MLIB/MTRACE Migration Guide

How to migrate MLIB/MTRACE scripts to XIL API .NET using the MATLAB Interface to .NET Framework

November 2016
How to Contact dSPACE

Mail: dSPACE GmbH
Rathenastraße 26
33102 Paderborn
Germany
Tel.: +49 5251 1638-0
Fax: +49 5251 16198-0
E-mail: info@dspace.de
Web: http://www.dspace.com

How to Contact dSPACE Support

To contact dSPACE if you have problems and questions, fill out the support request form provided on the website at http://www.dspace.com/go/supportrequest.
The request form helps the support team handle your difficulties quickly and efficiently.
In urgent cases contact dSPACE via phone: +49 5251 1638-941 (General Technical Support)

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/go/support for software updates and patches.

Important Notice

This document contains proprietary information that is protected by copyright. All rights are reserved. The document may be printed for personal or internal use provided all the proprietary markings are retained on all printed copies. In all other cases, the document must not be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form, in whole or in part, without the prior written consent of dSPACE GmbH.

© 2011 - 2016 by:
dSPACE GmbH
Rathenastraße 26
33102 Paderborn
Germany

This publication and the contents hereof are subject to change without notice.
CalDesk, ConfigurationDesk, ControlDesk, MicroAutoBox, MicroLabBox, SCALEXIO, SYNECT, SystemDesk, TargetLink and VEOS are registered trademarks of dSPACE GmbH in the United States or other countries, or both. Other brand names or product names are trademarks or registered trademarks of their respective companies or organizations.
Contents
1 Introduction .................................................................................................................................................................................. 4
2 Required Knowledge ................................................................................................................................................................ 5
3 Prerequisites/Preparatory Tasks ........................................................................................................................................... 5
4 Migration Examples................................................................................................................................................................... 6
   4.1 Initializing MLIB, Selecting the hardware Platform and Specifying the Variable Description ............... 6
       4.1.1 Example of an MAPort configuration file......................................................................................................... 7
   4.2 Reading and Writing Parameters ..................................................................................................................................... 7
   4.3 Capturing ............................................................................................................................................................................... 8
   4.4 Triggered Measuring ........................................................................................................................................................ 10
   4.5 Processing the Measurement Results (CaptureResults) .................................................................................. 11
5 Additional Information on Using dSPACE XIL API .NET in MATLAB ............................................................... 12
   5.1 Referencing the XIL API .NET Assemblies .............................................................................................................. 12
   5.2 Namespaces ...................................................................................................................................................................... 13
   5.3 Generic Collections from the .NET Framework ................................................................................................... 13
   5.4 Exception Handling ........................................................................................................................................................ 14
   5.5 Control Characters in Variable Paths ....................................................................................................................... 15
6 Reading out Processor Board Information ................................................................................................................... 15
7 Complete Sample Scripts ..................................................................................................................................................... 16
Further Reading ........................................................................................................................................................................... 17
1 Introduction

Using the dSPACE XIL API .NET server in the M scripting language is the method dSPACE recommends for migrating existing M scripts that contain MLIB/MTRACE commands.

This guide shows you:
- How to use the dSPACE XIL API .NET server in the M scripting language
- How to map MLIB/MTRACE commands to XIL API classes and methods

The dSPACE XIL API .NET server provides a programming interface that covers a part of the ASAM standard XIL API 2.0.1.

The dSPACE .NET Implementation of the ASAM XIL API 2.0.1 standard (dSPACE XIL API .NET server for short) currently covers large parts of the Model Access Port (MAPort) for access to real-time applications on dSPACE HIL and RCP platforms; larger parts of the common namespace of the XIL API, in which shared classes, data structures, and data types are defined for the XIL API Ports (especially the classes for acquisition data and real-time stimulation); and the Electrical Error Simulation Port (EESPort) for fault simulation on SCALEXIO systems.

The MAPort of the dSPACE XIL API .NET server currently supports all of the dSPACE processor boards on the market, including the multiprocessor and multicore platforms:
- DS1103
- DS1104
- DS1202 (MicroLabBox)
- DS1401 (MicroAutoBox and MicroAutoBox II)
- DS1005
- DS1006
- DS1007
- SCALEXIO
- VEOS

In contrast, MLIB/MTRACE supports only the following platforms:
- DS1103
- DS1104
- DS1401
- DS1005
- DS1006

For a number of years, MathWorks (TMW) has supported the use of .NET assemblies directly in the MATLAB M scripting language. This MATLAB product feature is called the MATLAB Interface to .NET Framework. The MATLAB Interface to .NET Framework lets you use the dSPACE XIL API .NET server with a simple method in the M scripting language.

For a comparative overview of the differences between MLIB/MTRACE and XIL API, refer to the following table:

<table>
<thead>
<tr>
<th></th>
<th>MLIB/MTRACE</th>
<th>XIL API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional command interface</td>
<td>Object-oriented high-level interface</td>
<td></td>
</tr>
<tr>
<td>Proprietary interface</td>
<td>API is standardized by ASAM</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>Supports auto-completion</td>
<td></td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>MLIB/MTRACE</th>
<th>XIL API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requires HW registration with ControlDesk 3.x</td>
<td>HW registration with ControlDesk (\geq 4.x)</td>
</tr>
<tr>
<td>Does not support SCALEXIO or VEOS</td>
<td>Supports all dSPACE HW platforms and VEOS</td>
</tr>
<tr>
<td>-</td>
<td>Supports real-time stimulation via XIL API</td>
</tr>
<tr>
<td>-</td>
<td>Supports complex and flexible trigger conditions</td>
</tr>
<tr>
<td>-</td>
<td>Supports stop triggers</td>
</tr>
<tr>
<td>Error handling through error codes</td>
<td>Structured exception handling</td>
</tr>
<tr>
<td>-</td>
<td>Supports multiple instances of RT application access</td>
</tr>
<tr>
<td>-</td>
<td>Supports multiple instances of captures</td>
</tr>
<tr>
<td>-</td>
<td>Future support of new hardware platforms</td>
</tr>
<tr>
<td>Supports address-based access to the platform’s memory</td>
<td>No support for address-based access, TRC file is required</td>
</tr>
<tr>
<td>Supports MAP files as variable description</td>
<td>No support for MAP files as variable description; SDF/TRC files are needed</td>
</tr>
</tbody>
</table>

### 2 Required Knowledge

You do not need previous knowledge of the Microsoft .NET framework or any of the .NET programming languages, such as C# or VB.NET. However, you should have a basic knowledge of developing programs in the M scripting language.

The following sections show concrete examples and code snippets that explain step by step how to migrate existing MLIB/MTRACE scripts to XIL API .NET.

First, the guide focuses on migrating MLIB commands to classes/methods of the XIL API. The final sections of this guide provide additional information on exception handling, using namespaces, and how to use generic collections of the Microsoft .NET Framework.

### 3 Prerequisites/Preparatory Tasks

The following sections requires that

1. You own a copy of the Platform API Package licenses
2. The dSPACE XIL API .NET server and the dSPACE Python Extensions are installed (the easiest way to install all of your licensed software is to execute the dSPACE_MasterSetup.exe file, located in the root folder of your dSPACE Release DVD)
3. The dSPACE processor board was registered either with ControlDesk 4.x or later, or with the dSPACE Platform Management API
4. The real-time application that is used was generated with RTI from at least dSPACE Release 7.3. This is required so that the new measurement and calibration service DsDaq is available in the real-time application.
5. The used MATLAB version is at least R2013a. Earlier versions of MATLAB do not fully support the .NET Framework 4.0 and its features.
4 Migration Examples

This section uses specific examples to show you step by step how to convert MLIB/MTRACE commands into the classes and method calls of dSPACE XIL API .NET. In each case, a short section of the MLIB script source code is compared to an equivalent section that uses the XIL API .NET server in M.

These samples are based on dSPACE Release 2015-A. Not all samples work with older versions. For example the dSPACE Platform Management API changed. For detail, refer to dSPACE Platform Management API Reference > Introduction > Basics on the Platform Management API > Migrating applications.

4.1 Initializing MLIB, Selecting the hardware Platform and Specifying the Variable Description

Each MLIB script starts with the initialization of MLIB, the selection of the hardware platform you want to work with, and a specification of the variable description associated with the real-time application.

Here is an excerpt from an MLIB script:

```matlab
% select processor board
mlib('SelectBoard','DS1005');
```

This means that MLIB determines which variable description (i.e., which TRC file) to use based on the hardware platform loaded for the real-time application.

The initialization of the XIL API .NET server in M consists of several steps. First, the XIL API .NET server is loaded to the workspace of MATLAB. In the next step, the namespace notation can be simplified in the script by an import directive so you do not have to specify the complete namespace each time:

```matlab
% load the XIL API .NET assemblies
NET.addAssembly('ASAM.XIL.Interfaces');
NET.addAssembly('ASAM.XIL.Implementation.TestbenchFactory');
% import namespace (this is only an example)
import ASAM.XIL.Implementation.TestbenchFactory.Testbench.*;
```

XIL API .NET uses the factory pattern for creating all kinds of Testbench-specific objects. At first, you have to create a TestbenchFactory and a VendorSpecificTestbench:

```matlab
% create a TestbenchFactory object; the TestbenchFactory is needed to
% create the vendor-specific Testbench
tbFactory = ASAM.XIL.Implementation.TestbenchFactory.Testbench.TestbenchFactory();
% create a dSPACE Testbench object; the Testbench object is the central object to access
% factory objects for the creation of all kinds of Testbench-specific objects
testBench = tbFactory.CreateVendorSpecificTestbench('dSPACE GmbH', 'XIL API', '2015-A');
```

In the next step, an instance of the MAPort class is created to access the values of the real-time application. To configure an MAPort the following information is required:
• The path and file name of the variable description belonging to the real-time application to be used
• The platform identifier as it appears in ControlDesk 4.x or later after successful registration

You need to share these two bits of information with the MAPort by calling the LoadConfiguration() method with a path to the MAPort configuration file.

Corresponding XIL API .NET code fragment:

```csharp
% create a new instance of XIL API class MAPort
demoMAPort = testBench.MAPortFactory.CreateMAPort('DemoMAPort');
demoMAPortConfig = demoMAPort.LoadConfiguration('C:\DemoMAPortConfiguration.xml');
demoMAPort.Configure(demoMAPortConfig, false);

% add code for calibration and/or measurement here

demoMAPort.Dispose();
```

The XIL API .NET server has multi-instance capability: any arbitrary number of MAPort class instances can be created to perform tasks such as addressing several processor boards at the same time. This is not possible in MLIB/MTRACE.

4.1.1 Example of an MAPort configuration file

With the following configuration file, you can access the real-time application that is specified in the variable description file (SDF file) running on a modular system based on DS1005.

In the PlatformName key, you have to enter the platform name that you specified when you registered the platform.

```xml
<?xml version="1.0" encoding="utf-8"?>
<PortConfigurations>
  <MAPortConfig>
    <SystemDescriptionFile>C:\Application.sdf</SystemDescriptionFile>
    <PlatformName>ds1005</PlatformName>
  </MAPortConfig>
</PortConfigurations>
```

For details, refer to dSPACE XIL API Implementation Guide > Implementing an MAPort Application > Basics on the MAPort > Configuring the MAPort.

4.2 Reading and Writing Parameters

Suppose there are two parameters P and D of a PID controller in the real-time application that must be read and adjusted. When MLIB is used, the variable descriptors must first be specified by their paths:

```csharp
% full path to parameters P and D
PPath = {'Model Root/PID/P/Gain'};
DPath = {'Model Root/PID/D/Gain'};

% get descriptors for parameters P and D
PDescr = mlib('GetTrcVar', PPath);
DDescr = mlib('GetTrcVar', DPath);

% read parameters P and D
PValue = mlib('Read', PDescr);
```
DValue = mlib('Read', DDescr);

% write parameters P and D
mlib('Write', PDescr, 'Data', 0.01);
mlib('Write', DDescr, 'Data', 1.0);

Corresponding XIL API .NET code fragment:

% full path to parameters P and D
PPath = 'Model Root/PID/P/Gain';
DPath = 'Model Root/PID/D/Gain';

% read parameters P and D
PValue = demoMAPort.Read(PPath);
DValue = demoMAPort.Read(DPath);

% write parameters P and D
demoMAPort.Write(PPath, testBench.ValueFactory.CreateFloatValue(0.01));
demoMAPort.Write(DPath, testBench.ValueFactory.CreateFloatValue(1.0));

% dispose MAPort to free the allocated memory and any system resources
demoMAPort.Dispose();

A special feature in the above example is writing the variables by using write(). All numerical values are read and written by XIL API .NET with the help of value containers. Because no declaration of variables is necessary in M, no explicit type conversion is necessary for Read(). The situation is somewhat different when writing variables; here an explicit type conversion using FloatValue is required. The XIL API value containers are available from the VendorSpecificTestbench.ValueFactory object.

4.3 Capturing

This section shows how an MLIB code section can be ported to XIL API .NET for untriggered, continuous measurement. Two signals are to be measured over a period of 30 seconds.

In the following example, two signals are measured. MLIB code fragment:

% specify the variables used by MLIB and get their descriptors
variables = {'Model Root/Spring-Mass-Damper System/Out1';...
'Model Root/Signal Generator/Out1'};

[sys_output, sig_gen] = mlib('GetTrcVar', variables);

% set the options of data acquisition performed by service number 1 (default)
mlib('Set',...
  'Trigger', 'off',...
  'TraceVars', [sys_output;sig_gen],...
  'StepSize', 0.0001,....
  'Start', 0.0,....
  'Stop', 3,...
  'AcquisitionMode', 'continuous', ...
  'TimeStamping', 'ON');

out_data = [];

mlib('StartCapture'); % start capture
tic;
while toc < 30
  out_data = [out_data, mlib('FetchData')]; % append data to out_data
end;
To configure an XIL API .NET measurement, the following information is required:

- The name of a raster (also called a task) to be used for the measurement (MLIB does not use the symbolic name of the raster but the service number)
- The specification of the variable paths to be measured

In this XIL API example, a DurationWatcher specifies the duration of the measurement. It is assumed that the MAPort was set up before.

**Corresponding XIL API .NET M script section:**

```m
% name of the raster
taskName = 'HostService';

% create capture instance
capture = demoMAPort.CreateCapture(taskName);

% set variables to be captured
varList.Add('Model Root/Spring-Mass-Damper System/Out1');
varList.Add('Model Root/Signal Generator/Out1');
capture.Variables = varList;

% set the conditions for the stop trigger by defining an appropriate DurationWatcher
stopWatcher = testBench.WatcherFactory.CreateDurationWatcher(30.0);
capture.SetStopTriggerCondition(stopWatcher, 0.0);

At the start of each measurement, the instance of the Capture class must be told where to store the measurement data (e.g. in a data file or the main memory). To do this, this example includes a CaptureResultMemoryWriter to keep the measurement data in the main memory of the PC.

```m
% create an instance of a subclass of CaptureResultWriter
% in this case a CaptureResultMemoryWriter is used to store capture results in memory
captureWriter = testBench.CapturingFactory.CreateCaptureResultMemoryWriter();
capture.Start(captureWriter);
    pause(0.1);  % pause for 0.1 secs to reduce processor load
end;
capture.Stop();

% get capture result
captureResult = capture.CaptureResult;

% dispose Capture and MAPort to free the allocated memory and any system resources
capture.Dispose();
demoMAPort.Dispose();
```

What is important here is that the instance of the Capture class is released after its last use by Dispose() so that all of the associated resources are released as well.
4.4 Triggered Measuring

In the following example, measurement is to start when an amplitude with a rising edge exceeds a certain value. A pretrigger ensures that the measurement values are recorded for a certain period of time before start trigger event occurs. In addition, exactly 300 samples are to be measured.

In MLIB/MTRACE, this task can be performed as follows:

```matlab
% specify the variables used by MLIB and get their descriptors
variables = {'Model Root/Signal Generator/Amplitude'; ...
             'Model Root/Spring-Mass-Damper System/Out1'; ...
             'Model Root/Signal Generator/Out1'};

[ampl, sys_out, siggen] = mlib('GetTrcVar', variables);

% set the option of the data acquisition performed by service number 1 (default)
mlib('Set', ...
     'Trigger', 'ON',...
     'TriggerLevel', 0.25,...
     'TriggerEdge', 'rising',...  % default, can be omitted
     'TriggerVariable', ampl,...
     'TraceVars', [ampl; sys_out; siggen],...
     'NumSamples', 300,...
     'Delay', -50,...
     'TimeStamping', 'on');
```

XIL API .NET allows the formulation of very complex trigger conditions. You can define not only start triggers, but (in contrast to MLIB) also stop triggers. In XIL API .NET, triggers are implemented by using a Watcher. In addition to ConditionWatchers that allow the specification of freely definable trigger conditions, the stop trigger can also use a DurationWatcher, which terminates the measurement after an exactly defined time. The XIL API Watcher is implemented by the dSPACE XIL API .NET server through services of the real-time application. The evaluation of the trigger conditions and time periods is therefore very accurate and independent of the processes on the host PC.

Corresponding XIL API .NET code fragment:

```matlab
taskName = 'HostService';

% create capture instance
capture = demoMAPort.CreateCapture(taskName);

% capture each sample of the task
capture.DurationUnit = ASAM.XIL.Interfaces.Testbench.Common.Capturing.Enum.DurationUnit.eSAMPLES;
capture.Downsampling = 1;

% set variables to be captured
ampl    = 'Model Root/Signal Generator/Amplitude';
sys_out = 'Model Root/Spring-Mass-Damper System/Out1';
siggen  = 'Model Root/Signal Generator/Out1';
variableList.Add(ampl);
variableList.Add(sys_out);
variableList.Add(siggen);
capture.Variables = variableList;

% set a start and stop trigger
% set the conditions for the start trigger by defining an appropriate ConditionWatcher
startWatchDefs = NET.createGeneric('System.Collections.Generic.Dictionary', ...
                              {'System.String', 'System.String'}, ...)
                          {'System.String', 'System.String'});
```
startWatchDefs.Add('amplitude', 'Model Root/Signal Generator/Amplitude');

% trigger if the amplitude rises above 0.25 with a pretrigger of 50 samples
startWatcher = testBench.WatcherFactory.CreateConditionWatcher('posedge(amplitude, 0.25)',
startWatchDefs);
capture.SetStartTriggerCondition(startWatcher, -50);

% set duration watcher for 300 samples
stopWatcher = testBench.WatcherFactory.CreateDurationWatcher(300);
capture.SetStopTriggerCondition(stopWatcher, 0.0);

4.5 Processing the Measurement Results (CaptureResults)

After the measurement is completed with the XIL API, you want to further process the measurement results. In the following example, the measurement results are displayed graphically in a plot window.

The corresponding code section for MLIB/MTRACE is as follows:

```
out_data = mlib('FetchData');

% plot results
clf;
subplot(2,1,1);
plot(out_data(1,:),out_data(2,:))';
axis([out_data(1,1) out_data(1,end) -0.1 0.6]);
grid;
title('Trigger variable = Signal Generator-Amplitude');

subplot(2,1,2);
plot(out_data(1,:),out_data(3:4,:))';
axis([out_data(1,1) out_data(1,end) -1 1]);
grid;
title('SMD-System / Signal Generator');
xlabel('Time relative to the trigger event');

set(gcf,'NumberTitle','off');
set(gcf,'Name','Pre-Trigger with MLIB');
figure(gcf);
```

When using the XIL API, after the measurement was completed, you can get the measurement results as an instance of the CaptureResult class in the CaptureResult property of the Capture class:

```
% get the captured variables, containing the time of the sample points ("xAxis")
captureResult = capture.CaptureResult;
resultSGV = captureResult.GetSignalGroupValue(taskName);
resultAmpl = resultSGV.ExtractSignalValue(ampl);
resultSys_out = resultSGV.ExtractSignalValue(sys_out);
resultSiggen = resultSGV.ExtractSignalValue(siggen);

% plot results
clf;
subplot(2,1,1);
plot(resultAmpl.XVector.Value.ToArray().double, ...
resultAmpl.FcnValues.Value.ToArray().double');
axis([resultAmpl.XVector.Value.Item(0) ...
resultAmpl.XVector.Value.Item(resultAmpl.XVector.Value.Count -1) -0.1 0.6]);
grid;
title('Trigger variable = Signal Generator-Amplitude');
```
subplot(2,1,2);
plot(resultAmpl.XVector.Value.ToArray().double, ...
    [resultSys_out.FcnValues.Value.ToArray().double; ...
     resultSiggen.FcnValues.Value.ToArray().double]);
axis([resultAmpl.XVector.Value.Item(0) ...
     resultAmpl.XVector.Value.Item(resultAmpl.XVector.Value.Count - 1) -1 1]);
grid;
title('SMD-System / Signal Generator');
xlabel('Time relative to the trigger event');

set(gcf,'NumberTitle', 'off');
set(gcf,'Name', 'Pre-Trigger with MLIB');
figure(gcf);

5 Additional Information on Using dSPACE XIL API .NET in MATLAB

In the previous section, source code fragments were used to compare how sections of a typical MLIB
script can be ported to XIL API .NET. Some details were omitted to keep a clear view of the essential
points.

The following sections describe a few points in greater detail; points you have to consider when using
the dSPACE XIL API .NET server.

5.1 Referencing the XIL API .NET Assemblies

.NET assemblies must be loaded to MATLAB before you can use them. This requires using the MATLAB
command NET.addAssembly().

To reference the assemblies, you can use the available namespaces. These contain the classes and
methods of the .NET assemblies for use in the M source code.

The following .NET assemblies are needed for using the dSPACE XIL API NET server:
• ASAM.XIL.Interfaces.dll
• ASAM.XIL.Implementation.TestbenchFactory.dll

For the above assemblies, the relevant M code is as follows:

% add references to the XIL API .NET assemblies
NET.addAssembly('ASAM.XIL.Interfaces');
NET.addAssembly('ASAM.XIL.Implementation.TestbenchFactory');

Because the XIL API .NET assemblies are in the Global Assembly Cache (GAC), you do not have to
specify path information.

If there is more than one installation of the dSPACE XIL API .NET server, MATLAB automatically chooses
the highest version available. If a specific version is required, you have to use the fully qualified
assembly name. For version 2.0, the code is as follows:

NET.addAssembly('ASAM.XIL.Interfaces, Version=2.0.0.0, Culture=neutral,
    PublicKeyToken=bf471dff114ae984');
To determine the fully qualified assembly name, you can use Windows PowerShell with the following command:

```powershell
[System.Reflection.AssemblyName]::GetAssemblyName("C:\Path\To\MyAssembly.dll").FullName
```

### 5.2 Namespaces

In .NET, using namespaces is common. All classes of .NET frameworks are organized like this. The XIL API .NET is also divided into namespaces.

Example:

In XIL API .NET, the `TestbenchFactory` class is located in the namespace `ASAM.XIL.Implementation.TestbenchFactory.Testbench`.

In M, you can create a new instance of the `TestbenchFactory` class as follows:

```matlab
% create a TestbenchFactory object
tbFactory = ASAM.XIL.Implementation.TestbenchFactory.Testbench.TestbenchFactory();
```

You can use the import directive in M so you do not have to write the complete namespace each time you create a new instance.

```matlab
import ASAM.XIL.Implementation.TestbenchFactory.Testbench::*;
```

This makes the code much more compact:

```matlab
% create a TestbenchFactory object
tbFactory = TestbenchFactory();
```

If you use the import command in a MATLAB function, add the corresponding .NET assembly before calling the function. You cannot add the .NET.addAssembly(‘...’) to a MATLAB function. MATLAB processes the M file code before executing the .NET.addAssembly command. In this case, the function is not fully qualified and MATLAB does not recognize the name. For details, refer to Use import in MATLAB Functions in the MATLAB documentation.

### 5.3 Generic Collections from the .NET Framework

Using generic collections from the .NET framework requires a special approach in M. This section shows you how to create such a list, which is required for creating a XIL API Capture class, for example.

The list must have entries of the data type string; the values are added to the list by .NET methods.

In M, you have to use the special function `NET.createGeneric()` of the MATLAB Interface to .NET Framework.

```matlab
% Create a .NET list containing strings for the list of variables to be captured
```

Because the abbreviating notation is not available via the `[]` operator in the MATLAB Interface to .NET Framework, in M you have to use the methods defined for collections:
variableList.Add('Model Root/Spring-Mass-Damper System/Out1');
variableList.Add('Model Root/Signal Generator/Out1');

5.4 Exception Handling

Any errors during the execution of a XIL API .NET method are shown by .NET exceptions. Therefore, you are strongly recommended to use comprehensive exception handling. The exceptions defined by ASAM are located in the namespace ASAM.XIL.Interfaces.Testbench.Common.Error.

The following code shows an example of well-implemented exception handling in M:

```
try
demoMAPort = MAPort(configDict);
demoMAPort.Dispose();
catch e
    % Free the allocated memory and any system resources of the MAPort instance
    if (exist('demoMAPort','var'))
        demoMAPort.Dispose();
    end

    if (isa(e,'NET.NetException'))
        if (isa(e.ExceptionObject, 'ASAM.XIL.Interfaces.Testbench.Common.Error.TestbenchPortException'))
            disp('XIL API exception in');
            disp(e.stack);
            fprintf('Code: %d\n', e.ExceptionObject.Code);
            fprintf('CodeDescription: %s\n', char(e.ExceptionObject.CodeDescription));
            fprintf('VendorCode: %d\n', e.ExceptionObject.VendorCode);
            fprintf('VendorCodeDescription: %s\n', char(e.ExceptionObject.VendorCodeDescription));
        else
            disp('.NET exception in');
            disp(e.stack);
            fprintf('Message: %s', char(e.message));
        end
    else
        rethrow(e);
    end
end
```

As you can see in the example, not only XIL API exceptions, but also .NET exceptions must be caught. In addition, some objects of the dSPACE XIL API .NET require a call to the Dispose() method to ensure that in any case (not only when an exception occurs) the memory and, possibly, the system resources associated with the object are released. In a XIL API .NET exception, detailed information about the cause of the error is shown by Code, CodeDescription, VendorCode and VendorCodeDescription.

When you use the MATLAB Interface to .NET Framework, the process is slightly more complicated than in the C# programming language. For example, there is no finally branch in exception handling. First, when an exception occurs in M, the MAPort object must be released by Dispose(). You should do this only if the variable was really set; to ensure this, the test is carried out by exist('demoMAPort', 'var'). Next, a check is executed to see whether a .NET exception or a native M exception occurred. If it is a native M exception, it is forwarded to the caller by rethrow(e). If it is a .NET exception, then process is different, depending on whether a XIL API exception occurred.
5.5 Control Characters in Variable Paths

Variable paths that contain control characters such as \n must be treated separately. MATLAB interprets them during processing and replaces \n with a break, for example, which is why the variable can no longer be found in the TRC file. For the variable to keep the correct form, the control character must be replaced by LF (line feed, ASCII character 10).

Example:

```
% bad
variable = 'Model Root/Reference Generation/Reference\nPos';

% good
variable = ['Model Root/Reference Generation/Reference', char(10), 'Pos'];
```

5.6 Scaling Information in the Variable Description File

The variable description file defines whether a source value on a dSPACE platform can be represented as a converted value on the host PC and how it is to be converted. As an example, a specific parameter of some fixed point data type in the real-time application might be represented as a real world physical value in the test scenario.

MLIB/MTRACE always uses source values when reading, writing or capturing variables. It is not possible to use converted values.

With the dSPACE XIL API, the behavior changed to converted values. The scaling is enabled by default. This means that the value conversion is executed as specified by the scaling attribute in the variable description file. If this behavior is not desired, it can be disabled within the MAPort configuration file. For further information, refer to Software > Test Automation > dSPACE XIL API Implementations > dSPACE XIL API Reference > Model Access Port Implementation > MAPort Configuration > EnableScaling.

6 Reading out Processor Board Information

With MLIB/MTRACE, it is possible to read out information of the selected processor board and the running application. This functionality is not possible with just the dSPACE XIL API .NET server alone.

For this application case, you can use the Tool Automation of ControlDesk 4.x or 5.x or the dSPACE Platform Management API. In this document and in the sample scripts that come along with this document the dSPACE Platform Management API is used.

The dSPACE Platform Management API does not need any ControlDesk 4.x or 5.x licenses. Licenses for the dSPACE Platform Management API are included in the Platform API Package along with the licenses for the dSPACE XIL API .NET server. To install the dSPACE Platform Management API, simply install your dSPACE software using the dSPACE_MasterSetup.exe, located in the root folder of your dSPACE Release DVD. You can also install the dSPACE Python Extensions separately; the latter can be found in the tools folder of the dSPACE Release DVD.

The following example shows one possibility to represent the MLIB/MTRACE functionality:

Code fragment when using MLIB:
% check if the application is running
demoApplName = 'C:\Demos\demo.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

When you use with the XIL API .NET server, some helper scripts are provided with the samples that come along with this document. To accomplish the same tasks as in the MLIB script fragment above, the following code can be used:

% create the platform handler
demoApplName = 'C:\Demos\demo.ppc';
platformIdentifier = 'DS1005';

platformHandler = DSPlatformManagementAPI();
if (platformHandler.IsApplRunning(platformIdentifier))
    applInfo = platformHandler.GetApplInfo(platformIdentifier);
    if strcmpi(demoApplName, 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

7 Complete Sample Scripts

This document is accompanied by a ZIP archive containing complete sample scripts, some helper M scripts, and a Simulink model.

To run the sample scripts, you have to execute some preparatory tasks as described in section 3 Prerequisites/Preparatory Tasks of this document.

The file names of the samples indicate if MLIB/MTRACE or the dSPACE XIL API .NET server is used. The header of each sample gives a short explanation of the use case of the script. The first few lines of code must be adjusted to the dSPACE processor platform and location of the real-time application you use.
Further Reading

For details on the MATLAB Interface to .NET Framework see chapter MATLAB > User’s Guide > External Interfaces > Using .NET Libraries from MATLAB in the MATLAB product documentation.

For details on the dSPACE XIL API .NET server, refer to Software > dSPACE XIL API Implementations in dSPACE HelpDesk.

For details on the ASAM XIL API standard, visit http://www.asam.net (ASAM Automotive Electronics (AE) / AE X-in-the-Loop API V2.0.1).