   };
[a_var, b_var, c_var, d_var] = mlib('GetTrcVar', var_names);
% the GetTrcVar function already obtains the number of elements
% in each of specified state space
% Write down the new estimator matrices. Use the transpose
% operator to make sure the matrix elements are stored in the
% right order.
data_to_writing = [a'; b'; c'; d'];
mlib('Write', [a_var; b_var; c_var; d_var], ...
   'Data', data_to_writing);
```

If zeros or ones are involved, omitted elements must also be taken care of, and the code will not be as simple. An easy way to avoid zeros and ones is to add eps (machine precision) to the matrices before generating code with RTI, though this method involves additional computational loads and may change the system dynamics slightly.

mlib_1dlookup

General syntax

`output = mlib_1dlookup(Command, ParameterList)`

Description

Depending on the selected command, `mlib_1dlookup` obtains the descriptors of the vectors of input and output values (input vector and output vector) of the specified 1-D look-up table, and performs Read/Write operations.

Init

`myTable = mlib_1dlookup('Init',lutName [,length][,procName])`

The descriptors of the 1-D look-up table parameters, input vector and output vector, are obtained and returned in the `myTable` structure.

The `myTable` structure contains the following fields:

Field	Description
length	Length of the 1-D table vectors
inputDesc	Descriptor of the input vector
outputDesc	Descriptor of the output vector
name	1-D table name, set by lutName
cpu	Processor name, if specified, otherwise an empty string

The syntax for the `lutName` parameter is the same as for the Simulink block parameters, whose addresses are to be obtained with the GetTrcVar function `Model Root/<subsystem>/<blockName>,` where `<subsystem>` is the path name of the Simulink system containing the block called `<blockName>`, but with the model name omitted.

- Use the trcview utility to get the correct `lutName`.
- 'Length' is only supported for downward compatibility with previous versions of this tool. It is not necessary to specify it.

If this function is called in multiprocessor mode, you must specify the processor name.

ReadInput

```
data = mlib_1dlookup('ReadInput',myTable [,indices])
```

This call reads the values of the input vector. If the `indices` vector parameter is specified only the values of the elements that it specifies are read; otherwise, the whole vector is returned.

`myTable` must be the structure returned by the `Init` function.

ReadOutput

```
data = mlib_1dlookup('ReadOutput',myTable [,indices])
```

This call reads the values of the output vector. If the `indices` vector parameter is specified only the values of the elements that it specifies are read; otherwise, the whole vector is returned.

`myTable` must be the structure returned by the `Init` function.

WriteInput

```
mlib_1dlookup('WriteInput',myTable, [,indices], data)
```

This call writes data to the vector of input values. If the `indices` vector parameter is specified data is written only to the elements that it specifies; otherwise to the whole input vector.

`myTable` must be the structure returned by the `Init` function. The `data` parameter must be a vector containing the data to be written.

The number of elements in `data` must match the number of elements to be written.

WriteOutput

```
mlib_1dlookup('WriteOutput', myTable [,indices], data)
```

This call writes data to the vector of output values. If the `indices` vector parameter is specified data is written only to the elements that it specifies; otherwise, to the whole output vector.

`myTable` must be the structure returned by the Init function. The `data` parameter must be a vector containing the data to be written.

The number of elements in `data` must match the number of elements to be written.

Example

The 1-D look-up table block named **1DTable** resides in the Simulink subsystem **Linearization**. The number of elements is 11.

```
% Init the mlib_1dlookup function
myTable = mlib_1dlookup('Init',...
   'Model Root/Linearization/1DTable' );
% Read all elements of the vector of input values - the
% indices' specification is superfluous
inputData = mlib_1dlookup('ReadInput', myTable);
% Read the elements 2, 3 and 4 of the vector of output values
outputData = mlib_1dlookup('ReadOutput', myTable, [2:4])
% Write the new data to the area read before
newOutputData = [1 2 3];
mlib_1dlookup('WriteOutput', myTable, [2:4], newTableData);
```

mlib_2dlookup

General syntax

```
output = mlib_2dlookup(Command, ParameterList)
```

Description

Depending on the selected command, `mlib_2dlookup` obtains the descriptors of the row vector, column vector and table of the specified 2-D look-up table, and performs Read/Write operations.

Init

```
myTable = mlib_2dlookup('Init',lutName [,rows,cols]...
   [,procName])
```

The descriptors of the 2-D look-up table parameters row vector, column vector and table values are obtained and returned in the `myTable` structure.

The `myTable` structure contains the following fields:

Field	Description
rows	Number of rows in the 2-D table
cols	Number of columns in the 2-D table
rowDesc	Descriptor of the row vector
colDesc	Descriptor of the column vector
tableDesc	Descriptor of the table values
name	2-D table name, set by `lutName`
cpu	Processor name, if specified, otherwise empty string

The syntax for the `lutName` parameter is the same as for the Simulink block parameters whose addresses are to be obtained with the GetTrcVar function `Model Root/<subsystem>/<blockName>,` where `<subsystem>` is the path name of the Simulink system containing the block named `<blockName>`, but with the model name omitted.

- Use the trcview utility to get the correct `lutName`.
- 'Rows' and 'cols' are only supported for downward compatibility with previous versions of this tool. It is not necessary to specify them.

If this function is called in multiprocessor mode, you must specify the processor name.

ReadRow	`data = mlib_2dlookup('ReadRow',myTable [,indices])` This call reads the values of the row vector. If the `indices` vector parameter is specified, only the values of the elements that it specifies are read; otherwise, the whole vector is returned. `myTable` must be the structure returned by the `Init` function.
ReadCol	`data = mlib_2dlookup('ReadCol',myTable [,indices])` This call reads the values of the column vector. If the `indices` vector parameter is specified, only values of the elements that it specifies are read; otherwise, the whole vector is returned. `myTable` must be the structure returned by the `Init` function.
ReadTable	`data = mlib_2dlookup('ReadTable', myTable [,rows, cols])` This call reads the table values. If the vector parameters `rows` and `cols` are specified, only the rows and columns that they specify are read; otherwise, the whole table is returned. `myTable` must be the structure returned by the `Init` function.
WriteRow	`mlib_2dlookup('WriteRow',myTable, [,indices], data)` This call writes data to the row vector. If the vector `indices` parameter is specified, data is only written to the elements that it specifies; otherwise, to the whole row vector. `myTable` must be the structure returned by the `Init` function. The `data` parameter must be a vector containing the data to be written. The number of elements in `data` must match the number of elements to be written.
WriteCol	`mlib_2dlookup('WriteCol', myTable [,indices], data)` This call writes data to the column vector. If the `indices` vector parameter is specified, data is written only to the elements that it specifies; otherwise, to the whole column vector.

`myTable` must be the structure returned by the `Init` function. The `data` parameter must be a vector containing the data to be written.

The number of elements in `data` must match the number of elements to be written.

WriteTable

`mlib_2dlookup('WriteTable', myTable, [,rows, cols], data)`

This call writes the table data. If the vector parameters `rows` and `cols` are specified, data is written to the rows and columns that they specify; otherwise to the whole table.

`myTable` must be the structure returned by the `Init` function. The `data` parameter must be a matrix containing the data to be written.

The size of the `data` matrix must match the lengths of `rows` and `cols` vectors, or the size of the whole table.

Example

The 2-D look-up table block named **2DTable** resides in the Simulink subsystem **Linearization**. The number of rows is 3 and the number of columns is 5.

```
% Init the mlib_2dlookup function
myTable = mlib_2dlookup('Init',...
  'Model Root/Linearization/2DTable' );
% Read all elements of the row vector - the indices'
% specification is superfluous
rowData = mlib_2dlookup('ReadRow', myTable);
% Read the elemnts 2,3 and 4 of the column vector
colData = mlib_2dlookup('ReadCol', myTable, [2:4])
% Read the table values: columns 1-3 from rows 2 and 3.
% The dimension of the output data matrix will be 2 x 3.
tableData = mlib_2dlookup('ReadTable',myTable, [2:3], [1:3]);
% Write the new data to the area read before
newTableData = [1 2 3; 5 6 7];
mlib_2dlookup('WriteTable', myTable, [2:3], [1:3],...
  newTableData);
```

mlib_matrix

General syntax

```
output = mlib_matrix(Command, ParameterList)
```

Description

Depending on the selected command, `mlib_matrix` obtains the descriptor of the specified matrix and performs Read/Write operations.

Init

```
myMatrix = mlib_matrix('Init',matName, rows, cols,...
   alignment [,procName])
```

The matrix descriptor is obtained and returned in the `myMatrix` structure.

The `myMatrix` structure contains the following fields:

Field	Description
rows	Number of matrix rows, set by `rows` parameter
cols	Number of matrix cols, set by `cols` parameter
matDesc	Matrix descriptor
name	Name, set by matName
alignment	Describes how the matrix elements are stored in real-time memory; possible values: `Row-Wise`, `Col-Wise`
cpu	Processor name, if specified, otherwise empty string

The name of the first matrix element must be specified as the `matName` parameter. Apply the same syntax as for the Simulink block vectorized parameters, whose addresses are to be obtained with the GetTrcVar function `Model Root/<subsystem>/<blockName>/<matrixParameter>`, where `<subsystem>` is the path name of the Simulink system containing the block named `<blockName>`, but with the model name omitted.

Use the trcview utility to get the correct syntax of the `matName`.

If this function is called in multiprocessor mode, you must specify the processor name.

Read

```
data = mlib_matrix('Read', myMatrix [,rows, cols])
```

This call reads the matrix values. If the vector parameters `rows` and `cols` are specified, only the rows and columns that they specify are read; otherwise the whole matrix is returned.

`myMatrix`must be the structure returned by the `Init` function.

Write

```
mlib_matrix('Write', myMatrix, [,rows, cols], data)
```

This call writes the data matrix. If the vector parameters `rows` and `cols` are specified, the data is written to the rows and columns that they specify; otherwise to the whole matrix.

`myMatrix`must be the structure returned by the `Init` function. The `data` parameter must be a matrix containing the data to be written.

The size of the data matrix must match the lengths of `rows` and `cols` vectors, or the size of the whole matrix.

Example

The State Space block **State-Space** resides in the Simulink subsystem **Controller**. The dimension is 3 x 3.

```
% Init the mlib_matrix function for A matrix
myMatrix = mlib_matrix('Init',...
   'Model Root/Controller/State-Space/A',...
   3, 3, 'Row-Wise');
% Read the values of the A matrix.
matrixData = mlib_matrix('Read',myMatrix);
% Write the new data to the following area: rows 2,3,
% columns 1:2
newData = [1 2; -0.5 1];
mlib_matrix('Write', myMatrix, [2:3], [1:2], newData);
```

MLIB/MTRACE Programming Examples

For a demonstration of MLIB/MTRACE programming, refer to:

- *MLIB/MTRACE with Handcoded Applications* on page 88
- *MLIB/MTRACE with dSPACE RTI* on page 92
- *Additional MLIB/MTRACE Programming Examples* on page 94
- *MLIB/MTRACE with dSPACE RTI-MP* on page 95

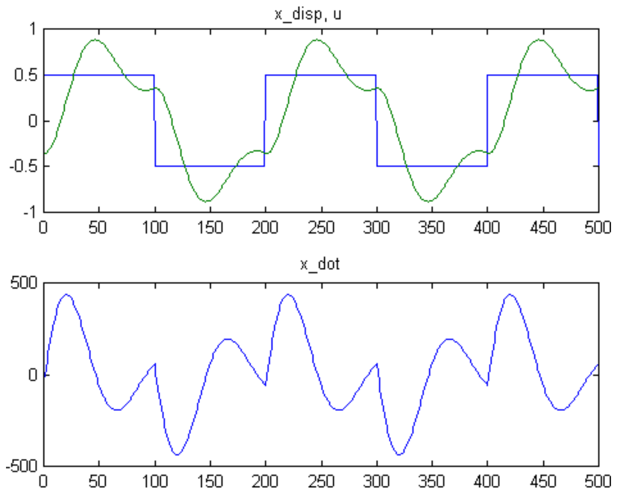
MLIB/MTRACE with Handcoded Applications

The following M file (`simple_xxxx.m`) is an example of a program that demonstrates MLIB/MTRACE capture capabilities with a handcoded application. It is located in `%DSPACE_ROOT%\demos\dsxxxx\mlib`.

```
% Syntax: simple_1005;
% Purpose: Demonstrates MLIB/MTRACE on a DS1005
% This file needs the demo application
% '%DSPACE_ROOT%\demos\ds1005\GettingStarted\HandCode\
% smd_1005_hc.obj' running.
%
% The input signal 'x_disp', and the states 'x' and 'x_dot' of
% a second order lag system are traced. Data is plotted in a
% figure window.
% Copyright (c) 1996-1999 by dSPACE GmbH
% select processor board 'ds1005'
mlib('SelectBoard','DS1005');
% select variables to be traced and obtain their descriptors
var_names = {'Model/x_disp';...
  'Model/x';...
  'Model/x_dot'...};
var = mlib('GetTrcVar',var_names)
% set the options of data acquisition performed by service
% number 1 (default)
mlib('Set','Trigger','ON', ...
   'TriggerLevel',0, ...         % default, can be omitted
   'TriggerEdge','rising', ...    % default, can be omitted
   'TriggerVariable', var(1), ... % trigger on variable 'Model/x_disp'
   'TraceVars', var, ...
   'NumSamples', 500, ...
   'Delay',0);                   % default, can be omitted
% start capture on DS1005
mlib('StartCapture');
% wait until capture is done
while mlib('CaptureState')~=0,end
% fetch after capture is complete
out_data = mlib('FetchData');
```

```
% plot results
clf;
subplot(2,1,1);
plot(out_data(1:2,:)');
title('x\_disp, x');
subplot(2,1,2);
plot(out_data(3,:)');
title('x\_dot');
set(gcf,'NumberTitle','off');
set(gcf,'Name','A Simple MLIB/MTRACE Example');
figure(gcf);
```

Running `simple_XXXX.m` should result in plots as shown below:

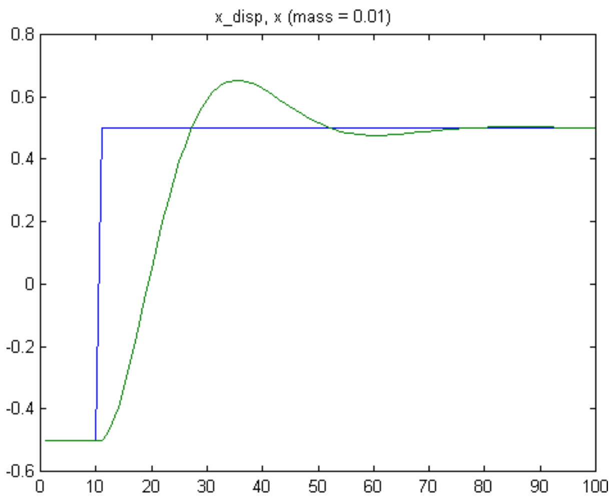


MLIB/MTRACE can be used for real-time data acquisition. It also provides asynchronous reading and writing of parameters in the real-time application. The example file `chmass_xxxx.m` demonstrates the use of both features. It is contained in `%DSPACE_ROOT%\demos\dsxxxx\mlib`.

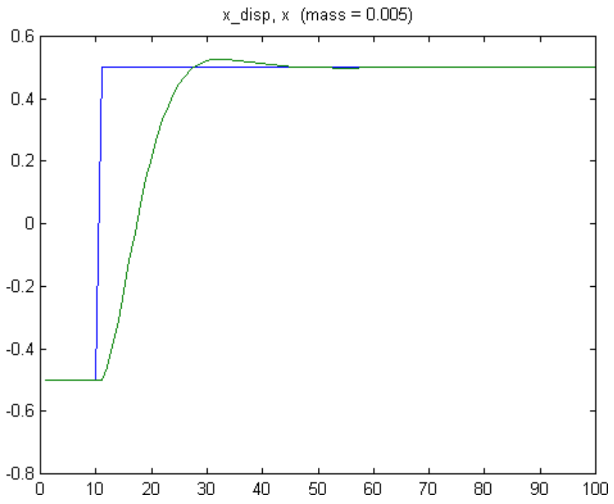
```
function chmass_1005(mass);
% Syntax: chmass_1005(mass)
% Purpose: Demonstrates MLIB/MTRACE on a DS1005
% This file needs the demo application
%'DSPACE_ROOT\demos\ds1005\GettingStarted\HandCode\smd_1005_hc.obj' running.
% The input signal 'x_disp', and the state 'x' of a second
% order lag system are traced.
% Data is plotted in a figure window.
% The chmass_1005.m demo is augmented by MLIB/MTRACE
% functions, which  are used to change the mass 'm'
% of the spring-mass-damper system.
% Copyright (c) 1996-2000 by dSPACE GmbH
if margin ~= 1,
   mass=[0.002 0.005 0.01 0.02];
end
```

```
if mass<0.001 | mass>0.1,
   error('*** mass must be in the range 0.001 < mass < 0.1')
end
% select DS1005 board for use with MLIB/MTRACE
mlib('SelectBoard', 'DS1005');
% select the variable used by MLIB/MTRACE and get their
% descriptors
variables = {'Model/Model Parameters/m';...
   'Model/x_disp';...
   'Model/x'};
[m_desc, x_disp_desc, x_desc] = mlib('GetTrcVar', ...
   variables)
% set the option for data acquisition performed
% by service number 1 (default)
mlib('Set', 'Trigger', 'ON', ...
   'TriggerLevel', 0, ...       % default, can be omitted
   'TriggerEdge', 'rising', ... % default, can be omitted
   'TriggerVariable', x_disp_desc, ...
   'TraceVars', [x_disp_desc; x_desc], ...
   'NumSamples', 100, ...
   'Delay', -10, ...
   'DownSampling', 1);          % default, can be omitted
for m = mass,
   % lock program during capture
   mlib('LockProgram');
   % write new mass value to DS1005 via MLIB/MTRACE
   mlib('Write',    m_desc, 'Data', m);
   % start capture and wait until it is complete
   mlib('StartCapture');
   while mlib('CaptureState')~=0,end
   % transfer data when capture is complete and release lock
   out_data = mlib('FetchData');
   mlib('UnlockProgram');
   % plot results
   clf;
   plot('out_data');
   title(['x_disp, x  (mass=' num2str(m) ')']);
   set(gcf,'NumberTitle', 'off');
   set(gcf,'Name', 'MLIB/MTRACE');
   figure(gcf);
   if (m ~= mass(end)),
      fprintf ('-- press any key --\n');
      pause;
   end;
end
mclear;
```

The following plot contains the results of the `chmass_XXXX` sample M file.



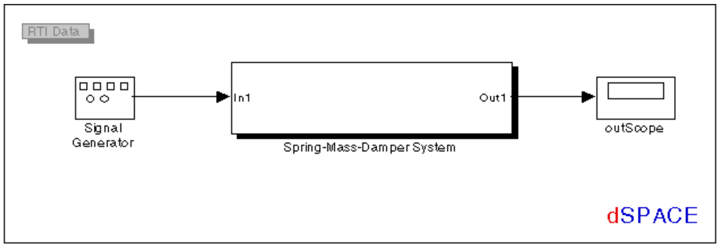
`(chmass_XXXX(0.01))`



`(chmass_XXXX(0.005))`

MLIB/MTRACE with dSPACE RTI

The following examples refer to the demo program for PPC boards `smdtf_XXXX_sl.ppc` located in `%DSPACE_ROOT%\demos\dsXXXX\mlib`. Although `smdtf_xxxx_sl.ppc` was generated from a Simulink model using dSPACE's Real-Time Interface (RTI) to MATLAB/Simulink, neither Simulink nor RTI is needed to run the MLIB/MTRACE sample programs. You can use ControlDesk's Platform Manager to download `smdtf_xxxx_sl.ppc` to the desired processor board.



The Simulink model `smdtf_XXXX_sl` shown above represents the transfer function of a spring-mass-damper system. A signal generator is part of `smdtf_XXXX_sl`. The following M file, `siggen_XXXX.m`, is used to display and change the signal generator parameters online.

Example

This example refers to a DS1005 board.

```
function siggen_1005 (amplitude,period);
% ----------------------------------------------------------
% function siggen_1005 (amplitude, period);
% SIGGEN_1005() sets the parameters (amplitude and period)of a
% signal generator using the dSPACE MATLAB Interface Library.
% The signal generator is part of the application
% 'smdtf_1005_sl'.
% It is assumed that the unit of the parameter 'frequency' of
% the signal generator is 'Hertz'.
% amplitude     peak value of the signal
% period        period length of the signal
%               (not used for noise signals)
% When called without input parameters, SIGGEN_1005 simply
% displays the current settings of the signal generator.
% -----------------------------------------------------------
% Copyright (c) 1996-2000 by dSPACE GmbH
% Initialize dSPACE MLIB/MTRACE
mlib('SelectBoard', 'DS1005');
% Get descriptors for signal generator parameters
variables = {...
      'Model Root/Signal\nGenerator/Amplitude'; ...
      'Model Root/Signal\nGenerator/Frequency' ...
   };
[amplit_var, freq_var] = mlib('GetTrcVar', variables);
```

```
if margin = 0,
   % Print out command syntax
   fprintf ('\n Usage: siggen(amplitude, period)\n\n');
   % Print out signal generator parameters
   [amplit,freq] = mlib('Read', [amplit_var; freq_var]);
   frpintf ('Signal generator parameters:' ...
      'Amplitude=%g, Period=%g sec\n'], amplit,1/freq');
   return;
end
% Set period and amplitude
values = {1/period; amplitude};
mlib('Write', [freq_var; amplit_var], 'Data', values);
```

The M file `ch_damp_XXXX.m` is used to modify the damping factor of `smdtf_XXXX_sl`. If you type **RETURN** at the input prompt of the `ch_damp_XXXX` function to accept the defaults, the damping factor will be changed from 0.5 to 1.0, in steps of 0.05 performed every second. The results can be displayed easily with ControlDesk instruments.

Example

```
% ------------------------------------------------------------
%  Modify the damping factor of the system 'smdtf_1005_sl' at
%  certain time steps
%
%  Transfer function of the spring-mass-damper system
%
%                  omega*omega                      b0
% G(s) = ---------------------------------- = --------------
%       s*s + 2*damp*omega s + omega*omega   s*s + a1 s + a0
%
%  Parameters stored in C Code:
%     P:Spring-Mass-Damper System.A(1)      -a1
%     P:Spring-Mass-Damper System.A(2)      -a0
%     P:Spring-Mass-Damper System.C(2)       b0
%
%  Before invoking this M file the real-time processor
%  application smdtf_1005_sl.ppc must be
%  loaded to the DS1005 processor board.
%
%  !! To view the results please start ControlDesk experiment
%  file smdtf_1005_sl.cdx
% ------------------------------------------------------------
% Initialize dSPACE MLIB
mlib('SelectBoard','DS1005');
% check if the application smdtf_1005_sl.ppc is running
DemoApplName = lower([pwd '\smdtf_1005_sl.ppc']);
if mlib('IsApplRunning'),
   ApplInfo = mlib('GetApplInfo');
   if strcmp(DemoApplName,lower(ApplInfo.name)) ~= 1,
      err_msg = sprintf(['*** This MLIB demo file needs ',...
         'the real-time processor application\n***',...
         ' ''%s'' running!'], DemoApplName);
      error(err_msg);
   end
else
   err_msg = sprintf(['*** This MLIB demo file needs ',...
      'the real-time processor application\n***',...
      ' ''%s'' running!'], DemoApplName);
   error(err_msg);
end;
```

```
% Get descriptors for the state space matrices
matrices = {'Model Root/Spring-Mass-Damper System/A';...
'Model Root/Spring-Mass-Damper System/C'};
[Amatrix_desc, Cmatrix_desc] = mlib('GetTrcVar',matrices);
% Read current parameter values from real-time processor
Amatrix = mlib('Read',Amatrix_desc);
Cmatrix = mlib('Read',Cmatrix_desc);
a1 = -Amatrix(1);
a0 = -Amatrix(2);
b0 = Cmatrix(2);
OMEGA = sqrt(a0);      % bandwidth
DAMP = a1/(2*OMEGA);   % damping factor
fprintf ('\n %8.1e \n',b0);
fprintf (' G(s) = -------------------------\n',b0);
fprintf (' s*s %+8.1e s %+8.1e\n\n',a1,a0);
fprintf (' Bandwidth: %6.0f\n',OMEGA);
fprintf (' Damping factor: %6.2f\n',DAMP);
% Input vector of new damping factors
damp_values = ...
   input(' New damping factor(s) [0.5:0.05:1.0] ?');
if isempty(damp_values), damp_values = 0.5:0.05:1; end;
% Input times when parameters are updated
update_time = input(' Update interval 1 sec] ? ');
if isempty(update_time), update_time=1; end;
fprintf (' ---------------------------------------------\n');
% Start MATLAB stopwatch timer (min. timer resolution 54 ms)
TIC; T=0;
for damp=damp_values,
   % Compute transfer function parameters
   % of spring-mass-damper system
   amatrix = [ -2*damp*OMEGA -OMEGA*OMEGA ]; % [ -a1 -a0 ]
   % Write new transfer function parameters
   % to real-time processor
   mlib('Write',Amatrix_desc,'Data', amatrix);
   fprintf ('T=%5.2f sec. Damping factor is now %5.2f,...\n', ...
            T, damp);
   % Let background processes become active while waiting
   T = T + update_time;
   while (TOC < T), end;
end
fprintf (' ---------------------------------------------\n');
% Reset gain to original value
mlib('Write',Amatrix_desc,'Data',Amatrix);
fprintf (' T=%5.2f sec. Damping factor reset to %5.2f\n',... T,DAMP);
```

Additional MLIB/MTRACE Programming Examples

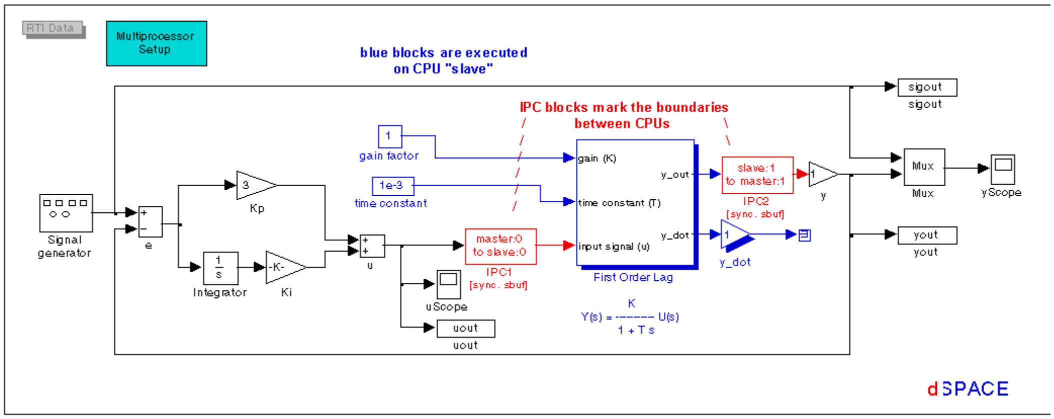
There are additional MLIB/MTRACE programming examples in the MLIB/MTRACE demo directories:

- The M file`exectime_XXXX.m` displays the execution time of the current application.
- `chkerr_XXXX.m`reads the error flag of the real-time program and prints the error code.

- `trigger_xxxx.m`shows how to start the data capture using the trigger signal and how to cause the trigger event by means of MLIB/MTRACE.
- Another M file, `cont_XXXX.m`, demonstrates the use of MLIB/MTRACE in continuous mode.
- For an interrupt-driven A/D conversion, look at `adc_XXXX.m`. You can find the corresponding C-sources in `adc_XXXX_hc.c`

MLIB/MTRACE with dSPACE RTI-MP

The following illustration shows the Simulink model `pipt1_dual1005_sl`, which can be used to generate the multiprocessor application with dSPACE RTI-MP.



The following example, `mlib_dual1005.m`, demonstrates how to use MLIB/MTRACE to change a parameter of the application and displays the results:

Example

```
% mlib_dual1005.m - This is an example M file for
% demonstrating the multiprocessing feature of MLIB.
%
% Before invoking this M file the application
% 'pipt1_dual1005_sl' must be build with RTI-MP and downloaded
% to the dual DS1005 MP-System.
%
% The period of the signal generator is changed by MLIB and
% the result is plotted in a MATLAB figure window
% Copyright (c) 2000 by dSPACE GmbH
%
% $Workfile: mlib_dual1005.m $ $Revision: 1 $
% $Date: 10.06.03 15:45 $
% $Archive: /sw/MATLAB/MLIB and MTRACE/Installation/MLIB_4.4.7
% /demos/ds1005MP/dual1005/mlib/mlib_dual1005.m $

% Initialize dSPACE MLIB
mlib('SelectBoard','ds1005');    % DS1005 multiprocessor
                                 % applications are based
                                 % on DS1005 boards

% select MP-application configuration file
sdfFile = [pwd '\pipt1_dual1005_sl.sdf'];
mlib('MultiProcessing',sdfFile);

% get descriptors for variables residing on processor 'MASTER'% and 'SLAVE'
master_var = {'Model Root/Signal\ngenerator/Amplitude';...
      'Model Root/Signal\ngenerator/Frequency';...
      'Model Root/Signal\ngenerator/Out1'};
slave_var  = 'Model Root/First Order Lag/y/Out1';

[ampl, freq, siggen] = mlib('GetTrcVar', master_var, 'CPU',...
 'MASTER');
sys_out              = mlib('GetTrcVar', slave_var,  'CPU',...
 'SLAVE');

% read the current values of amplitude and frequency
[org_ampl_val, org_freq_val] = mlib('Read', [ampl;freq]);

% set the option of the data acquisition performed by service
% group HostService
mlib('Set','Trigger','ON',...
   'TriggerLevel',0,...        % default, can be omitted
   'TriggerEdge','rising',...  % default, can be omitted
   'TriggerVariable',siggen,...
   'TraceVars',[sys_out;siggen],...
   'NumSamples',400,...
   'Delay',-3);

% set the amplitude of the Signal Generator to 0.5
mlib('Write',ampl,'Data',0.5);

clf;
ah = 0;
for per = [0.002:0.002:0.02],
  mlib('Write',freq,'Data',1/per);    % change the period of
                                      % the signal genarator
  mlib('StartCapture');               % start capture on the
                                      % DS1005, default
                                      % service number 1
  while mlib('CaptureState')~=0,end   % wait until data
                                      % acquisition finished

  out_data = mlib('FetchData');
  if ah == 0,
     % in DS1005-MP mode time stamps are always collected
     t = out_data(1,:);
     axis([t(1) t(end) -0.9 0.9]);
     ah = gca;
     l1 = line(t,out_data(2,:)');
     l2 = line(t,out_data(3,:)');
     set(l1,'Color','r');
     grid;
  else
     set(l1,'YData',out_data(2,:),'EraseMode','Xor');
     set(l2,'YData',out_data(3,:),'EraseMode','Xor');
      end
      figure(gcf);
      drawnow;
      pause(1);
    end
    % reset the amplitude and frequency
    mlib('Write',[ampl;freq],'Data',{org_ampl_val;org_freq_val});
    % return to single processor mode
    mlib('MultiProcessing','OFF');
```

File Reference

MAT File Generated by MLIB/MTRACE

When the StreamToDisk option is enabled, the data collected in continuous mode is stored in a MAT file on the hard disk. Even if the format of the collected data has been set to 'structure', it is stored as a matrix. To avoid large matrices, which are difficult to manage in the MATLAB workspace, MLIB/MTRACE splits the data in the MAT file into multiple matrices for long data captures (2 MB at the most).

If the data has been collected in SP mode, the resulting matrices are named `rt_data`, `rt_data_2`, etc.

If the data has been collected in MP1005 or MP1006 mode, the resulting matrices are named `<processorName>_rt_data`, `<processorName>_rt_data_1`, `<processorName>_rt_data_2`, etc.

The whole MAT file and the desired matrices can be loaded in the MATLAB workspace with the standard MATLAB load function. Any selected portion of data can be loaded with the MLIB/MTRACE `LoadFile` function.

In addition to the data matrices, the structures

- `experiment_info` for SP
- `<processor_name>_experiment_info` for MP1005 or MP1006

are also stored.

This structure contains fields providing information about the performed data acquisition:

Field	Information
.date	Date when the MAT file was created (date of the real-time experiment)
.step_size	Step size with which the data was collected
timestamping	■ 'ON': Time stamps at which data was collected are also traced. In this case MAT file matrices are matrices with time ■ 'OFF': Time stamps are not traced
.downsampling	Downsampling factor
.applications	Cell array of the name(s) of the application(s) running while data acquisition was performed
.variables	Cell array of names of the traced variables

Utilities

Basics on the TrcView Utility

Objective

The utility `%DSPACE_ROOT%\exe\TrcView.exe` provides a user interface to TRC files (generated or user-written). By using this tool you can enter variable names more conveniently and without type mismatches in the GetTrcVar function

Possible problems when entering variable names

Descriptors for global variables in a real-time application generated with RTI can be obtained with the GetTrcVar function. Every time a real-time application is built, RTI generates a TRC file that maps Simulink variable names to the corresponding global variables in the generated code. Thus variables can be specified with the names of Simulink blocks whose outputs, parameters, states, or state derivatives they represent, or with the labels of the corresponding signal lines. For more information about the syntax used when a variable is specified with GetTrcVar, refer to the description *GetTrcVar* on page 45.

The names of variables defined in a TRC file cannot always exactly match the names of the associated Simulink blocks, their parameters or signal labels. There are some cases when there are deviations; for example, if block or subsystem names contain carriage returns or slashes, or if identical signal labels exist. Moreover, it is not very convenient to enter lengthy strings manually, especially if the Simulink system features numerous hierarchies with long subsystem names. Because descriptor evaluation with Simulink variable names is based on string comparison, a call to GetTrcVar always results in an error if only one character in a variable specification does not comply.

How to Use TrcView

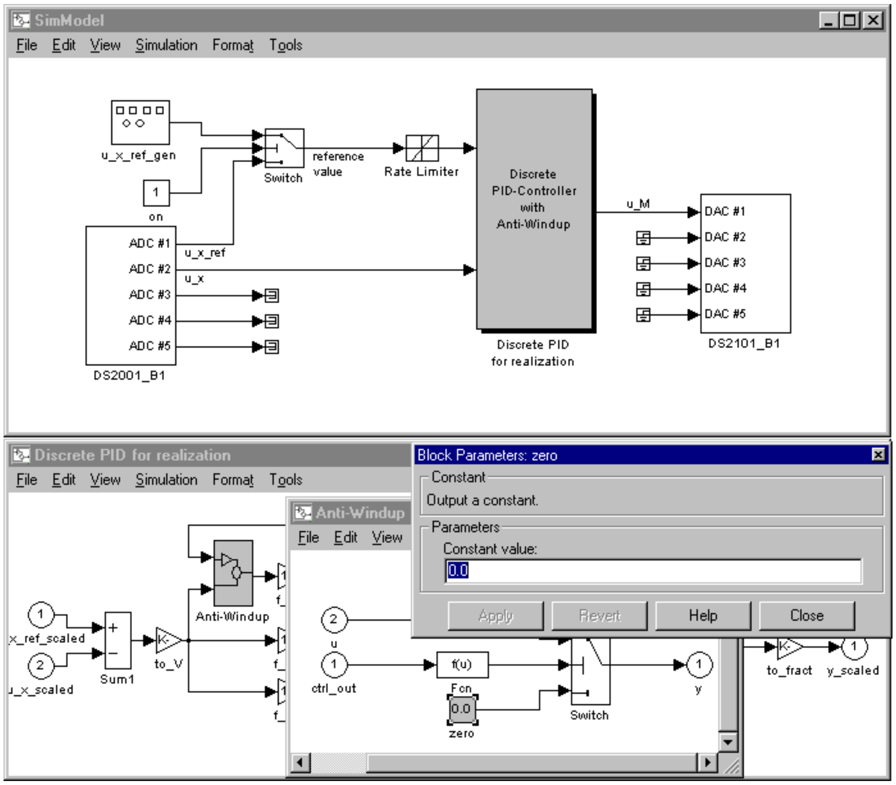
Method 1

To use TrcView

1. Enter the **`trcview`** command in the MATLAB Command Window or call the program directly from a DOS window. Then open the desired TRC file. A model browser and a variable list make it easy to search for a specific variable.
2. Double-click in the list to paste the full variable name, with the syntax required by GetTrcVar, into an edit field.
3. From the edit field, copy and paste the variable name into any M file editor or the MATLAB Command Window with the standard shortcut keys **`Ctrl-C`**, **`Ctrl-V`**

The illustration below shows the Simulink system SimModel. Suppose that the `Constant Value` parameter of the Simulink block Discrete PID for realization/Anti-Windup/zero is to be changed with MLIB/MTRACE.

The name of the subsystem contains a carriage return.

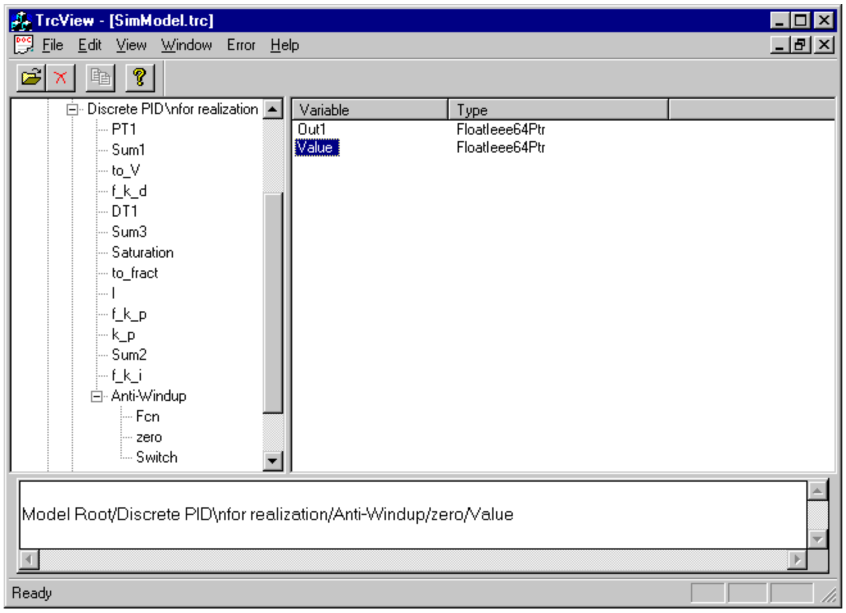


Constant
Output a constant.
Parameters
Constant value:
0.0

For example, to obtain a descriptor for the parameter `Constant Value`, the GetTrcVar function call is

```
mlib('GetTrcVar', ...
   ['Model Root/Discrete PID\nfor realization/'Anti-Windup/'...
   'zero.Value'];
```

The following illustration shows the TrcView window after the associated TRC file `SimModel.trc` has been opened. It shows a search for the Constant Value parameter of the Simulink block Discrete PID for realization/Anti-Windup/zero.



Variable	Type
Out1	FloatIeee64Ptr
Value	FloatIeee64Ptr

Model Root/Discrete PID\nfor realization/Anti-Windup/zero/Value

Method 2

To copy and paste a variable name to the M file-editor of the MATLAB Command Window

1 In the hierarchy browser on the left, click the node of the desired subsystem to display its available variables in the list box on the right.

2 Double-click the desired variable name to paste it, with the syntax required by GetTrcVar, to the edit field at the bottom of the window.

3 From here, copy and paste the variable name to any M file editor or the MATLAB Command Window.

If a TRC file contains syntax errors, attempts to open it with TrcView will fail, and the trace error file will be displayed instead. The error positions are marked and appropriate messages are also output.

For detailed information about TRC file syntax, refer to *Syntax of the TRC File* (*ControlDesk Reference*).

Limitations

For information about MLIB/MTRACE limitations, refer to the topics below:

- *Accessing Variables with Read/Write Functions* on page 108
- *Number of Applications Simultaneously Accessing the Hardware* on page 108

Accessing Variables with Read/Write Functions

Variables located in the local SRAM memory on the DS1103 and DS1401, and the cached memory on DS1104, DS1005 and DS1006 boards, can be accessed by MLIB/MTRACE Read/Write functions via the host interface and the special service, master_cmd_server. This service must run in the background routine of the application program.

Number of Applications Simultaneously Accessing the Hardware

The number of applications that simultaneously perform real-time data acquisition on PPC and X86 boards is limited to eight. ControlDesk can represent more than one application provided multithreading is used in the automation scripts. For example, while instrumentation is in animation mode, automation scripts can access the real-time hardware via the RTPLib.

Troubleshooting

MLIB/MTRACE Error Messages

This section shows MLIB/MTRACE's error messages in numerical order and offers solutions where applicable.

MLIB 1

No dSPACE processor board selected. A dSPACE processor board registered in ControlDesk's Platform Manager must be selected before any MLIB/MTRACE function other than SelectBoard and GetBoardInfo can be performed.

1. Use **`mlib('`**SelectBoard**`',<board>)`** to select a processor board.

MLIB 2

MLIB function <name> not implemented. The first parameter of `mlib()` call does not specify an implemented MLIB/MTRACE function.

1. Type **`help mlib`** in the MATLAB Command Window to get information about the implemented functions.

MLIB 3

Improper number of input arguments. The number of input arguments does not match the number of arguments needed by the specified function.

1. Type **`help <function_name>`** in the MATLAB Command Window to get information about the function function_name.

MLIB 4

RESERVED

MLIB 5

Boards of type <boardType> cannot be selected Only dSPACE processor boards DS1103, DS1104, DS1401, DS1005 and DS1006 can be selected for direct access with MLIB. Other dSPACE boards cannot be accessed at all.

MLIB 6

RESERVED

MLIB 7

Variable descriptor expected. A variable descriptor as returned by one of the address evaluation functions or created by SetVarProperty is required as an argument of the MLIB/MTRACE function.

1. Type `help <function_name>` in the MATLAB Command Window to get information about the function function_name that caused this error message.

See *Naming Conventions* on page 30 for information on variable descriptors.

MLIB 8

Argument 'count' is of improper type. The specified 'count' parameter must be scalar.

MLIB 9

Number of output arguments must agree with the size of input argument. The address evaluation functions GetTrcVar, GetMapVar and the Read function allow a vectorized input parameter. If more than one output parameter is specified, their number must match the size (number of rows) of the input vector.

MLIB 10

Input argument must be of string type. An input argument of non-string type (type: double, struct, cell, etc.) was specified.

1. Type `help <function_name>` in the MATLAB Command Window to get information about the respective MLIB/MTRACE function.

MLIB 11

Syntax error in variable name specification. The syntax of a variable name is not valid.

1. Refer to the syntax of the appropriate address evaluation function shown in *MLIB/MTRACE Function Reference* on page 31.

MLIB 12

RESERVED

MLIB 13

RESERVED

MLIB 14

Input argument is of improper type or size.

1. Type **`help <function name>`** in the MATLAB Command Window to get information about the function <function name> that issued this error message.

MLIB 15

RESERVED

MLIB 16

Only <maxnumber> variables allowed. The number of traceable variables that can be accessed directly is limited to <maxnumber>.

MLIB 17

Only <maxnumber> pointers allowed. The number of traceable variables that can be referenced by pointers is limited to <maxnumber>.

MLIB 18

Variable name missing. The first argument of one of the address evaluation functions must be the name(s) of variables() for which the descriptor is to be obtained.

1. Type **`help <function_name>`** in the MATLAB Command Window to get information about the respective MLIB/MTRACE function.

MLIB 19

Processor specification missing or incorrect. If the multiprocessing mode is selected, you have to specify the processor for some functions.

- Type **`mlib('MultiProcessing', 'OFF')`** if you want to proceed with MLIB/MTRACE in the single-processing mode.
- Type **`help <function_name>`** in the MATLAB Command Window to get information about the respective MLIB/MTRACE function and how to specify the processor.

MLIB 20

Variable which resides on the selected board cannot be directly accessed. Error caused by variable <variableName>

Only global variables residing in

- Global memory of DS1103
- Cached memory of DS1104
- Global memory of DS1401
- Non cached global memory of DS1005
- Non cached global memory of DS1006

boards can be accessed directly by the functions ReadDirect and WriteDirect.

MLIB 21

RESERVED

MLIB 22

RESERVED

MLIB 23

Error while accessing the field <field_name>. Field missing.

The function that issued this error message requires a variable descriptor (a structure) as an input parameter.

1. Use only descriptors that are returned by one of the address evaluation functions or generated by the SetVarProperty function.

MLIB 24

RESERVED

MLIB 25

RESERVED

MLIB 26

File name extension missing. SDF or TRC expected. The extension of the specified variable description file name must be TRC or SDF. In the second case the multiprocessing mode will be activated.

MLIB 27

Size mismatch between input parameter. The Write function requires that a set of data is specified for each `var_vec`. The number of elements to be written at memory address `var_vec(i).addr` must match the length given under `var_vec(i).length`.

1. Type **`help <function_name>`** in the MATLAB Command Window to get more information.

MLIB 28

Invalid property <property_name>.

1. Type **`help <function name>`** in the MATLAB Command Window to get information about the function <function name> that issued this error message.

MLIB 29

Bad value for property <property_name>

1. Type **`help <function name>`** in the MATLAB Command Window to get information about the function that issued this error message.

MLIB 30

Invalid property/value pair arguments.

Type **`help <function name>`** in the MATLAB Command Window to get information about the function that issued this error message.

MLIB 31

Multiprocessor application not specified. If the function GetApplInfo is called in SP mode and the CPU-property is specified, you must also specify the name of the SDF file describing the multiprocessor application.

For further information, type **help GetApplInfo** in your MATLAB Command Window.

MLIB 32

Can't create a variable descriptor. <property_name> missing.

1. If you create a variable descriptor using the SetVarProperty function, you must also specify the following properties `name`, `addr`, `type`. In MP mode, the CPU name is also required.

MLIB 33

Trigger mode cannot be set. Trigger variable not specified.

Before the trigger mode can be turned on, the trigger variable must be specified.

MLIB 34

Illegal type of variable on which to trigger. The permitted data types for the trigger variable are:

- `Int8`, `Int8Ptr`, `Int16`, `Int16Ptr`, `Int32`, `Int32Ptr`, `UInt8`, `UInt8Ptr`, `UInt16`, `UInt16Ptr`, `UInt32`, `UInt32Ptr`, `FloatIeee32`, `FloatIeee32Ptr`, `FloatIeee64`, `FloatIeee64Ptr`

MLIB 35

Illegal file extension. Type **`help <function_name>`** in the MATLAB Command Window to get information on the respective functionthat issued this error message.

MLIB 36

Variable <variable_name> cannot be traced. Illegal data type. The permitted data types for the trace variable are:

- `Int8, Int8Ptr, Int16, Int16Ptr, Int32, Int32Ptr, UInt8, UInt8Ptr, UInt16, UInt16Ptr, UInt32, UInt32Ptr, FloatIeee32, FloatIeee32Ptr, FloatIeee64, FloatIeee64Ptr`

MLIB 37

No trace variables specified. Before the StartCapture function can be called, the trace variables must be specified. See the description of the Set function for information on how to specify the variable to be traced.

MLIB 38	**Set the time frame for capturing real-time data first.** Before the function StartCapture can be called, you must specify the time frame. See the description of the Set function for information on how to specify the data capturing time frame.
MLIB 39	RESERVED
MLIB 40	**The collected data is stored directly into the MAT file. Please change the file extension to .mat.**
MLIB 41	**StepSize not specified in the <file_name> file. Please use the 'Samples' property instead of 'Interval'.** The information about StepSize is not saved in the MAT file <file_name> generated during continuous data acquisition (see experiment_info structure). For this reason the interval at which the data is to be loaded into the MATLAB workspace can only be specified by using the Samples property. See the description of the LoadFile function for further information.
MLIB 42	**Continuous mode cannot be activated. Set the 'Trigger' property to 'OFF' first.** The continuous data capture mode requires that triggering is turned off.
MLIB 43	**Inconsistent specifications of the capture frame.** If the capture frame is defined by Delay/NumSamples as well as by Start/Stop properties during one call of the Set function, MLIB/MTRACE checks if the specifications are consistent. The delay and number of samples resulting from Start, Stop and StepSize must match the values of Delay and NumSamples.
MLIB 44	**Time interval cannot be specified! Unknown sample time.** The capture frame can be specified by using the Start and Stop properties only if the StepSize or MaxSamplesPerPeriod was defined beforehand.
MLIB 46	**No data to be read in the trace buffer. Call the function StartCapture first.** Before the FetchData function can be called, data capturing must have been completed. If all the data was read from the trace buffer and the **AutoClearBuffer** property was set to 'ON', any call to **FetchData** without first capturing new data results in this error message.

MLIB47	**TriggerState cannot be obtained. Call the StartCapture function first.** The actual trigger state can only be obtained after the start of the data capture process.
MLIB 48	**Data acquisition options not specified for service/service group <service number>/<service group>.** Before the StartCapture function is called with the specified service number (SP) or service group (MP1005, MP1006), the Set function must be called to initialize the options of the data acquisition performed by the specified service/service group.
MLIB 49	**Improper value for service number.** MLIB/MTRACE supports up to 10 services concurrently running within the real-time application. The valid service numbers are 1-10.
MLIB 50	**Acquisition service <service_number> not available or not active.** The specified service <service_number> either is not available within the real-time application or is not used for real-time data acquisition.
MLIB 51	**Trace buffer overloaded. Select less variables or increase the trace interval.** If MLIB is running in continuous mode, the collected data is written to the ring buffer. The size of this ring buffer is set by NumSamples or StepSize/Start/Stop. After the last position of the buffer is reached, the acquisition service continues writing the collected data at the beginning of the ring buffer, etc. The existing, previously written data will be overwritten only if it was read (fetched) in time by MLIB. Otherwise this error message is displayed.
MLIB 52	RESERVED
MLIB 53	RESERVED
MLIB 54	RESERVED
MLIB 55	**In the continuous mode with DS1005-MP or DS1006-MP systems the format of the output data must be set to 'structure'.** If continuous mode is selected for multiprocessor systems based on DS1005 or DS1006, the data collected on different processors is not put in a single time correlation. It is therefore not possible to return the data in the MATLAB workspace in a matrix with a common time axis.

However, in the generated MAP files (if StreamToDisc mode is activated) the data is still stored as a matrix, independently of the selected format of the output data.

Use the Set function property 'OutputFormat' to set the format for the output data.

MLIB 56

The trace interval can be specified either by Start/Stop/StepSize or by NumSamples/Delay properties combination In multiprocessing mode (DS1005, DS1006) it is possible to trace data with different sampling periods (step sizes) on different processors or trace data asynchronously. If the trace interval is specified by NumSamples and Delay, MLIB/MTRACE collects the desired number of samples on each processor.

If the trace interval is specified by Start, Stop and StepSize, MLIB/MTRACE finishes the data acquisition after a specified time. Different numbers of samples may be collected depending on each processor's sampling periods.

MLIB 57

StreamToDisk mode cannot be activated. Set the name of the MAT file first. Before StreamToDisk mode can be selected, you must specify the name of the MAT file where the collected data is to be stored. Use the Set function property 'FileName' to specify the name of the MAT file.

MLIB 100

Specified SDF file does not describe a multiprocessor system based on DS1005 or DS1006 boards. The SDF file which is specified as a parameter of the multiprocessing function must describe a multiprocessor system. Check your SDF file.

MLIB 101	**No processor board registered with index <board_index>.** There is no board registered in ControlDesk's Platform Manager with the index <board_index>. 1. Invoke **`mlib(´GetBoardInfo´)`** to get information on which boards are registered at ControlDesk's Platform Manager.
MLIB 102	**Processor board <board_name> not registered.** The board is not registered in ControlDesk's Platform Manager. 1. Invoke **`mlib(´GetBoardInfo´)`** to get information which boards are registered at ControlDesk's Platform Manager. For detailed information, refer to the *Installation and Configuration Guide* of your hardware.
MLIB 103	**No real-time program running on board <name>.** A program must be downloaded to the processor board before you can work with MLIB/MTRACE functions. 1. Use ControlDesk's Platform Manager to download an application to the selected processor board.
MLIB 104	**Board is locked by another host application.** This error message occurs if MLIB/MTRACE tries to access a board simultaneously with another host program. This can happen only if another host application crashed when it was about to access a board and was not able to finish the operation. 1. If continuous attempts to call the respective MLIB/MTRACE function fail, refresh the hardware connection by using ControlDesk's Platform Manager (Platform - Initialization - Refresh the platform connection).
MLIB 105	**Can't reset/restart, real-time application program is locked.** As long as a real-time application is locked it cannot be reset or restarted.
MLIB 106	**Processor <processor_name> not found.** The processor does not belong to the multiprocessor system specified in the SDF file. 1. Check if <processor_name> is spelled correctly.
MLIB 107	RESERVED

MLIB 108	**The board <board_name> cannot be initialized The file <file_name> is missing.** A processor board can only be initialized if a real-time application is currently running on it and the matching MAP file exists.
MLIB 109	**Can't find processor board with index <index>.** There are no boards with the specified index registered in ControlDesk's Platform Manager. 1. Invoke mlib('`GetBoardInfo`') to get information on which boards are currently registered with which index in ControlDesk's Platform Manager.
MLIB 110	**Multiprocessor system is not running or it is running the wrong application.** 1. Check whether all processor boards related to the multiprocessor application (as defined in the system description file, SDF) are registered in ControlDesk's Platform Manager, and whether the correct real-time programs are running on them.
MLIB 111	RESERVED
MLIB 112	RESERVED
MLIB 113	RESERVED
MLIB 114	RESERVED
MLIB 115	**master_cmd_server missing or not initialized properly.** To read data from or write data to the DS1103, DS1104, DS1401, DS1005 or DS1006, the `master_cmd_server` function must be called in the background task of the real-time application. This service call is inserted automatically for applications generated by RTI.
MLIB 116	**Time out. Maybe master_cmd_server is not called.** The application is probably stopped or has crashed. Reload the real-time application and try again.

MLIB 117	**Communication error.** There is a failure in the communication between the host PC and the real-time processor board. The possible reasons for this are: `master_cmd_server` is not initialized, the application crashed or illegal data access was performed, such as read/write from a wrong memory address.
MLIB 118	RESERVED
MLIB 119	RESERVED
MLIB 120	**No free entry available in the instances list of service <service_number>.** Each `host_service` specified by a service number and called within the real-time application can collect data for up to 3 independent host PC applications. For example, if ControlDesk and Pylib already used the specified service <service_number> for real-time data acquisition, it has only one service entry free, i.e., for MLIB. This error message occurs, for example, from an attempt to start data acquisition with the service <service_number> with an additional MLIB instance.
MLIB 121	**Cannot install an acquisition service.** An entry of the structure `_TraceServiceStruct` cannot be found in the MAP file belonging to the currently running application. Probably the data capture service was not correctly initialized. Check if the function `init_trace()` is called in your application.
MLIB 122	RESERVED
MLIB 123	**Trigger variable does not belong to selected MP subsystem.** Refer to *Using MLIB/MTRACE for Multiprocessor Systems* on page 21.
MLIB 124	RESERVED
MLIB 125	**Can’t find address of variable <name>.** Check if the global variable <name> is spelled correctly. Inspect the linker map file of the currently loaded real-time application where all global variables appear with a preceding underscore.

MLIB 126	**Can't find variable <name> in <filename>** If <filename> points to a TRC file, inspect the TRC file or start the TrcView utility to obtain permissible variable names. Check If <name> is spelled correctly. If <filename> points to a MAP file, check if <name> is spelled correctly. Inspect the linker map file of the currently loaded real-time application, where all global variables appear with a preceding underscore.
MLIB 127	**Time stamping is not supported. Probably the application was build by the older version on the real-time library.** Time stamping on PPC boards is supported by the following RTLIB software: ■ RTLIB 1103 version 1.3 or higher ■ RTLIB 1401 version 1.3 or higher ■ RTLIB 1005 version 1.1 or higher ■ RTLIB 1006 version 1.0 or higher To use the time stamping feature with MLIB/MTRACE, you have to rebuild your applications with the appropriate version of the RTLIB library.
MLIB 128	**Your application has been created with an older version of the real-time library which does not support the distributed tracing in multiprocessing mode. Please recompile your application and try again to start this function.** The distributed tracing is supported by: ■ RTLIB 1005 version 1.1 or higher ■ RTLIB 1005 version 1.0 or higher
MLIB 129	**Processor ID mismatch.** Each member processor of a multiprocessor system based on DS1005 or DS1006 has its own unique processor index (ID). this error message occurs if the processor indices that are read directly from the real-time application do not match the indices specified in the SDF file (Keyword: ServiceID). If the real-time application was generated by RTI-MP, these indices will be written to the SDF file automatically. If the real-time application was handcoded, you must set the indices, and they must start with 0 in a real-time application and with 1 in a SDF file.
MLIB 130	**No free entry in the reservation table.**

MLIB 131

Trigger variable does not belong to the selected service group <service_group>! A service group specifies the processors and the services with which data acquisition can be carried out. Only variables which reside on processors belonging to a specified service group can be traced and selected as trigger variables.

Example An example of calling up this error message:

A DS1005-based multiprocessor system consists of 3 member processors *Alpha*, *Beta* and *Gamma*. The ServiceGroupAB is defined as *Processor Alpha*, *service 1* and *Processor Gamma*, *service 1*.

An attempt to set a trace variable residing on processor Beta as a trigger variable will cause this error message.

- Call `mlib('Get','ServiceGroup',<service_group>)` to get information about the CPUs and the services combined in the service group.
- Call the function 'Set' to add a processor with the desired trigger variable to your service group.

MLIB 132

Trace variable does not belong to the selected service group <service_group>! A service group specifies the processors and the services with which data acquisition can be carried out. Only variables which reside on processors belonging to a specified service group can be traced and selected as a trigger variables.

Example An example of calling up this error message:

A DS1005-based multiprocessor system consists of 3 member processors *Alpha*, *Beta* and *Gamma*. The ServiceGroupAB is defined as *Processor Alpha*, *service 1* and *Processor Gamma*, *service 1*.

If a trace variable resides on processor Beta, an attempt to set this trace variable for data acquisition carried out by service group ServiceGroupAB will cause this error message.

- Call `mlib('Get','ServiceGroup',<service_group>)` to get information about the CPUs and the services combined in the service group.
- Call the 'Set' function to add a processor with the desired trace variable to your service group.

MLIB133

Error while parsing the SDF file <file_name>. <Message>

Parsing of SDF file failed. The reason is given in the <message> string.

MLIB134	**Error while parsing the TRC file <file_name>. <Message>** Parsing of TRC file failed. The reason is given in the <message> string.
MLIB135	**Error while reading file <file_name>. Probably file was not created by MLIB/MTRACE** The LoadFile function can only be used with the MAT file generated by MLIB/MTRACE in StreamToDisk mode.

MLIB136

It is not possible to find common time axis for variables traced on the single processors. Probably the data has been collected with different sampling periods or asynchronously. Use 'structure' output format instead of 'matrix'. On multiprocessor systems based on DS1005 or DS1006, variables are traced separately on each processor. The time stamps at which the data was collected are also traced. If the output format is set to 'matrix' ('matrix with time' in MP mode), the data series from all processors must have the same time stamps. It is only possible if data acquisition is carried out with the same sampling period on each processor and if the absolute times on each processor are synchronized with each other.

Use the Set function property 'OutputFormat' to change the format of the traced data from 'matrix' to 'structure'

MLIB137

The time stamps have not been traced during the data acquisition. Due to it is not possible to find a time correlation between data being collected on different processors. The reason can be a wrong call to the host_service macro in your real-time application. The time stamps can be collected only if the host_service is called with the pointer to the time stamp structure as a second parameter:

```
host_service(service_no, ts)
```

If your application was generated by RTI/RTI-MP, a call to the macro is made automatically. If your application is handcoded, make sure that the macro is called correctly.

MLIB 138

The TRC file <file_name> was generated/written in the old syntax! It does not contain information about which service belongs to which service group (denoted by service name).

This error message may occur if the ShowServices function is called.

To show service groups predefined in the TRC file, entries in the form of

```
sampling_period[<index>]
{
   value:      0.0
   alias:      "<serviceName>"
}
```

are required. They are generated with RTI/RTI-MP version 3.6 and higher. Regenerate your application with the newer RTI/RTI-MP version if you want to use this function.

MLIB 139

Time out after <number> sec while waiting for desired number of samples to be collected in the trace buffer. If necessary you can increase the time after which the timeout error is reported by altering the 'Set' function property 'TimeOutValue'.

Altering this property allows you to increase the time after which this error message is displayed, for example, if your applications run very slowly.

If the FetchData function is called with `count` as an input argument and data acquisition is still in progress, MLIB/MTRACE waits until the desired number of samples are collected in the trace buffer.

To avoid MLIB/MTRACE hanging when the data is not collected in the trace buffer (for example, if the application crashed), the MATLAB Command Window is automatically reactivated after <number> second(s). In this case a timeout error message is displayed.

MLIB 140

Time out after <number> sec while waiting for number of samples collected in the trace buffer greater or equal the StartSample value. If necessary you can increase the time after which the timeout error is reported by altering the 'Set' function property 'TimeOutValue'. Altering this property allows you to increase the time after this error message is displayed, for example, if your applications run very slowly.

If the FetchData function is called with the 'StartSample' property while data acquisition is still in progress, MLIB/MTRACE waits until the desired number of samples are collected in the trace buffer.

To avoid MLIB/MTRACE hanging when the data is not collected in the trace buffer (for example, if the application crashed), the MATLAB Command Window is automatically reactivated after <number> second(s). In this case a timeout error message is displayed.

MLIB 200

Host memory allocation failure. There is insufficient memory on the host heap.

MLIB 201	**Real-time processor memory allocation failure.** There is insufficient free memory on the real-time processor.
MLIB 202	**Host PC - real-time processor communication error.** This error can occur in an attempt to download data to or upload data from a processor board that is not connected to a host PC (for example, AutoBox was turned off).
MLIB 203	**Can’t open file <file_name>.** The most likely reason is that the file cannot be found or does not exist at all. 1. Check if <file name> is spelled correctly and you are in the appropriate working folder. All linker map files, associated objects (PPC, X86) and SDF files (for multiprocessor mode only) must reside in the same folder.
MLIB 204	**Cannot find file <filename>.** The file probably does not exist or is not in the MATLAB search path.
MLIB 205	**Cannot load file <filename>.** Please check if the specified file <filename> exists.
MLIB 206	**Cannot write data into file <filename>.** The destination disk on which the file resides is probably full.
MLIB 207	**Cannot read from file <filename>.** Check if the file <filename> exists and can be found on the MATLAB search path.
MLIB 208	RESERVED
MLIB 209	**Cannot create file <filename>.** Check if there is enough free space on your hard disk. Alternatively, check if the file <filename> already exists and is read-only. In this case it cannot be overwritten. ■ Free space on your local disk. ■ Change the attribute of the existing file <filename>. ■ Change the name or the location of the file.

Appendix

Where to go from here

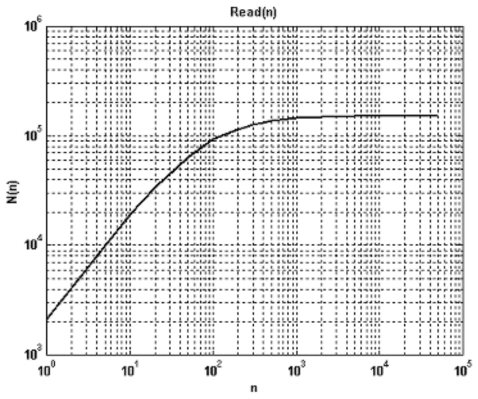
- *MLIB/MTRACE Benchmarks* on page 125
- *Impact of MLIB/MTRACE on Execution Time* on page 128

MLIB/MTRACE Benchmarks

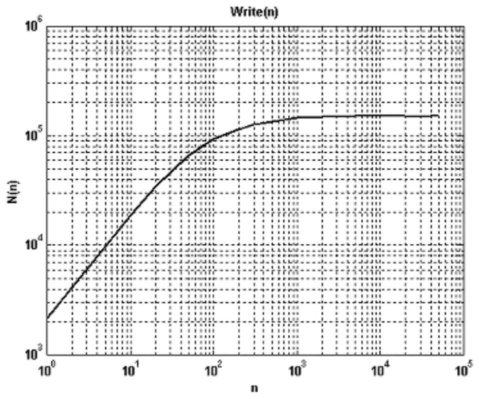
As described in *Multiple MLIB/MTRACE Data Captures in One Real-Time Application* on page 76, the time needed for processor board access from host applications mainly depends on the type of connection; for example, whether a PC/AT bus interface or a TCP/IP network is used. It is impossible to provide general benchmarks for network connections. Network performance depends on many factors such as the type of Ethernet cards and cabling, the implementation of the TCP/IP software, the TCP/IP packet length, and the actual traffic. In typical configurations, network access is 3-10 times slower than PC/AT bus access.

The following two graphs show the performance of MLIB/MTRACE with a Pentium II 400 MHz host connected to a DS1005/400 MHz processor board via a bus interface.

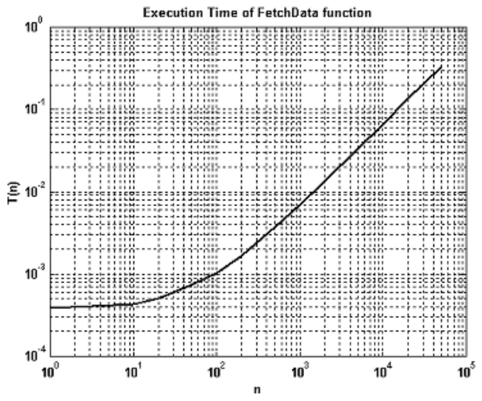
The first graph shows the number of 64-bit words transferred per second (N) with mlib('Read', `var`) versus the number of words to be read (n = `var.length`):



The following graph shows the number of transferred 64-bit words transferred per second (N) via mlib('Write', var, 'Data', fval) versus the number of words to be written (n = var.length):



The following graph shows the execution time of the MLIB/MTRACE FetchData function with a Pentium II 400 MHz PC connected to a DS1005/400 MHz processor board via a bus interface. Nearly identical results are achieved for other dSPACE boards.



Measurement details

For further information, refer to *Impact of MLIB/MTRACE on Execution Time* on page 128.

Execution times

Execution times are available for:

- *MLIB/MTRACE with DS1103* on page 128
- *MLIB/MTRACE with DS1401* on page 128
- *MLIB/MTRACE with DS1005* on page 129

- Execution times for DS1104 and DS1006 are not available at all.
- Execution times for DS1103, DS1401 and DS1005 have not been measured with the latest board revisions. The processor clock rates of the boards used are documented.

Related topics

Basics

- *Impact of MLIB/MTRACE on Execution Time* on page 128

Impact of MLIB/MTRACE on Execution Time

In addition to the execution time necessary to calculate model equations and perform I/O routines of a real-time application, the service macro `host_service`, which must be inserted into the interrupt service routine to let MLIB/MTRACE work (see *Getting Started* on page 17), requires additional execution time. The amount of execution time needed depends on many parameters, such as the number and kind of variables to trace, the chosen trigger options, and the memory location used for program data and captured data. Insufficient execution time will result in processor overflow when the data acquisition service is performed.

The capture of directly accessed variables requires different execution times than the capture of variables referenced by pointers. In the following tables, the symbol `nptr` indicates the number of variables accessed by pointers, and `nvar` is used for the number of the directly accessed variables.

MLIB/MTRACE with DS1103

The execution times for the host_service code were measured on a 333 MHz DS1103 processor board. The program code and data were loaded into local memory. The traced data is written to the trace buffer located in the global memory. The following table shows the measurement results.

Trigger State	Execution Time of host_service in µsec
pre-trigger	ca. 3.2 + nvar · 0.061 + nptr · 0.069
post-trigger	ca. 3.2 + nvar · 0.061 + nptr · 0.069
trigger OFF	ca. 3.2 + nvar · 0.061 + nptr · 0.069

MLIB/MTRACE with DS1401

The execution times for the host_service code were measured on a 200 MHz DS1401 processor board. The program code and data were loaded into local memory. The following table shows the measurement results.

Trigger State	Execution Time of host_service in µsec
pre-trigger	ca. 4.9 + nvar · 0.2 + nptr · 0.2
post-trigger	ca. 5.3 + nvar · 0.2 + nptr · 0.2
trigger OFF	ca. 4.7 + nvar · 0.2 + nptr · 0.2

MLIB/MTRACE with DS1005

The execution times for the host_service code were measured on a 400 MHz DS1005 processor board. The program code and data were loaded into the cached area of the global memory. The collected data was written to the trace buffer located in the non cached area of the global memory. This is the predefined configuration. The following table shows the measurement results.

Trigger State	Execution Time of host_service in µsec
pre-trigger	ca. 2.3 + nvar · 0.083 + nptr · 0.083
post-trigger	ca. 2.3 + nvar · 0.083 + nptr · 0.083
trigger off	ca. 2.3 + nvar · 0.083 + nptr · 0.083