SIGMASTUDIO SCRIPTING

ANALOG DEVICES, INC.

www.analog.com

KT- (REV 0.0, SEPTEMBER-2010)

Table of Contents

List of Figures

List of Tables

Copyright, Disclaimer & Trademark Statements

Copyright Information

Disclaimer

Analog Devices, Inc. reserves the right to change this product without prior notice. Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use; nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under the patent rights of Analog Devices, Inc.

Trademark and Service Mark Notice

Analog Devices, the Analog Devices logo, Blackfin, SHARC, TigerSHARC, SigmaStudio, CROSSCORE, VisualDSP, VisualDSP++, EZ-KIT Lite, EZ-Extender and Collaborative are trademarks and/or registered trademarks "®" of Analog Devices, Inc.

All other brand and product names are trademarks or service marks of their respective owners.

Trademarks and Service Marks must be reproduced according to ADI's Trademark Usage guidelines. Any licensee wishing to reproduce ADI's Trademarks and Service Marks must obtain and follow these guidelines for the specific marks to be reproduced

1 Introduction

SigmaStudio is a development environment designed for the SigmaDSP family of audio specific signal processors. SigmaStudio has a highly intuitive user interface for Audio system development and tuning.

SigmaStudio defines a Microsoft .NET functional interface, IScripted, which provides control over the most common elements of SigmaStudio. Script files can be created and reused from within the SigmaStudio development environment. SigmaStudioServer is a software automation interface to SigmaStudio that allows external client applications to control and automate SigmaStudio functions from external software.

The overall system and its components are illustrated in Figure 1.

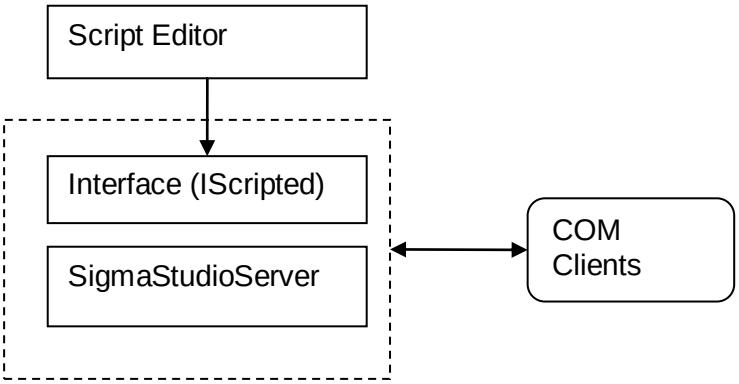

Figure 1

The document is provides information on the Script Editor, the Scripting interface and the SigmaStudio automation server. An example COM client/server configuration with National Instruments LabVIEW is also detailed in the document.

1.1 Scope

This document is intended to assist testing engineers and advanced design engineers. This document provides an overview of SigmaStudio scripting interface and the SigmaStudio automation server.

1.2 Organization of this Guide

Information supplied in this guide is organized as follows:

Section 1: contains the introduction.

Section 2: describes the IScripted Interface

Section 3: describes how to create and run SigmaStudio Scripts.

Section 4: contains information regarding SigmaStudio server

Section 5: details how to use SigmaStudio as a LabVIEW .NET server.

Section 6: describes using SigmaStudio as a MATLAB COM server.

1.3 Acronyms

ADI	Analog Devices Inc.
API	Application Program Interface
ISR	Interrupt Service Routine
COM	Component Object Model

1.4 Version Information

The document describes SigmaStudio 3.2 build 1.

1.4.1 Other Information

For more information on the latest ADI processors, silicon errata, code examples, development tools, system services and devices drivers, technical support and any other additional information, please visit our website at www.analog.com/processors.

2 IScripted Interface

Analog.SStudioScripting.IScripted is contained in a .NET assembly, BaseLib.dll, installed in the SigmaStudio program folder.

2.1 Return Type

The interface defines an integer return type, "HResult", as follows:

```
HResult.S_OK = 0
HResult.E_FAILED = 1
HResult.E_INVALID_ARGS = 2
HResult.E_EXCEPTION = 3
```

2.2 General Functions:

A list of general functions and their prototypes are given below.

2.2.1 Undo

Undo an action in active project file

```
HResult ScriptUndo();
```

Undo an action in specific project file

```
HResult ScriptUndo( string projectName );
"projectName" = An open project file's name or fully qualified path
```

2.2.2 Redo

Redo an action in active project file

```
HResult ScriptRedo();
```

Redo an action in specific project file

```
HResult ScriptRedo( string projectName );
"projectName" = An open project file's name or fully qualified path
```

2.2.3 Pause

Pause script execution for delayMilliseconds

```
HResult ScriptDelay( int delayMilliseconds );
"delayMilliseconds" = Amount of delay in milliseconds
```

2.2.4 Run

Run script

```
HResult ScriptRun( string script );
"script" = script code as a System.String
```

Run script file

```
HResult ScriptRunFile( string scriptFilename );
"scriptFilename" = The fully qualified script file name
```

2.3 Project File Interface:

The following functions can be used to interface with a project file.

2.3.1 Create

Create a new project file

```
HResult ProjectNew();
The function takes no parameter
```

2.3.2 Open

Open a project file from disk

```
HResult ProjectOpen( string filename );
"filename" = A fully qualified file path
```

2.3.3 Save/Save As

Save the Active project file

```
HResult ProjectSave();
The function takes no parameter
```

Save a specific project file

```
HResult ProjectSave( string projectName );
"projectName" = An open project file's name or fully qualified path
```

Save as the Active project file

```
HResult ProjectSaveAs( string saveAsFilename );
"saveAsFilename" = A new file name or fully qualified path
```

Save as a specific project file

```
HResult ProjectSaveAs( string projectName, string saveAsFilename );
"projectName"    = An open project file's name or fully qualified path
"saveAsFilename" = The new file name or fully qualified path
```

2.3.4 Close

Close the Active project file

```
HResult ProjectClose();
```

Close a specific project file

```
HResult ProjectClose( string projectName );
"projectName" = An open project file's name or fully qualified path
```

2.3.5 Set Project as active

Set a project as the active project

```
HResult ProjectSetActive( string projectName );
"projectName" = An open project file's name or fully qualified path
```

2.3.6 Link

Link the active schematic

```
HResult ProjectLink();
The function takes no parameter
```

Link and compile the active schematic

```
HResult ProjectLinkCompile();
The function takes no parameter
```

Link and compile the active schematic

```
HResult ProjectLinkCompile( string projectName );
"projectName" = An open project file's name or fully qualified path
```

Link, compile and download the active schematic

```
HResult ProjectLinkCompileDownload();
The function takes no parameter
```

Link, compile and download a specific project file

```
HResult ProjectLinkCompileDownload( string projectName );
"projectName" = An open project file's name or fully qualified path
```

2.3.7 Others

Set the "New Item Sampling Rate" for the active project

```
HResult DesignSetSamplingRate( int samplingRate );
```

Propagate project sample rate

```
HResult DesignPropagateSamplingRate();
The function takes no parameter
```

Toggle Schematic Freeze On/Off

```
HResult DesignToggleSchematicFreeze( string password );
"password" = Schematic freeze password
```

Set the activate hierarchy board in the current schematic

```
HResult DesignSetActiveBoard( string boardName );
"boardName" = Board name in the active schematic
```

2.4 Register Interface:

Functions for working with the registers and its attributes are listed below.

```
Write data to a register
HResult ICRegisterWrite( string ICName, int writeAddress,
                         int writeNumberBytes, long dataToWrite );
"ICName"           = Friendly name of DSP(IC) to write to
"writeAddress"     = The register address to write
"writeNumberBytes" = Number of bytes in 'dataToWrite' to write to the dsp
"dataToWrite"      = The data to write (long == 64bit max data)

Write data to a register,  specific device address
HResult ICRegisterWrite( string ICName, int deviceAddress, int writeAddress,
                         int writeNumberBytes, long dataToWrite );
"ICName"           = Friendly name of DSP(IC)
"deviceAddress"    = I2C or SPI address
"writeAddress"     = The register address to write
"writeNumberBytes" = Number of bytes in 'dataToWrite' to write to the dsp
"dataToWrite"      = The data to write (long == 64bit max data)

Write data to a register, data specified as a byte array
HResult ICRegisterWrite( string ICName, int writeAddress,
                         int writeNumberBytes, byte[] dataToWrite );
"ICName"           = Friendly name of DSP(IC) to write to
"writeAddress"     = The register address to write
"writeNumberBytes" = Number of bytes in 'dataToWrite' to write to the dsp
"dataToWrite"      = The data to write, byte array of length writeNumberBytes

Write data to a register, specific device address
HResult ICRegisterWrite( string ICName, int deviceAddress, int writeAddress,
```

```
                        int writeNumberBytes, byte[] dataToWrite );
"ICName"           = Friendly name of DSP(IC)
"deviceAddress"    = I2C or SPI address
"writeAddress"     = The register address to write
"writeNumberBytes" = Number of bytes in 'dataToWrite' to write to the dsp
"dataToWrite"      = The data to write, byte array of length writeNumberBytes

Read data from a register, read value returned in method parameter
HResult ICRegisterRead( string ICName, int readAddress, int readNumberBytes,
                        out long bytesRead );
"ICName"          = Friendly name of DSP(IC) to read from
"readAddress"     = The register address to read
"readNumberBytes" = Number of bytes to read
"bytesRead"       = Return data if read is successful

Read data from a register, specific device address
HResult ICRegisterRead( string ICName, int deviceAddress, int readAddress,
                        int readNumberBytes, out long bytesRead );
"ICName"          = Friendly name of DSP(IC)
"deviceAddress"   = I2C or SPI address
"readAddress"     = The register address to read
"readNumberBytes" = Number of bytes to read
"bytesRead"       = Return data if read is successful

Read data from a register, data as byte array
HResult ICRegisterRead( string ICName, int readAddress, int readNumberBytes,
                        ref byte[] bytesRead );
"ICName"          = Friendly name of DSP(IC) to read from
"readAddress"     = The register address to read
"readNumberBytes" = Number of bytes to read
"bytesRead"       = Return data if read is successful

Read data from a register, specific device address
HResult ICRegisterRead( string ICName, int deviceAddress, int readAddress,
                        int readNumberBytes, ref byte[] bytesRead );
"ICName"          = Friendly name of DSP(IC)
"deviceAddress"   = I2C or SPI address
"readAddress"     = The register address to read
"readNumberBytes" = Number of bytes to read
"bytesRead"       = Return data if read is successful

Read data from a register, read value is return type
long ICRegisterRead( string ICName, int readAddress, int readNumberBytes );
"ICName"          = Friendly name of DSP(IC) to read from
"readAddress"     = The register address to read
"readNumberBytes" = Number of bytes to read

Read buffer of data from a register, read value array returned
byte[] ICRegisterRead( string ICName, int readAddress, int readNumberBytes,
                        ref bool bRet );
"ICName"          = Friendly name of DSP(IC) to read from
"readAddress"     = The register address to read
```

```
"readNumberBytes" = Number of bytes to read
"ret"             = Did the read succeed

Write safeload register
HResult ICRegisterSafeload( string ICName, int safeloadRegister,
                            long dataToWrite );
"ICName"           = Friendly name of DSP(IC) to write to
"safeloadRegister" = Regsiter address to safeload
"dataToWrite"      = Data to write to the safeload register (5Bytes)

Write multiple contiguous safeload registers
HResult ICRegisterSafeload( string ICName, int safeloadRegister,
                            int writeNumberBytes, Byte[] dataToWrite );
"ICName"           = Friendly name of DSP(IC) to write to
"safeloadRegister" = Regsiter address to safeload
"writeNumberBytes" = Number of data bytes in dataToWrite
"dataToWrite"      = Data to write to the safeload register (5Bytes)

Write multiple safeload registers
HResult ICRegisterSafeload( string ICName, int[] writeAddresses,
                            int[] writeNumberBytes, byte[] dataToWrite );
"ICName"           = Friendly name of DSP(IC) to write to
"writeAddresses"   = Array of addresses to safeload
"writeNumberBytes" = Write bytes per address (for each writeAddresses entry)
"dataToWrite"      = Data array to write to the safeload registers


Write parameter data, floating point value
HResult ICParameterWrite( string ICName, int writeAddress, float valToWrite )
"ICName"           = Friendly name of DSP(IC) to write to
"writeAddress"     = The register address to begin writing data
"valToWrite"       = Parameter data value to write

Write multiple contiguous parameters
HResult ICParameterWrite( string ICName, int writeAddress,
                          int writeNumParams, float[] valsToWrite );
"ICName"           = Friendly name of DSP(IC) to write to
"writeAddress"     = The register address to begin writing data
"writeNumParams"   = Number of values in valsToWrite
"valsToWrite"      = Parameter data values to write

Write parameter data, specifying target fixed-point format
HResult ICParameterWrite( string ICName, int writeAddress, int intbits,
                          int fracbits, float valToWrite );
"ICName"           = Friendly name of DSP(IC) to write to
"writeAddress"     = The register address to begin writing data
"intbits"          = number of integer (magnitude) bits
"fracbits"         = number of fraction bits
"valToWrite"       = Parameter data value to write

Write parameter data array, specifying target fixed-point format
```

```
HResult ICParameterWrite( string ICName, int writeAddress, int intbits,
                          int fracbits, int writeNumParams,
                          float[] valsToWrite );
"ICName"           = Friendly name of DSP(IC) to write to
"writeAddress"     = The register address to begin writing data
"intbits"          = number of integer (magnitude) bits
"fracbits"         = number of fraction bits
"writeNumParams"   = Number of values in valsToWrite
"valsToWrite"      = Parameter data values to write

Write parameter data via safeload, floating point value
HResult ICParameterSafeload( string ICName, int writeAddress,
                             float valToWrite );
"ICName"           = Friendly name of DSP(IC) to write to
"writeAddress"     = The register address to begin writing data
"writeNumParams"   = Number of values in valsToWrite
"valToWrite"       = Parameter data value to write

Write multiple parameters via safeload, floating point values
HResult ICParameterSafeload( string ICName, int writeAddress,
                             int writeNumParams, float[] valsToWrite );
"ICName"           = Friendly name of DSP(IC) to write to
"writeAddress"     = The register address to begin writing data
"writeNumParams"   = Number of values in valsToWrite
"valsToWrite"      = Parameter data values to write

Safeload parameter data, specifying target fixed-point format
HResult ICParameterSafeload( string ICName, int writeAddress, int intbits,
                             int fracbits, float valToWrite );
"ICName"           = Friendly name of DSP(IC) to write to
"writeAddress"     = The register address to begin writing data
"intbits"          = number of integer (magnitude) bits
"fracbits"         = number of fraction bits
"valToWrite"       = Parameter data value to write

Safeload parameter data array, specifying target fixed-point format
HResult ICParameterSafeload( string ICName, int writeAddress, int intbits,
                             int fracbits, int writeNumParams,
                             float[] dataToWrite );
"ICName"           = Friendly name of DSP(IC) to write to
"writeAddress"     = The register address to begin writing data
"intbits"          = number of integer (magnitude) bits
"fracbits"         = number of fraction bits
"writeNumParams"   = Number of values in valsToWrite
"valsToWrite"      = Parameter data values to write

Read fixed-point parameter data, read value returned as float
float ICParameterRead( string ICName, int readAddress, int intbits,
                       int fracbits );
"ICName"          = Friendly name of DSP(IC) to read from
"readAddress"     = The register address to read
```

```
"intbits"          = number of integer (magnitude) bits
"fracbits"         = number of fraction bits

Read data from a register, data returned as float
HResult ICParameterRead( string ICName, int readAddress, int intbits,
                         int fracbits, out float valRead );
"ICName"          = Friendly name of DSP(IC) to read from
"readAddress"     = The register address to read
"intbits"          = number of integer (magnitude) bits
"fracbits"         = number of fraction bits
"valRead"          = returned read value

Read fixed-point parameter data array, read values returned as float[]
float[] ICParameterRead( string ICName, int readAddress, int intbits,
                         int fracbits, int readNumParams, ref bool bRet );
"ICName"          = Friendly name of DSP(IC) to read from
"readAddress"     = The register address to read
"intbits"          = number of integer (magnitude) bits
"fracbits"         = number of fraction bits
"readNumParams"    = Number of values to read
"bRet"             = result, true if read was successful asdf

Read fixed-point parameter data array, read values returned in float[]
HResult ICParameterRead( string ICName, int readAddress, int intbits,
                    int fracbits, int readNumParams, ref float[] valsRead );
"ICName"          = Friendly name of DSP(IC) to read from
"readAddress"     = The register address to read
"intbits"          = number of integer (magnitude) bits
"fracbits"         = number of fraction bits
"readNumParams"    = Number of values to read
"valsRead"         = returned read values

Load comma-delineated byte data from a text file at a particular register
HResult ICLoadDataFile( string ICName, string filename, int writeAddress );
"ICName"       = Friendly name of DSP(IC) to write to
"filename"     = fully qualified filename of data file to load
"writeAddress" = The register address to begin writing data
```

2.5 Insert:

The functions listed below are used to insert schematic objects into a board.

NOTE: Schematic objects are inserted into the currently selected hierarchy board.

```
Insert an object into the active project, returns object reference
object ObjectInsert( string typeName );
"typeName" = object description (toolbox name)

Insert an object into a specific open project, returns object reference
object ObjectInsert( string projectName, string typeName );
```

```
"projectName" = An open project file's name or fully qualified path
"typeName"    = object description (toolbox name)

Insert an object into the active project at a specific position
object ObjectInsert( string typeName, Point pointInsert );
"typeName" = object description (toolbox name)
"point"    = System.Drawing.Point schematic screen position

Insert an object into a specific open project, at a specific position
object ObjectInsert( string projectName, string typeName, Point point );
"projectName" = An open project file's name or fully qualified path
"typeName"    = object description (toolbox name)
"point"       = System.Drawing.Point schematic screen coordinates

Insert an object into the active project at a specific position
object ObjectInsert( string typeName, int X, int Y );
"typeName" = object description (toolbox name)
"X" & "Y"  = schematic x- and y- coordinates to position the object

Insert an object into a specific open project, at a specific position
object ObjectInsert( string projectName, string typeName, int X, int Y );
"projectName" = An open project file's name or fully qualified path
"typeName"    = object description (toolbox name)
"X" & "Y"     = schematic x- and y- coordinates to position the object

Insert an object into the active project, returns an HResult
HResult ObjectInsert( string typeName, out string objectName );
"typeName"   = object description (toolbox name)
"objectName" = return name of inserted object, null if insertion fails

Insert an object into a specific open project, returns an HResult
HResult ObjectInsert( string projectName, string typeName,
                      out string objectName );
"projectName" = An open project file's name or fully qualified path
"typeName"    = object description (toolbox name)
"objectName"  = return name of inserted object, null if insertion fails

Insert an object into the active project at a specific position
HResult ObjectInsert( string typeName, Point point, out string objectName );
"typeName"   = object description (toolbox name)
"point"      = System.Drawing.Point schematic screen coordinates
"objectName" = return name of inserted object, null if insertion fails

Insert an object into a specific open project, at a specific position
HResult ObjectInsert( string projectName, string typeName,
                      Point point, out string objectName );
"projectName" = An open project file's name or fully qualified path
"typeName"    = object description (toolbox name)
"point"       = System.Drawing.Point schematic screen coordinates
"objectName"  = return name of inserted object or null if insertion fails

Insert an object into the active project at a specific position
```

```
HResult ObjectInsert( string typeName, int X, int Y, out string objectName );
"typeName"   = object description (toolbox name)
"X" & "Y"    = schematic x- and y- coordinates to position the object
"objectName" = return name of inserted object, null if insertion fails

Insert an object into a specific open project, at a specific position
HResult ObjectInsert( string projectName, string typeName, int X, int Y,
                      out string objectName );
"projectName" = an open project file's name or fully qualified path
"typeName"    = object description (toolbox name)
"X" & "Y"     = schematic x- and y- coordinates to position the object
"objectName"  = return name of inserted object, null if insertion fails
```

2.6 Remove:

The functions below are used to remove objects from projects.

```
Delete an object in the active project
HResult ObjectRemove( string objectName );
"objectName" = Name of object to delete

Delete an object from a specific open project
HResult ObjectRemove( string projectName, string objectName );
"projectName" = An open project file's name or fully qualified path
"objectName"  = Name of object to delete

Delete an object in the active project
HResult ObjectRemove( object object );
"object" = Reference to object to delete

Delete an object from a specific open project
HResult ObjectRemove( string projectName, object object );
"projectName" = An open project file's name or fully qualified path
"object"      = Reference to object to delete
```

2.7 Connection:

Use the functions below for connecting an object's input and output in a project.

```
 Connect a pair of objects' output to input in the active project
HResult ObjectConnect( string fromName, int fromOutPinIndex,
                       string toName, int toInPinIndex );
"fromName"        = Name of object to connect FROM
"fromOutPinIndex" = Output pin index to connect FROM (zero-based)
"toName"          = Name of object to connect TO
"toInPinIndex"    = Input pin index to connect TO (zero-based)

Connect a pair of objects' outputs to inputs in a specific open project
```

```
HResult ObjectConnect( string projectName, string fromName,
                       int fromOutPinIndex, string toName, int toInPinIndex);
"projectName"     = An open project file's name or fully qualified path
"fromName"        = Name of object to connect FROM
"fromOutPinIndex" = Output pin index to connect FROM (zero-based)
"toName"          = Name of object to connect TO
"toInPinIndex"    = Input pin index to connect TO (zero-based)

Connect a pair of objects' output to input in the active project
HResult ObjectConnect( object fromObject, int fromOutPinIndex,
                       object toObject, int toInPinIndex );
"fromObject"           = Reference to object to connect FROM
"fromOutPinIndex" = Output pin index to connect FROM (zero-based)
"toObject"        = Reference to object to connect TO
"toInPinIndex"    = Input pin index to connect TO (zero-based)

Connect a pair of objects' output to input in a specific open project
HResult ObjectConnect( string projectName, object fromObject,
                    int fromOutPinIndex, object toObject, int toInPinIndex );
"projectName"     = An open project file's name or fully qualified path
"fromObject"      = Reference to object to connect FROM
"fromOutPinIndex" = Output pin index to connect FROM (zero-based)
"toObject"        = Reference to object to connect TO
"toInPinIndex"    = Input pin index to connect TO (zero-based)
```

2.8 Disconnect:

The functions below are used to disconnect input and output from objects in a project.

```
Disconnect output from input of a pair of objects in the active project
HResult ObjectDisconnect( string fromName, int fromOutPinIndex,
                          string toName, int toInPinIndex );
"fromName"        = Name of object to disconnect FROM
"fromOutPinIndex" = Output pin index to disconnect FROM (zero-based)
"toName"          = Name of object to disconnect TO
"toInPinIndex"    = Input pin index to disconnect TO (zero-based)

Disconnect output from input of a pair of objects in a specific open project
HResult ObjectDisconnect( string projectName, string fromName,
                       int fromOutPinIndex, string toName, int toInPinIndex );
"projectName"     = An open project file's name or fully qualified path
"fromName"        = Name of object to disconnect FROM
"fromOutPinIndex" = Output pin index to disconnect FROM (zero-based)
"toName"          = Name of object to disconnect TO
"toInPinIndex"    = Input pin index to disconnect TO (zero-based)

Disconnect output from input of a pair of objects in the active project
HResult ObjectDisconnect( object fromObject, int fromOutPinIndex,
                          object toObject, int toInPinIndex );
"fromObject"      = Reference to object to disconnect FROM
```

```
"fromOutPinIndex" = Output pin index to disconnect FROM (zero-based)
"toObject"        = Reference to object to disconnect TO
"toInPinIndex"    = Input pin index to disconnect TO (zero-based)

Disconnect output from input of a pair of objects in a specific open project
HResult ObjectDisconnect( string projectName, object fromObject,
                   int fromOutPinIndex, object toObject, int toInPinIndex );
"projectName"     = An open project file's name or fully qualified path
"fromObject"      = Reference to object to disconnect FROM
"fromOutPinIndex" = Output pin index to disconnect FROM (zero-based)
"toObject"        = Reference to object to connect TO
"toInPinIndex"    = Input pin index to disconnect TO (zero-based)
```

2.9 Properties:

The functions below may be used to manage object properties.

```
Manipulate an object's properties or parameters in the active project
HResult ObjectSetProperties( string opcode, string objectName,
                             params object[] propertyParams );
"opcode"         = Opcode of function to perform (see below)
"objectName"     = Name of the object to update
"propertyParams" = Parameters associated with the specified opcode

Manipulate an object's properties or parameters in the active project
HResult ObjectSetProperties( string opcode, object object,
                             params object[] propertyParams );
"opcode"         = Opcode of function to perform (see below)
"object"         = Reference to object to update
"propertyParams" = Parameters associated with the specified opcode

Manipulate an object's properties or parameters in a specific open project
HResult ObjectSetProperties( string projectName, string opcode,
                         string objectName, params object[] propertyParams );
"projectName"    = An open project file's name or fully qualified path
"opcode"         = Opcode of function to perform (see below)
"objectName"     = Name of the object to update
"propertyParams" = Parameters associated with the specified opcode

Manipulate an object's properties or parameters in a specific open project
HResult ObjectSetProperties( string projectName, string opcode,
                             object object, params object[] propertyParams );
"projectName"    = An open project file's name or fully qualified path
"opcode"         = Opcode of function to perform (see below)
"object"         = Reference to object to update
"propertyParams" = Parameters associated with the specified opcode
```

The property interfaces require an opcode (Operation Code), which specifies the type of operation to perform. Relevant opcodes depend on the type of object. Some opcodes apply to all objects (e.g.

setPosition and setName); others are specific to particular object categories. Essential opcodes are listed in the table below:

OPcode	PropertyParams	
"setPosition"	1. System.Drawing.Point	Screen position at which to center the object.
"setName"	1. System.String	New name for object (must be unique).
"changeIC"	1. System.String	Name of IC to associate with the algorithm.
	2. int (optional)	Index of Algorithm to change.
"addAlgorithm"	1. Sytem.String (optional)	Name of IC to associate with the algorithm.
	2. Sytem.String (optional)	Name of algorithm to add.
"removeAlgorithm"	NONE	
"growAlgorithm"	1. int	Index of algorithm to grow
	2. int	Amount to grow algorithm
"reduceAlgorithm"	1. int	Index of algorithm to reduce
	2. int	Amount to reduce algorithm
"setSamplingRate"	1. int	New sampling rate
"setControlValue"	1. int	Index of algorithm
	2. int	Repeat Index (Grow index)
	3. System.String	Control value name***
	4. System.object	Value to set

Table 1: Opcode and Property Parameters

NOTE: Control value names are not exposed in the user interface and vary among toolbox blocks. The following is a list of standard names; please contact ADI for additional information and assistance.

Gain, LevelL, LevelH, TimeConstant, HoldTime, DecayTime, Damping, OnOff, SoftKnee, PostGain, RMSvalue, Threshold, DelaySamples, MaxDelaySamples

Boost, Step, A1, A2, B0, B1, B2.

3 Creating and Running SigmaStudio Scripts

SigmaStudio scripts allow project files to be created and manipulating using textual commands. The scripting interface is defined in the C# language and supports scripts written in either C# or Visual Basic languages. No previous knowledge of C# is required to write simple SigmaStudio scripts.

SigmaStudio scripts can be created in any text editor or within SigmaStudio using the "Script Editor" tool. The Script Editor provides IntelliPrompt display of the script interfaces and syntax highlighting functionality. Script files can be saved as text files (*.txt) or SigmaStudio Script files (*.sss). Scripts are loaded and run from the Script Editor window.

3.1 Opening the Script Editor

In SigmaStudio, click on Tools→Script in main menu to start the Script Editor shown in Figure 2.

To create a new script file, open the Script Editor window and click File → New (or CTRL + N). This creates a new script file and automatically inserts "// #LANGUAGE# C#" on the first line. This identifies the script language as C# (c-sharp) the default SigmaStudio scripting language. Visual Basic scripts are also supported. The language identifier is optional for C# scripts but is required for Visual Basic scripts.

C# (c-sharp) script language identifier — #LANGUAGE# C#

Visual Basic script language identifier — #LANGUAGE# VB

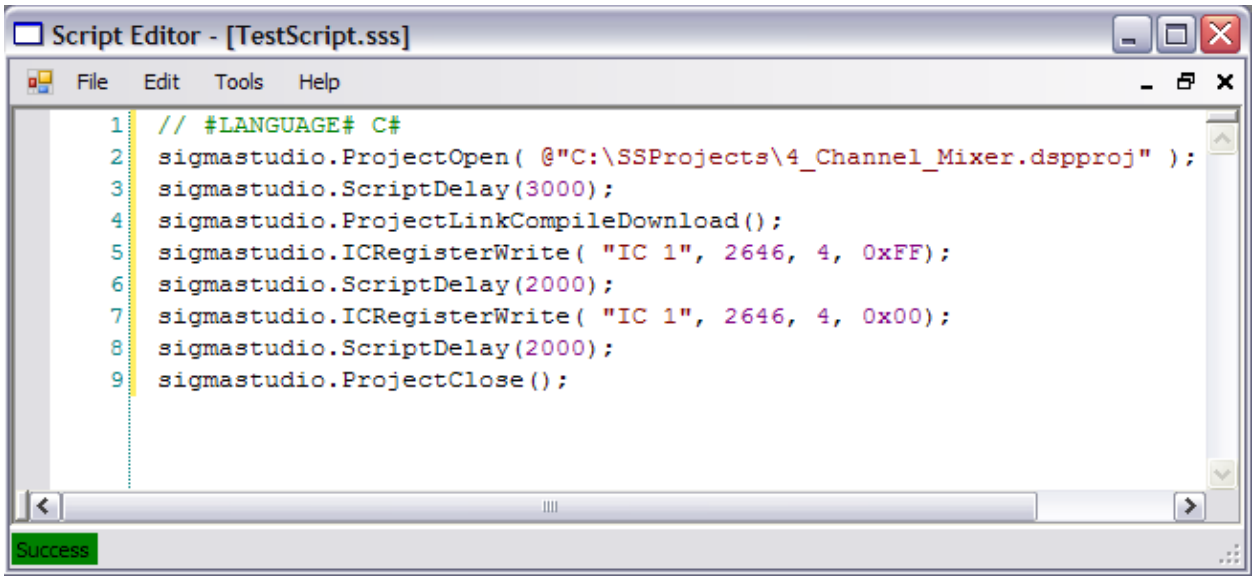

Figure 2: Script Editor

3.2 Writing a Script

To access the IScripted interface list, type "sigmastudio" (or optionally "ss") followed by a period in the Script Editor window. This will display the IntelliPrompt window listing all available IScripted methods. ("sigmastudio" and "ss" are public references to the SigmaStudio IScripted interface, see the following usage example).

An example script is given below:

```
// #LANGUAGE# C#
sigmastudio.ProjectNew();

sigmastudio.ObjectInsert( "USBSerialConv" );
sigmastudio.ObjectInsert( "AD1940" );
sigmastudio.ObjectConnect( "USBSerialConv", 0, "IC 1", 0 );

sigmastudio.ObjectInsert( "Input" );
sigmastudio.ObjectInsert( "Output" );
sigmastudio.ObjectInsert( "Output" );
sigmastudio.ObjectConnect( "Input1", 0, "Output1", 0 );
sigmastudio.ObjectConnect( "Input1", 1, "Output2", 0 );

sigmastudio.ProjectLinkCompileDownload();

sigmastudio.ProjectSaveAs( @"C:\SStudioProjects\SampleScript.dspproj" );
sigmastudio.ProjectClose();
```

This example creates a new project, inserts an AD1940 processor and a USB communication channel, and connects them with a wire. Next, it inserts an input object and 2 output objects, connecting the 2 input pins to the output object pins. The project is then linked, compiled, downloaded, saved to disk and closed.

3.3 Running a Script

To run a SigmaStudio Script, in the Script Editor Window, click on main menu Tools > RunScript or press "F5", see Figure 3.

Figure 3: Running Script

If there are any errors in the script code, a dialog detailing the errors is displayed and "Script Failure" is shown in the status bar. If the script successfully compiles and runs, "Success" is shown in the Script Editor status bar.

Figure 4: Script Error

3.4 Object Names

To reference schematic objects contained in hierarchy boards, the complete object name must be used. The "complete name" consists of the object's name preceded by the names of all parent Hierarchy boards separated by periods ('.').

For example, in Figure 5 "Filter2" is contained in a Hierarchy board named "Board1". The complete name of this object is "Board1.Filter2".

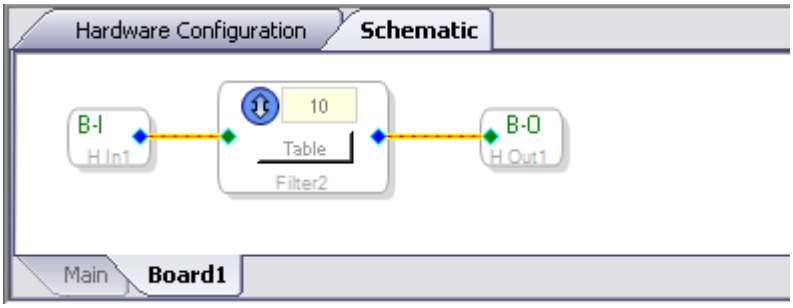

Figure 5: Script Object Name Usage

The following script would remove the Filter2 object:

```
sigmastudio.ObjectRemove( "Board1.Filter2" );
```

In the next example (Figure 6), Board2 is contained within Board1, so the full name of object in Board2 includes both board names, "Board1.Board2.Gen 1st Order1".

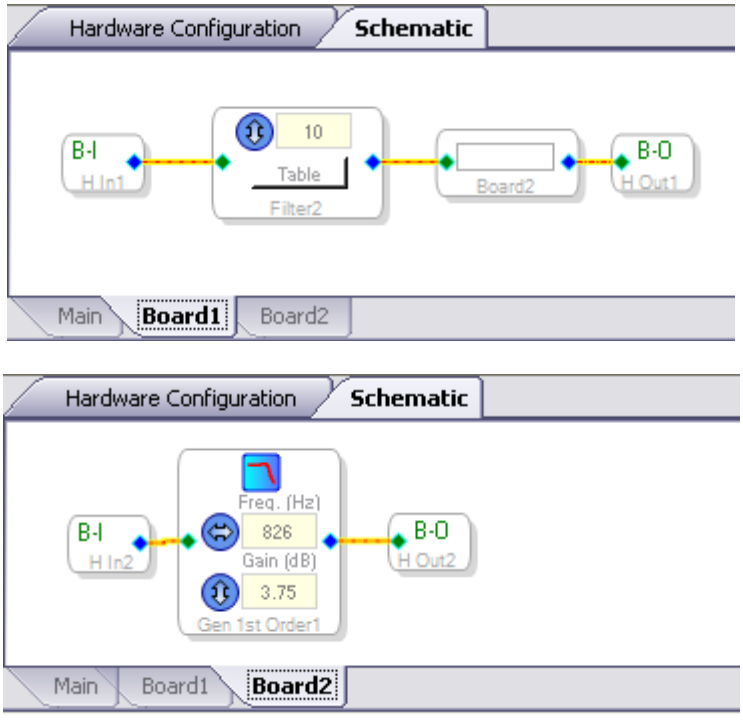

Figure 6: Script Object Name for Board

3.5 Advanced Script Support

Scripts are not limited to the IScripted interface functions. Scripts can take advantage of the C# language and some elements of the .NET framework. A sample script to add object to a schematic, modify its attributes and interconnect them is given below.

```
// #LANGUAGE# C#
ss.ProjectOpen( @"C:\SStudioProjects\SampleScript.dspproj");
ss.ObjectDisconnect( "Input1", 0, "Output1", 0 );
ss.ObjectDisconnect( "Input1", 1, "Output2", 0 );

try
{
  int nNumFilters = 4;
  for (int i = 0; i < nNumFilters; ++i)
  {
        object oFilter = ss.ObjectInsert( "General (2nd order)" );
        if (null != oFilter)
        {
          string strNewName = "Filter_" + (i + 1);
          ss.ObjectSetProperties( "setName", oFilter, strNewName );
          ss.ObjectSetProperties( "addAlgorithm", strNewName, "IC 1",
                                  "2 Channel - Single Precision" );
        }
        else
        {
          throw new Exception( "ln 11" );
        }
  }

  HResult hr = HResult.S_OK;
  for (int ixPin = 0; ixPin < 2; ++ixPin)
  {
        if (HResult.S_OK != sigmastudio.ObjectConnect( "Input1",ixPin,
                                                       "Filter_1", ixPin ))
          throw new Exception( "ln 28" );
        if (HResult.S_OK != sigmastudio.ObjectConnect( "Filter_1", ixPin,
                                                       "Filter_2", ixPin ))
          throw new Exception( "ln 31" );
        if (HResult.S_OK != sigmastudio.ObjectConnect( "Filter_2", ixPin,
                                                       "Filter_3", ixPin ))
      throw new Exception( "ln 34" );
        if (HResult.S_OK != sigmastudio.ObjectConnect( "Filter_3", ixPin,
                                                       "Filter_4", ixPin ))
          throw new Exception( "ln 37" );
  }

  if (HResult.S_OK != sigmastudio.ObjectConnect( "Filter_4", 0,
                                                 "Output1", 0 ))
     throw new Exception( "ln 42" );
  if (HResult.S_OK != sigmastudio.ObjectConnect( "Filter_4", 1,
```

```
                                                    "Output2", 0 ))
            throw new Exception( "ln 45" );
}
catch(Exception e)
{
    System.Windows.Forms.MessageBox.Show( "FAILURE: " + e.ToString() );
}
```

4 SigmaStudioServer

You can use an automation client to access the objects, properties, methods, and events associated with the SigmaStudioServer interface. SigmaStudioServer is a .NET server as well as an ActiveX server.

To access the server interface, launch SStudio.exe and your client application on the same machine, and then point your client application to Analog.SigmaStudioServer.dll which is installed along side SStudio.exe in the SigmaStudio program folder.

Note: The default installation location is: C:\Program Files\Analog Devices Inc\SigmaStudio

4.1 SigmaStudioServer Commands (ISigmaStudioServer)

Once you have launched the server interface, the following commands are available for use.

```
Open a SigmaStudio project file (*.dspproj)
   Bool OPEN_PROJECT( string fullyQualifiedFileName );

Compile (compile,link,download) the active SigmaStudio project
   bool COMPILE_PROJECT();

Close the active SigmaStudio project
   bool CLOSE_PROJECT();

Write to IC register (active project must be compiled and downloaded)
   bool REGISTER_WRITE( string ICName, int writeAddess, int numBytesToWrite,
                        int dataToWrite );

Write data array to IC regster(s)
   bool REGISTER_WRITE_ARRAY( string ICName, int writeAddress,
                              int numBytesToWrite, byte[] aDataToWrite );

Read from an IC register (active project must be compiled and downloaded)
   long REGISTER_READ( string ICName, int readAddess, int readNumberBytes );

Read array of data from an IC register(s)
   byte[] REGISTER_READ_ARRAY( string ICName, int readAddress,
                               int readNumberBytes );

Safeload write data to IC register
   bool REGISTER_SAFELAOD_WRITE( string ICName, int writeAddress,
                                 int numBytesToWrite, int dataToWrite );

Safeload write data array to IC regster(s)
```

```
   bool REGISTER_SAFELAOD_ARRAY( string ICName, int writeAddress,
                                 int numBytesToWrite, byte[] aDataToWrite );

Run SigmaStudio script
   bool RUN_SCRIPT( string script );

Open and Run SigmaStudio script file
   bool RUN_SCRIPT_FILE( string fullyQualifiedFileName );
```

5 Using SigmaStudio as a LabVIEW .NET Server

To use SigmaStudio as a LabVIEW automation client, both applications must be installed and running on the same machine. In this configuration, LabVIEW is a .NET automation client and SigmaStudio (via SigmaStudioServer) becomes a .NET automation server.

To create a virtual instrument to control SigmaStudio, follow the steps given below:

1. Launch LabVIEW and SigmaStudio on the same machine, and then open a new VI file in LabVIEW
2. Open the .NET palette to access the .NET objects.
3. Insert a Constructor Node which will open the Select .NET Constructor dialog box.
4. Click the Browse… button.
5. Navigate to the SigmaStudio installation directory, select Analog.SigmaStudioServer.dll and press OK.
6. Next choose SigmaStudioServer from the object list, SigmaStudioServer() should be displayed in the constructors list.
7. Click OK to select the SigmaStudioServer constructor.
8. Insert an Invoke Node and connect the constructor Node to it.
9. Click on Method to display and select the accessible SigmaStudioServer commands.

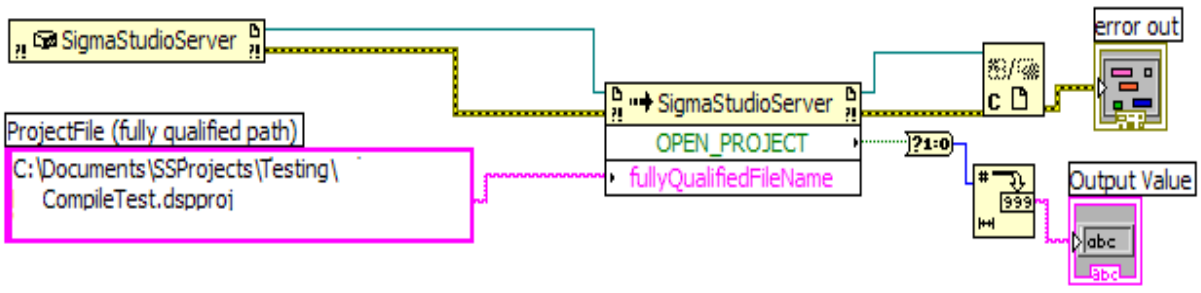

Figure 7: SigmaStudio Server.

Refer to the LabVIEW online help and tutorials for more information; the "Using .NET with LabVIEW" online help topic is a good place to start.

6 SigmaStudio as a MATLAB® COM server

To use MATLAB as a SigmaStudio server client, both applications must be installed and running on the same machine.

In this configuration, MATLAB is a COM automation client and SigmaStudio (via SigmaStudioServer) becomes a COM automation serverFor COM operation, the SigmaStudioServer must first be registered as a COM object.

To register SigmaStudioServer as a COM object:

1. Locate the Microsoft .NET "regAsm.exe" application. If it's not installed, download and install the ".NET Framework 2.0 Software Development Kit (SDK)", available from Microsoft.

2. Locate the framework installation directory.For example:
   C:\Windows\Microsoft.NET\Framework\v2.0.50727

3. From the windows command prompt execute regAsm.exe with the "/codebase" argument to register the assembly file, "Analog.SigmaStudioServer.dll", for COM interoperability:
   regAsm.exe "C:\Program Files\Analog Devices Inc\SigmaStudio 3.1\Analog.SigmaStudioServer.dll" /codebase
   NOTE: For Windows Vista or Window7, use "Run as Administrator" open for the command prompt cmd.exe.

4. Open the MATLAB application. Before continuing the MATLAB path must be updated to include the SigmaStudioServer installation directory. Set the MATLAB working directory to the SigmaStudio installation directory or add the SigmaStudio directory to the MATLAB environment path. For example:
   cd 'C:\Program Files\Analog Devices Inc\SigmaStudio 3.0'

To create a SigmaStudioServer process:

1. Create a SigmaStudio COM server using MATLAB's actxserver function.
   SS = actxserver('Analog.SigmaStudioServer.SigmaStudioServer');

2. Use the interface function to see a list of the exposed COM interfaces, the ISigmaStudioServer interface.
   SSinterfaces = SS.interfaces;
   SSinterfaces SSinterfaces = 'ISigmaStudioServer'

3. Create a handle to the interface with the invoke function.

sssvr = SS.invoke('ISigmaStudioServer');

4. Use the invoke function on the interface handle to get a list of the interface methods.
   sssvr.invoke
   CLOSE_PROJECT = bool CLOSE_PROJECT(handle)
   COMPILE_PROJECT = bool COMPILE_PROJECT(handle)
   OPEN_PROJECT = bool OPEN_PROJECT(handle, string)
   REGISTER_READ = int32 REGISTER_READ(handle, string, int32, int32)
   REGISTER_WRITE = bool REGISTER_WRITE(handle, string, int32, int32, int32)
   RUN_SCRIPT = bool RUN_SCRIPT(handle, string)
   RUN_SCRIPT_FILE = bool RUN_SCRIPT_FILE(handle, string)

5. Call the interface methods from the interface handle
   sssvr.OPEN_PROJECT('C:\Projects\SigmaFlow.dspproj');