SIGMASTUDIO - SCRIPTING GUIDE

Document Status	Approved
Approved by	ASH

ANALOG DEVICES, INC.

www.analog.com

Revision List

Table 1: Revision List

Revision	Date	Description
2.1	01.06.2015	Draft Version for M3
2.2	08.06.2015	Incorporated review comments
2.3	12.06.2016	Incorporated review comments
3.0	13.07.2015	Approved and Base-lined for 3.2.0
3.1	06.10.2015	Updated for Release 3.4.0
4.0	07.10.2015	Approved and Base-lined for 3.4.0
4.1	16.03.2018	Updated for Release 4.0.0
5.0	21.03.2018	Approved and Base-lined for 4.0.0
5.1	22.12.2020	Addressed review comments for release 4.6.0
6.0	23.12.2020	Approved and baselined for release 4.6.0
6.1	06.04.2022	Updated for release 4.7.0
7.0	13.04.2022	Approved and baselined for release 4.7.0

Copyright, Disclaimer Statements

Copyright Information

Disclaimer

Table of Contents

List of Figures

List of Tables

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.

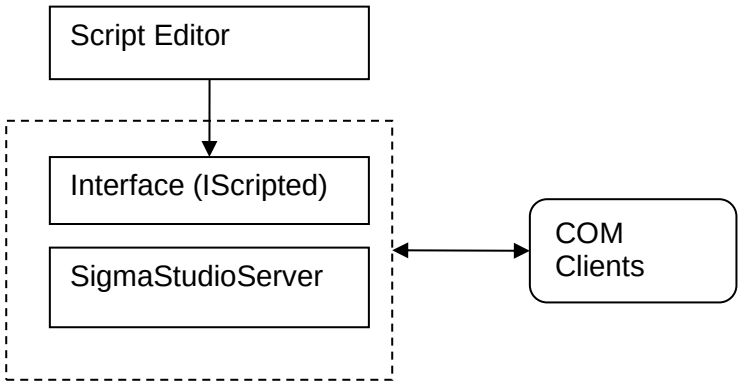

Figure 1: Overall Automation System

The document provides information on the Script Editor, the Scripting interface and the SigmaStudio automation server.

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 the Guide

Information supplied in this guide is organized as follows:

Section 1 : contains the introduction

Section 2 : contains the specifications of the product

Section 3 : describes the IScripted Interface

Section 4 : describes how to create and run SigmaStudio scripts

Section 5 : describes the SigmaStudio server APIs

2 Specifications

2.1 Version Information

The SigmaStudio software version to be used is 4.7.0 .

3 IScripted Interface

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

3.1 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
```

3.2 General Functions

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

3.2.1 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
```

3.2.2 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
```

3.2.3 Pause

```
Pause script execution for delay Milliseconds
HResult ScriptDelay( int delayMilliseconds );
"delayMilliseconds" = Amount of delay in milliseconds
```

3.2.4 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
```

3.3 Project File Interface

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

3.3.1 Create

```
Create a new project file
HResult ProjectNew();
The function takes no parameter
```

3.3.2 Open

```
Open a project file from disk
HResult ProjectOpen( string filename );
"filename" = A fully qualified file path
```

3.3.3 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
```

3.3.4 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
```

3.3.5 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
```

3.3.6 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
```

3.3.7 Others

```
Set the "New Item Sample Rate" (Schematic Sampling Rate) for the active project
HResult DesignSetSamplingRate( int samplingRate );

Propagate Schematic Sampling 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

Export the system files of the active Schematic
HResult ProjectExportSystemFiles ( string fullyQualifiedFileName );
```

```
"fullyQualifiedFileName" = fully qualified path of the system file without the
extension

Build Plug-In DLL using Algorithm Designer
HResult BuildExternalModule ( string ICName, string fullyQualifiedProjectName,
string fullyQualifiedLibraryName );
"ICName" = Friendly name of DSP(IC)
"fullyQualifiedProjectName" = fully qualified path of designer project file
"fullyQualifiedLibraryName" = fully qualified path of the output DLL

Set the "Schematic Block Size" of IC and Algorithms belonging to the IC for the
active project
HResult DesignSetBlockSize(string ICName, int nSize);
"ICName" = Friendly name of DSP(IC)
"nSize" = The Schematic Block Size given as 'nSize' will be set to all
Modules/Algorithms belonging to the IC corresponding to the 'ICName'. The
Schematic Block Size (nSize) should be greater than 8 and a multiple of 8.
```

3.4 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
"dataToWrite"      = 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
```

3.5 Insert

The functions listed below are used to insert Schematic objects into a 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)
"pointInsert"     = 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
```

NOTE: Schematic objects are inserted into the currently selected hierarchy board of 'Schematic' tab by default. Use 'BlockObjectInsert' function instead of 'ObjectInsert' function to insert Schematic objects into the 'Block Schematic' tab.

3.6 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
```

3.7 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)
```

3.8 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)
```

3.9 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 be performed. 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 centre 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 Module Sampling Rate
"setBlockSize"	1. int	New Source Module Block Size
"setControlValue"	1. int	Index of Algorithm
	2. int	Repeat Index (Grow index)
	3. System.String	Control value name***
	4. System.object	Value to set
"setOutputBufferSize"	1. int	New Output Buffer Size in bytes
"setPCMxOutputBlockSize"	1. int	New PCMx Output Module Block Size

Table 2: 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.

3.10 Settings

The function below may be used to manage SigmaStudio settings.

```
Manipulate SigmaStudio settings in the active project
HResult DesignSettings(string cmd, params object[] arg);
"cmd" = Command opcode of setting to modify
"arg" = Modified value of the setting. This should be of string datatype
```

Command	Argument
"XMLOnly"	1. "true" – Only XML file will get generated upon exporting
	2. "false" – Non XML system files will also get generated upon exporting
"SchematicTag"	"String" tag to be associated with the Schematic xml file. e.g. "SigmaStudioforSHARC - Version x_y_z[1]"
"Pre-build"	The command to be executed before the Schematic is compiled.. e.g. "ssTestSettings.bat arg1"
"Post-build"	The command to be executed after the Schematic is compiled. e.g. "ssTestSettings.bat arg1"
"Pre-export"	The command to be executed before the export of a Schematic. e.g."ssTestSettings.bat arg1"
"Post-export"	The command to be executed after the export of a Schematic. e.g."ssTestSettings.bat arg1"

[1] x.y.z denotes the version of SigmaStudio for SHARC. Refer Release Notes for version information of SigmaStudio for SHARC

"Auto-export"	1. "false", "D:\\test\\autoexport" – Auto export of system files after compile is disabled. The second string is the path where the export files should be generated. The path entered should be a valid one
	2. "true", "D:\\test\\autoexport" – Auto export of system files after compile is disabled. The second string is the path where the export files should be generated. The path entered should be a valid one.
"ToolSel"	This should be the full name of the tool- chain (as available in the Tools -> Settings -> SHARC -> Tool-chain list) to be used for Schematic compilation. e.g. "CrossCore Embedded Studio for Analog Devices Processors v1.0.3" "VisualDSP++ 5.1"

Table 3: Command and Arguments for Settings API

```
Manipulate properties in the IC Control window
HResult SetICControlProperties(string ICName, string cmd, object arg);
"ICName" = Friendly name of DSP(IC) to be modified
"cmd" = Command opcode of property to modify
"arg" = Modified value of the setting
```

Command	Type	Argument
"MaxInChannels"	int	Number of input channels
"MaxOutChannels"	int	Number of output channels
"MasterBypass"	bool	true - Master Bypass enabled false - Master Bypass disabled
"DefaultCore"	bool	true - Core 1 enabled false - Core 2 enabled
"TargetBuildMode"	bool	true - Debug mode enabled false – Release mode enabled
"BlockSize"	int	Blocksize value (8 or multiple of 8)

"LoadAppDXE"	string	The Application DXE, to be selected, with full path

Table 4: Command and Argument for Modify IC Control Window Properties API

3.11 Plug-Ins

The function given below may be used to manipulate Plug-Ins.

```
Enable or disable a Plug-In added to the SigmaStudio Add-Ins window
HResult ModifyPlugInState ( string plugin, bool state);
"plugin" = name of the Plug-In as string
"state"  = "True" or "False" indicating whether to 'enable' or 'disable' Plug-In
```

The first argument, which is the name of the Plug-In, can either be the full path of the Plug-In DLL as listed in the Add-Ins list or only the file name of the Plug-In DLL. SigmaStudio will first try to match the argument string against full path of the Plug-Ins in the Add-Ins list. If a match is not found, the string is matched against the file name of the Plug-Ins. If multiple Plug-Ins with the name passed as argument are present in the Add-Ins list, only the first occurrence in the list is modified by this function. The Plug-In name is case insensitive.

3.12 Float Packets Interface

3.12.1 ICParameterSafeload

The function has two overloaded methods as shown in prototype-1 and prototype-2 given below. This is an API to handle floating point packets and hence is applicable for SHARC only.

Prototype-1

```
HResult ICParameterSafeload ( string ICname,
                              int32 WriteAddress,
                              single DataToWrite)
```

Description

The method sends a parameter data to the address specified by the write address for the specified IC. The parameters will be loaded to the target using the "Safeload" mechanism.

Parameters

Name: ICname

Type: String

Description: Name of the IC to which data is to be sent

Name: `WriteAddress`

Type: `int32`

Description: Offset address location to which the parameter needs to be written

Name: `DataToWrite`

Type: `Single`

Description: Single precision floating point parameter value to be written to target memory.

Prototype-2

```
HResult ICParameterSafeload ( string ICname,
                              int32 WriteAddress,
                              int32 WriteNumParams,
                              single[] DataToWrite)
```

Description

The method sends an array of parameter data of size “WriteNumParams” to the address specified by the write address for the specified IC. The parameters will be loaded to the target using the “Safeload” mechanism.

Parameters

Name: `ICname`

Type: `String`

Description: Name of the IC to which data is to be sent

Name: `WriteAddress`

Type: `int32`

Description: Offset address location to which the parameter needs to be written

Name: `WriteNumParams`

Type: `int32`

Description: Number of parameters to be written to target parameter memory

Name: DataToWrite

Type: Single

Description: Single precision floating point array of parameter values to be written to target parameter memory.

3.12.2 ICParameterWrite

The function has two overloaded methods as shown in prototype-1 and prototype-2 given below. This is an API to handle floating point packets and hence is applicable for SHARC only.

Prototype-1

```
HResult ICParameterWrite (    string ICname,
                              int32 WriteAddress,
                              single DataToWrite)
```

Description

The method sends a parameter data to the address specified by the write address for the specified IC. The parameters will be loaded to the target without “Safeload” mechanism.

Parameters

Name: ICname

Type: String

Description: Name of the IC to which data is to be sent

Name: WriteAddress

Type: int32

Description: Offset address location to which the parameter needs to be written

Name: DataToWrite

Type: Single

Description: Single precision floating point parameter value to be written to target memory.

Prototype-2

```
HResult ICParameterWrite (    string ICname,
                              int32 WriteAddress,
                              int32 WriteNumParams,
                              single[] DataToWrite)
```

Description

The method sends an array of parameter data of size “WriteNumParams” to the address specified by the write address for the specified IC. The parameters will be loaded to the target as a block without using the “Safeload” mechanism.

Parameters

Name: ICname

Type: String

Description: Name of the IC to which data is to be sent

Name: WriteAddress

Type: int32

Description: Offset address location to which the parameter needs to be written

Name: WriteNumParams

Type: int32

Description: Number of parameters to be written to target parameter memory

Name: DataToWrite

Type: Single

Description: Single precision floating point array of parameter values to be written to target parameter memory.

3.12.3 ICParameterRead

This is an API to handle floating point packets and hence is applicable for SHARC only.

Prototype

```
float ICParameterRead ( string ICname,
                        int32  readAddress)
```

Description

The method reads a parameter data from the address specified by the read address for the specified IC.

Parameters

Name: ICname

Type: String

Description: Name of the IC from which data is to be read

Name: readAddress

Type: int32

Description: Offset address location from parameter memory from where the parameter needs to be read

3.13 SHARCTargetBoot

This is applicable only for the SHARC target, where booting using a Loader File is applicable.

Prototype

```
HResult SHARCTargetBoot (string    ICName,
                         string    fileName,
                         ArrayList bootArgs)
```

Description

The method is used to boot the SHARC Target with a Loader File.

Parameters

Name: ICname

Type: String

Description: Name of the IC which should be booted

Name: Filename

Type: String

Description: Full path of the LDR file

Name: `bootArgs`

Type: `ArrayList`

Description: Arguments for boot configuration

ArrayList with the boot option (2, 3, 4, 5 or 10) as the first entry

2: Analog-In

3: Digital-In

4: Digital-Out alone

5: Analog\Digital Co-existence

10: Analog\Digital Co-existence (Digital Clock)

3.14 SHARCReadMIPS

This API is applicable only for the SHARC target, to get the MIPS information from the target.

Prototype

```
float SHARCReadMIPS (string    ICName)
```

Description

The method is used to Read MIPS consumed by the SHARC Target hardware.

Parameters

Name: `ICname`

Type: `String`

Description: Name of the IC from which to read the MIPS

3.15 SHARCReadLibVersion

This API is applicable for SHARC target only, to fetch the target library version from the target.

Prototype

```
uint SHARCReadLibVersion (string    ICName)
```

Description

The method is used to read version number of the Target Library running on SHARC Target processor.

Parameters

Name: `ICname`

Type: `String`

Description: Name of the IC from which the version should be obtained

4 Creating and Running SigmaStudio Scripts

SigmaStudio scripts allow project files to be created and manipulated 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.

4.1 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

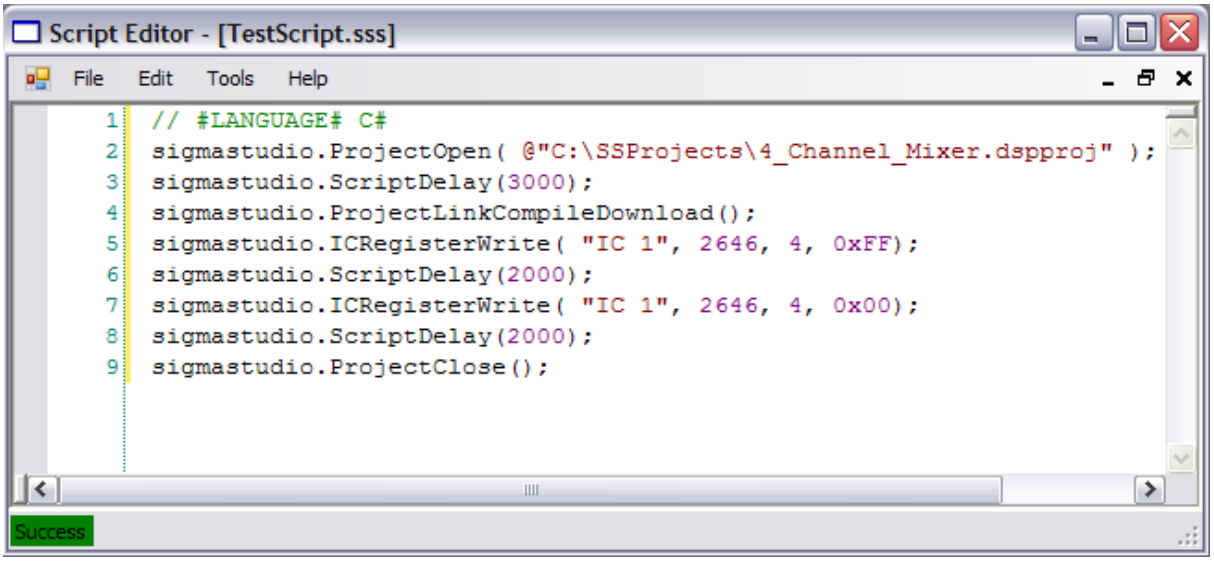

Figure 2: Script Editor

4.2 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( "USBi" );
sigmastudio.ObjectInsert( "ADSP-214xx" );
sigmastudio.ObjectConnect( "USB Interface", 0, "IC 1", 0 );

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

sigmastudio.ProjectLinkCompileDownload();

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

This example creates a new project, inserts an ADSP-214xx 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.

4.3 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.

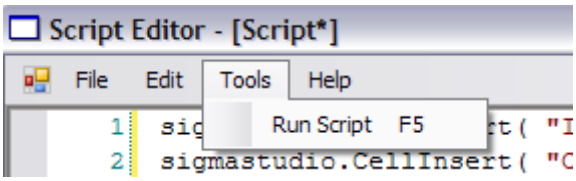

Figure 3: Running Script

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

4.4 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”.

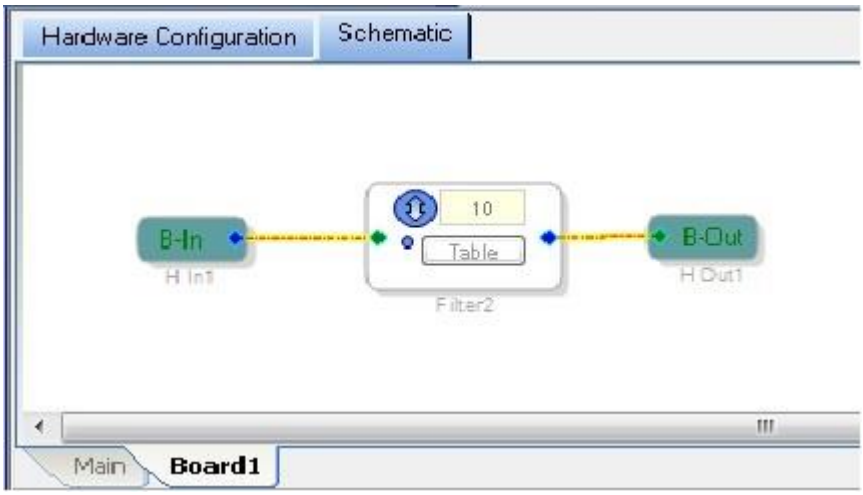

Figure 5: Script Object Name Usage

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”.

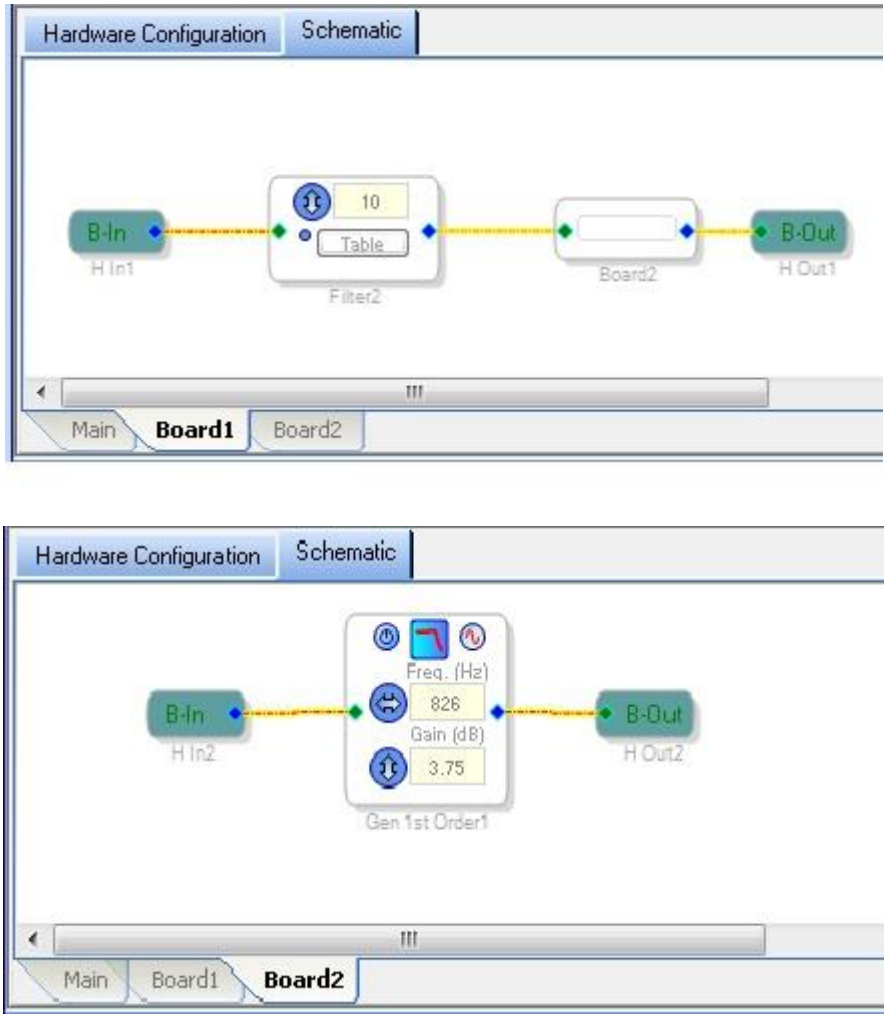

Figure 6: Script Object Name for Board

4.5 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( "Audio Input1", 0, "Output1", 0 );
ss.ObjectDisconnect( "Audio 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( "Audio 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() );
}
```

5 SigmaStudioServer

An automation client can be used 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, SStudio.exe and the client application have to be launched on the same PC, and then the client application has to point to the Analog.SigmaStudioServer.dll which is installed alongside SStudio.exe in the SigmaStudio program folder.

5.1 SigmaStudioServer Commands (ISigmaStudioServer)

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

```
Start a new SigmaStudio project
bool NEW_PROJECT();

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

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

Link the active SigmaStudio project
bool LINK();

Compile the active SigmaStudio project
bool COMPILE();

Download the active SigmaStudio project
bool DOWNLOAD();

Link/Compile/Download the SigmaStudio project, specified as an .NET CLR
compatible System.String
bool COMPILE_PROJECT_NAME( string projectName );

Close the active SigmaStudio project
bool CLOSE_PROJECT();

Close SigmaStudio project by project name
bool CLOSE_PROJECT_NAME( string projectName );

Save the active SigmaStudio project
bool SAVE_PROJECT();

Save the active SigmaStudio project under the mentioned name
bool SAVEAS_PROJECT( string saveAsFileName );

Save a SigmaStudio project to file
bool SAVE_PROJECT_NAME( string projectName );
```

```
Save a SigmaStudio project under the mentioned name
bool SAVEAS_PROJECT_NAME( string projectName, string saveAsFileName );

Set a specified SigmaStudio project as the active project
bool SET_ACTIVE_PROJECT( string projectName );

Set a specified SigmaStudio board as the active board
bool SET_ACTIVE_BOARD( string boardName );

Set Schematic Sampling Rate for the SigmaStudio project
bool SET_SAMPLING_RATE( int samplingRate );

Propagate the Schematic Sampling Rate across the Schematic
bool PROPAGATE_SAMPLING_RATE();

Set Block Size for the SigmaStudio project
bool SET_BLOCK_SIZE ( string ICname, int nSize );

Toggle the Freeze/Unfreeze state of the schametic
bool TOGGLE_FREEZE_SCHEMATIC( string password );

Delay a SigmaStudio script for period specified as an argument
bool DELAY_SCRIPT( int delayMilliseconds );

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 );

Write value to a Parameter, SStudio handles conversion
bool PARAMETER_WRITE( string ICName, int writeAddress, int intbits,
                      int fracbits, float valToWrite );

Write parameters values, SStudio handles conversion
```

```
bool PARAMETER_WRITE_ARRAY( string ICName, int writeAddress, int intbits,
                            int fracbits, int numValsToWrite,
                            float[] aValsToWrite );

Wrate value to a Parameter as float
bool PARAMETER_WRITE_FLOAT( string ICName, int writeAddress, float valToWrite );

Write parameter values as float
bool PARAMETER_WRITE_ARRAY_FLOAT( string ICName, int writeAddress,
                                  int numValsToWrite, float[] aValsToWrite );

Read parameter value from a register
float PARAMETER_READ( string ICName, int readAddress, int intbits,
                      int fracbits );

Read float parameter value from a registe
float PARAMETER_READ_FLOAT( string ICName, int readAddress );

Read array of parameters
float[] PARAMETER_READ_ARRAY( string ICName, int readAddress, int intbits,
                              int fracbits, int numValsToRead );

Write parameter value via safeload
bool PARAMETER_SAFELAOD_WRITE( string ICName, int writeAddress, int intbits,
                               int fracbits, float valToWrite );

Write data to parameters via safeload
bool PARAMETER_SAFELAOD_ARRAY( string ICName, int writeAddress, int intbits,
                               int fracbits, int numValsToWrite,
                               float[] aValsToWrite );

Write parameter value via safeload as float
bool PARAMETER_SAFELOAD_WRITE_FLOAT( string ICName, int writeAddress,
                                     float valToWrite );

Write data to parameters via safeload as float
bool PARAMETER_SAFELOAD_ARRAY_FLOAT( string ICName, int writeAddress,
                                     int numValsToWrite, float[] aValsToWrite );

Run SigmaStudio script
bool RUN_SCRIPT( string script );

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

Undo the last operation performed on active SigmaStudio project
bool UNDO_SCRIPT();

Redo the last operation on active SigmaStudio project
bool REDO_SCRIPT();

Undo the last operation performed on SigmaStudio project
bool UNDO_SCRIPT_PROJECT( string projectName );
```

```
Redo the last operation on SigmaStudio project
bool REDO_SCRIPT_PROJECT( string projectName );

Auto dismiss all error,warning, and notifications and log messages to a file
bool SET_LOGGING_MODE( bool enabled );

Insert an object to Schematic
bool INSERT_OBJECT( string objectTypeName );

Insert an object to Schematic at specified location
bool INSERT_OBJECT_POINT( string objectTypeName, int pointX, int pointY );

Insert a block object to Schematic
bool INSERT_BLOCKOBJECT( string objectTypeName );

Insert a block object to Schematic at specified location
bool INSERT_BLOCKOBJECT_POINT( string objectTypeName, int pointX, int pointY );

Remove an object from a Schematic
bool REMOVE_OBJECT( string objectName );

Establish connection betweenn pins belonging to two objects
bool CONNECT_OBJECT( string fromObjectName, int fromOutPinIndex,
                     string toObjectName, int toInPinIndex );

Remove a connection betweenn two objects
bool DISCONNECT_OBJECT( string fromObjectName, int fromOutPinIndex,
                        string toObjectName, int toInPinIndex);

Manage the properties of the objects on the active praoject
bool SET_OBJECT_PROPERTY( string opcode, string objectName,
                          params object[] propertyParams);

Export the system files of the active project
bool EXPORT_SYSTEM_FILES( string fullyQualifiedFileName );

Generate SigmaStudio plugin using Algorithm Designer
bool BUILD_DESIGNER_DLL( string ICName, string fullyQualifiedProjectName,
                         string fullyQualifiedLibraryName );

Boot SHARC with a loader (*.ldr) file
bool BOOT_SHARC( string ICName, string fullyQualifiedFileName );

Read MIPS consumed by the Target SHARC hardware
float READ_MIPS_SHARC( string ICName );

Read version number of the software running on target SHARC processor
uint READ_VERSION_SHARC( string ICName );

Modify SigmaStudio settings
bool MODIFY_DESIGN_SETTINGS ( string cmd, params object[] arg );
```

```
Modify IC Control window properties
bool MODIFY_IC_CONTROL_PROPERTIES ( string ICName, string cmd, object arg );

Enable or Disable a Plug-In in the Add-Ins Window
bool MODIFY_PLUGIN_STATUS ( string plugin, bool state );
```

5.2 Accessing server APIs from Python

There are two ways the SigmaStudio server APIs can be accessed from Python.

5.2.1 Using it as a win32com Client

```
import win32com
import win32com.client
server =
win32com.client.dynamic.Dispatch("Analog.SigmaStudioServer.SigmaStudioServer")
projectFile="C:\Work\schematics\Example_schematic.dsppproj" #Just a dummy example
server.OPEN_PROJECT(projectFile)
```

5.2.2 Using ActivePython to access the server APIs

Perform the following steps to access SigmaStudio server scripts using ActivePython

5.2.2.1 Environment setup

1. Install ActivePython (version 2.7.2 for Python 2.7)
2. Install PythonNet (version 2.0 for Python 2.7)
3. Set PYTHONPATH=SIGMASTUDIO_INSTALLED_FOLDER
4. Launch python.exe from *'C:\Python27\pythonnet'.*

5.2.2.2 Create a SigmaStudioServer instance and run APIs via it

```
import clr
clr.AddReference('Analog.SigmaStudioServer')
from Analog.SigmaStudioServer import SigmaStudioServer
server=SigmaStudioServer()
server.OPEN_PROJECT(r'C: \Work\schematics\Example_schematic.dsppproj')
```

A. Command Line Execution

The SigmaStudio application has the capability to run a script from command line. A step-by-step procedure is given below.

1. Open a windows command prompt and change the directory to location of SigmaStudio application "*SStudio.exe*".
2. Ensure that there is no other active SigmaStudio application running in the PC.
3. Execute the command "*SStudio.exe \script-file Script-file-Name-With-Full-Path*", where "*Script-file-Name-With-Full-Path*" contains the complete path to the SigmaStudio script.

B. Matlab Connection

This example script designs an FIR filter with Matlab using following Matlab function:

```
h = firpm(N, a*2, b);
```

N: the filter order

Example of a length 31 lowpass filter:

```
H = firpm(30, [0 .1 .2 .5]*2, [1 1 0 0]);
```

Type help firpm in Matlab command line for more information on this function.

This scrip can take a Schematic as an input argument along with the FIR address and FirOrder. If the arguments are valid the script will run the filter on SHARC target through SigmaStudio. It is expected that the target is booted with an appropriate script.

B.1 Hardware and Software Requirement

All infrastructures required to run SigmaStudio.

Matlab installation with valid license.

B.2 Usage

```
matlabConnectionTest.exe schematic_name.dspproj fir Address firOrder
```

matlabConnectionTest.exe fir1.dspproj 10 30

B.3 FAQ

Q: How do I get the FIR address?

You may get it from the capture window or use export to get the address.

Q: How do I build the application?

Use the code given below to build the application

```
// #LANGUAGE# C#
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Analog.SigmaStudioServer;

namespace ConsoleApplication1
{
    class Program
    {
        class mYSsMatlab
        {
            public SigmaStudioServer sserver;
            public MLApp.MLApp matlab;
            public void Message(string s)
            {
                Console.WriteLine(s);
            }
            public bool runFir(string schematicName, int AddrFirCoef, float[]
coef, int N)
            {
                int delayms = 10000;
                bool ret = sserver.SET_LOGGING_MODE(true);
                try {
                    ret = sserver.SET_LOGGING_MODE(true);
                    if (ret != true) {
                       Message("Couldn't connect to SigmaStudio, please make
sure that the SigmaStudio is running"); return ret;
                    }
                }
                catch (Exception ex){
                    Console.WriteLine("Server Connection failed. exception {0}",
ex.Message);
                    Message("Couldn't connect to SigmaStudio, please make sure
that the SigmaStudio is running");
                    return ret;
                }
                sserver.CLOSE_PROJECT();
                ret = sserver.OPEN_PROJECT(schematicName);
                if (ret == false){
                    Message("schematic load failed " + schematicName);
                    return ret;
                }
                ret = sserver.DELAY_SCRIPT(delayms);
                sserver.DOWNLOAD();
                if (ret == false) {
                    Message("schematic load failed " + schematicName);
                    return ret;
                }
                sserver.PARAMETER_WRITE_ARRAY_FLOAT("IC 1", AddrFirCoef, N,
```

```
coef);
                sserver.DELAY_SCRIPT(delayms);
                Message("Test case for : " + "matlabTest");
                return ret;
            }

            public float[] designFir(int N)
            {
                float[] h = new float[N+1];
                System.Array h_r = new double[N+1];
                System.Array h_i = new double[N+1];
                string returnString; int i = 0;

                /* Example of a length 31 lowpass filter:
h=firpm(30,[0 .1 .2 .5]*2,[1 1 0 0]); */
                System.Array a = new double[4]; a.SetValue(0, 0);
a.SetValue(0.1, 1); a.SetValue(0.2, 2); a.SetValue(0.5, 3);
                System.Array b = new double[4]; b.SetValue(1, 0); b.SetValue(1,
1); b.SetValue(0, 2); b.SetValue(0, 3);
                System.Array z = new double[4]; z.SetValue(0, 0); z.SetValue(0,
1); z.SetValue(0, 2); z.SetValue(0, 3);

                matlab.PutFullMatrix("a", "base", a, z);
                matlab.PutFullMatrix("b", "base", b, z);

                StringBuilder matlabCmdFir = new StringBuilder();

                matlabCmdFir.AppendFormat("h = firpm({0}, a*2, b)", N);
                returnString = matlab.Execute(matlabCmdFir.ToString());
                /* returnString = matlab.Execute("h = firpm(30, a*2, b)"); */

                matlab.GetFullMatrix("h", "base", ref h_r, ref h_i);

                foreach(double c in h_r)h[i++] = (float)Convert.ToDecimal(c);
                return h;
            }
            public mYSsMatlab()
            {
                matlab = new MLApp.MLApp();
                sserver = new SigmaStudioServer();
            }
        }

        class Class1
        {
            [STAThread]
            static void Main(string[] args)
            {   int i = 0;
                int orderFir = 30;
                int addressFir = 30;
                if(args.Length != 3)
                {
                    Console.WriteLine("    ERROR in arguments. Usage testMatlab
```

```
schematic.dspproj firOrder");
                    return ;
                }
                try { addressFir = Int32.Parse(args[1]); }
                catch { Console.WriteLine("ERROR : wrong fir Address {0} Exiting
", args[1]);  return;  }
                try { orderFir = Int32.Parse(args[2]); }
                catch { Console.WriteLine("ERROR : wrong filter order {0}",
args[1]); orderFir = 30; }

                Console.WriteLine("Testing FIR coef design with matlab and
running with SigmaStudio, schematic used = {0} firAddress = {1} fir Order = {2}
please wait...", args[0], orderFir, addressFir);

                mYSsMatlab ssMatLabTest = new mYSsMatlab();

                /* design FIR using matlab */
                float[] h = ssMatLabTest.designFir(orderFir);

                /* print the fir coef */
                Console.Write("coef [{0}]= ", h.Length);
                foreach (double c in h) {
                    Console.Write(" {0}", c); i++;
                } Console.WriteLine("] ");

                /* run FIR on sharc through SigmaStudio */
                ssMatLabTest.runFir(args[0], addressFir, h, 30);
            }
        }


    }
}
```

Terminology

Table 5: Terminology

Term	Description
ADI	Analog Devices Inc.
API	Application Program Interface
OBJ	Object file format in CrossCore Embedded Studio environment
SPI	Serial Peripheral Interface
SSn	SHARC machine code corresponds to SigmaStudio Schematic
USB	Universal Serial Bus
XML	Extensible Markup Language

References

Table 6: References

Reference No.	Description
[1]	AE_42_SS4G_QuickStartGuide.docx
[2]	AE_42_SS4G_ReleaseNotes.docx