# Sim Reference Manual

Generated by Doxygen 1.3.8

Wed Mar 9 15:36:28 2005

# Contents

# Chapter 1

# The Sim(p. 40) Class Library

## 1.1 Introduction

This class library can be used to help generate S-Function binaries whilst using the C++ language. Although C++ is supported under the `mex` compiler (when used in conjunction with a C++ compiler such as Borland C++ or Microsoft visual C++) the interface is severly bogged down within its original C roots. This class library helps the user 'break free' from the ties of the oringinal S-Function C templates.

## 1.2 Usage

### 1.2.1 Inheritance and Definitions

The main class is an abstract class called **Sim**(p. 40). This class may be inherited from with the user's own simulation class. There are a number of pure virtual member functions which must be overloaded in the user's class to achieve compilation — these will be discussed later.

There are a few #defines that are required at the top of the users class file if the file is to be properly compiled into a usable S-Function. These #defines are:

- `SIM_CLASS_NAME` <*class_ name*>
- `SIM_FILE_NAME` <*file_ name* (minus extension)>
- `SIM_CLASS_DECLARATION` (to be inserted directly before the class *declaration*)
- `SIM_CLASS_DEFINITION` (to be inserted directly before the class *definition*)

The position of the `SIM_CLASS_DECLARATION` and `SIM_CLASS_DEFINITION` defines are very important and it is suggested that they are placed at the top of the cpp file before any other includes. Because of the complicated compilation and linking order that takes place when compiling an S-Function, **sim.h**(p. 69) has to be included *twice* — Once before the class *declaration*, and once before the class *definition(s)*. It is suggested that a header file is used to abstract the declaration. If this is done then order of instructions will be as shown in the following example:

An example of a header file follows:

`example.h`:

```
#ifndef __EXAMPLE_H__
#define __EXAMPLE_H__

#include <sim.h>


class ExampleClass : public Sim
{
    private:
            // Interface Protocal
            ContinuousInterface * m_if;

    public:
            // Constructor
            ExampleClass(SimStruct *s);
            // Destructor
            virtual ~ExampleClass();

            // Declare Sim Functions
            virtual void start();
            virtual void init_cond();
            virtual void update();
            virtual void derivative();
            virtual void output();
            virtual void terminate();
};

#endif // __EXAMPLE_H__
```

An example of the code that could define the class in `example.h` follows:

`example.cpp`

```
//------------------------------------------------
//            Compulsary Defines
//------------------------------------------------

#define SIM_CLASS_NAME      ExampleClass
#define SIM_FILE_NAME       example

//------------------------------------------------
//            Declare the Sim class
//------------------------------------------------

#define SIM_CLASS_DECLARATION
#include "example.h"

//------------------------------------------------
//            Define the Sim class
//------------------------------------------------

#define SIM_CLASS_DEFINITION
#include <sim.h>

// Constructor
ExampleClass::ExampleClass(SimStruct *p_S) : Sim(p_S)
{
    // Register a new Interface with:
            // 1 input, 3 outputs and 1 state
    m_if = addContinuousInterface("My Interface", 1, 3, 1, 0);
}

// Destructor
ExampleClass::~ExampleClass()
{
}
```

```
void ExampleClass::start()
{
}

void ExampleClass::init_cond()
{
    // Set initial contition of state
    m_if->x(0) = 0;
}

void ExampleClass::update()
{
}

void ExampleClass::derivative()
{
    // Put input into dx for integration
    m_if->dx(0) = m_if->u(0);
}

void ExampleClass::output()
{
    // First output a direct copy of input
    m_if->y(0) = m_if->u(0);

    // Second output a multiple of input
    m_if->y(1) = 3 * m_if->u(0);

    // Third output the integral of the input
    m_if->y(2) = m_if->x(0);
}

void ExampleClass::terminate()
{
}
```

Note the use of `SIM_CLASS_DECLARATION` and `SIM_CLASS_DEFINITION` in the above files. After each `#define` is written the **sim.h**(p. 69) file is included. This file must be included after the `#define` commands as it is important for the **sim.h**(p. 69) file to know the circumstance under which it is being included. Note also that the `SIM_CLASS_DECLARATION` *must* be included *before* the `SIM_CLASS_DEFINITION` (although, this is fairly obvious if you think about it!).

### 1.2.2 Inputs and Outputs

The ExampleClass above uses 1 input, 3 outputs and 1 state. It obtains these IOs by 'requesting' a new 'ContinuousInterface' using the command **Sim.addContinuousInterface()**(p. 42):

```
m_ip = addContinuousInterface("My Interface", 1, 3, 1, 0); // 1 input, 3 outputs and 1 state
```

The member variable m_if is a pointer to a **ContinuousInterface**(p. 17) class. The **Sim.addContinuousInterface()**(p. 42) function returns a pointer to the newly created interface, and can then be used to access the IOs requested. The **ContinuousInterface**(p. 17) class has 5 main member functions to access the IOs:

- ContinuousInterface.u(unsigned int index) - To access the inputs
- ContinuousInterface.y(unsigned int index) - To access the outputs
- ContinuousInterface.x(unsigned int index) - To access the states
- ContinuousInterface.dx(unsigned int index) - To access the state derivatives

- ContinuousInterface.param(unsigned int index) - To access the S-Function parameters

In the above example, the three outputs are used to output:

1. The first output is a direct copy of the input signal.
2. The second output is an amplification (by a factor of 3) of the input signal.
3. The third output is that of the single state we are using. The state derivative of which is given by the input in ExampleClass::derivative(). The third output is therefore the integral of the input.

### 1.2.3 Compulsary Member Functions

There are a number of member functions that are declared as pure virtual in class **Sim**(p. 40). These functions will have to defined in the users own simulation class. These functions are:

- start()
- init_cond()
- update()
- derivative()
- output()
- terminate()

These functions are designed to mimic the behaviour of thier original S-Function counterparts:

- mdlStart()
- mdlInitializeConditions()
- mdlUpdate()
- mdlDerivatives()
- mdlOutputs()
- mdlTerminate()

It is advised that the user reads the documentation in the Matlab literature to obtain an understanding of when these functions are called, and what operations to perform with each function.

## 1.3 Compilation

The code for all the relevant **Sim**(p. 40) classes are compiled into a library file called `sim.lib`. To compile your code it will need to be linked with `sim.lib`. For example, the `example.cpp` file listed above could be compiled using the following command on the matlab command prompt:

```
>> mex example.cpp sim.lib -v
```

If compilation is successfull an `example.dll` file will be created for use within Simulink.

## 1.4 Installation

### 1.4.1 Installing on MATLAB 5 with Borland 5.x compiler

To install the SimClassLib, download the SimClassLib folder to a suitable directory. Then navigate to \Documents and Settings\<user name>\Application Data\Mathworks\MATLAB\ and open mexopts.bat in a text editor.

Add the line

```
set SIMCLASSLIB=<path to SimClassLib\code\>
```

to the file just under the definition for the MATLAB path. For example:

```
rem ************************************************************************
rem General parameters
rem ************************************************************************
set MATLAB=%MATLAB%
set BORLAND=c:\Program Files\Borland\BCC55\
set SIMCLASSLIB=c:\ltn100\Shared\SimClassLib\
...
```

On the line beginning set INCLUDE=..., add the following to the end:

```
;%SIMCLASSLIB%\code\include
```

Finally, a bit further down add the following to the end of the line starting set LINKFLAGS=...

```
-L"%SIMCLASSLIB%\code\bin"
```

The complete file should look something like this:

```
@echo off
rem BCC53OPTS.BAT
rem
rem    Compile and link options used for building MEX-files
rem    with the Borland C compiler
rem
rem    $Revision: 1.2 $  $Date: 1998/12/30 18:59:07 $
rem
rem ************************************************************************
rem General parameters
rem ************************************************************************
set MATLAB=%MATLAB%
set BORLAND=c:\Program Files\Borland\BCC55\
set SIMCLASSLIB=c:\ltn100\Shared\SimClassLib\
set PATH=%BORLAND%\BIN;%MATLAB_BIN%;%PATH%
set INCLUDE=%BORLAND%\INCLUDE;%SIMCLASSLIB%\code\include
set LIB=%BORLAND%\LIB;%BORLAND%\LIB\32BIT

rem ************************************************************************
rem Compiler parameters
rem ************************************************************************
set COMPILER=bcc32
set COMPFLAGS=-c -3 -P- -w- -pc -a8 -I"%INCLUDE%" -DMATLAB_MEX_FILE
set OPTIMFLAGS=-O2
set DEBUGFLAGS=-v
set NAME_OBJECT=-o
```

```
rem ********************************************************************
rem Library creation command
rem ********************************************************************
set PRELINK_CMDS1=copy "%MATLAB%\extern\include\_mex.def" "%OUTDIR%%MEX_NAME%.def"
set PRELINK_CMDS2=implib -i %LIB_NAME%1.lib "%MATLAB%\extern\include\_matlab.def"
set PRELINK_CMDS3=implib -i %LIB_NAME%2.lib "%MATLAB%\extern\include\_libmatlbmx.def"
set PRELINK_DLLS=implib -i %DLL_NAME%.lib "%MATLAB%\extern\include\_%DLL_NAME%.def"

rem ********************************************************************
rem Linker parameters
rem ********************************************************************
set LINKER=perl %MATLAB_BIN%\link_borland_mex.pl
set LINKFLAGS=-aa -c -Tpd -x -Gn -L\"%BORLAND%"\lib\32bit .... "%OUTDIR%%MEX_NAME%.def" -L"%SIMCLASSLIB%\code\bin"
set LINKOPTIMFLAGS=
set LINKDEBUGFLAGS=-v
set LINK_FILE=
set LINK_LIB=
set NAME_OUTPUT="%OUTDIR%%MEX_NAME%".dll
set RSP_FILE_INDICATOR=@

rem ********************************************************************
rem Resource compiler parameters
rem ********************************************************************
set RC_COMPILER=brcc32 -w32 -D_NO_VCL -fomexversion.res
set RC_LINKER=
```

# Chapter 2

# Sim Hierarchical Index

## 2.1 Sim Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

# Chapter 3

# Sim Class Index

## 3.1 Sim Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 4

# Sim File Index

## 4.1 Sim File List

Here is a list of all files with brief descriptions:

# Chapter 5

# Sim Page Index

## 5.1 Sim Related Pages

Here is a list of all related documentation pages:

The MatrixClass Library `[external]`

# Chapter 6

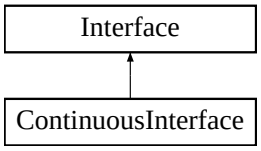
# Sim Class Documentation

## 6.1 ContinuousInterface Class Reference

ContinuousInterface class.

`#include <sim_interface.h>`

Inheritance diagram for ContinuousInterface::




### Public Member Functions

- **ContinuousInterface** (**InterfaceContainer** *p_interfaceContainer, const char *p_name, const unsigned int p_inputs, const unsigned int p_outputs, const unsigned int p_states, const unsigned int p_params)

  *Constructor for ContinuousInterface class.*

- virtual unsigned int **getNumStates** () const

  *Return the number of states for this interface.*

- virtual void **invalidateX** ()

  *Invalidate the state pointer for this* **Interface**(p. 26).

- void **invalidateDx** ()

  *Invalidate the state derivative pointer for this* **Interface**(p. 26).

### Public Attributes

- **SimStateDerivativeVector dx**

  *State derivative vector.*

- **SimContinuousStateVector x**

  *State vector.*

### 6.1.1 Detailed Description

ContinuousInterface class.

**Author:**
Peter Mendham

The ContiuousInterface class extends the **Interface**(p. 26) class to define an interface to Simulink with a with state derivatives, which are accessible via **dx( )**(p. 19).

A ContiuousInterface is assumed to require continuous sample time.

A new interface protocol can be 'requested' via InterfaceContainer.newInterface( ).

**See also:**
**Interface**(p. 26), **Sim**(p. 40), **InterfaceContainer**(p. 30)

### 6.1.2 Constructor & Destructor Documentation

#### 6.1.2.1 ContinuousInterface::ContinuousInterface (InterfaceContainer * *p_interfaceContainer*, const char * *p_name*, const unsigned int *p_inputs*, const unsigned int *p_outputs*, const unsigned int *p_states*, const unsigned int *p_params*)

Constructor for ContinuousInterface class.

**Parameters:**
***p_interfaceContainer*** Pointer to the **InterfaceContainer**(p. 30) class.
***p_name*** Name of interface
***p_inputs*** Number of inputs required.
***p_outputs*** Number of outputs required.
***p_states*** Number of states required.
***p_params*** Number of params required.

### 6.1.3 Member Function Documentation

#### 6.1.3.1 virtual unsigned int ContinuousInterface::getNumStates () const [inline, virtual]

Return the number of states for this interface.

Implements **Interface** (p. 28).

#### 6.1.3.2 void ContinuousInterface::invalidateDx () [inline]

Invalidate the state derivative pointer for this **Interface**(p. 26).

#### 6.1.3.3 virtual void ContinuousInterface::invalidateX () [inline, virtual]

Invalidate the state pointer for this **Interface**(p. 26).

Implements **Interface** (p. 28).

### 6.1.4 Member Data Documentation

#### 6.1.4.1 SimStateDerivativeVector ContinuousInterface::dx

State derivative vector.

#### 6.1.4.2 SimContinuousStateVector ContinuousInterface::x

State vector.

The documentation for this class was generated from the following files:

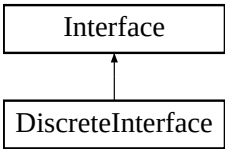
- code/include/**sim_interface.h**
- code/src/**sim_interface.cpp**

## 6.2 DiscreteInterface Class Reference

DiscreteInterface class.

`#include <sim_interface.h>`

Inheritance diagram for DiscreteInterface::




### Public Member Functions

- **DiscreteInterface** (**InterfaceContainer** *p_interfaceContainer, const char *p_name, const unsigned int p_inputs, const unsigned int p_outputs, const unsigned int p_states, const unsigned int p_params, const double p_inputSampleTime, const double p_outputSampleTime, const double p_inputSampleOffset, const double p_outputSampleOffset)

  *Constructor for DiscreteInterface class.*

- **DiscreteInterface** (**InterfaceContainer** *p_interfaceContainer, const char *p_name, const unsigned int p_inputs, const unsigned int p_outputs, const unsigned int p_states, const unsigned int p_params, const double p_sampleTime)

  *Constructor for DiscreteInterface class.*

- virtual unsigned int **getNumStates** () const

  *Return the number of states for this interface.*

- virtual void **invalidateX** ()

  *Invalidate the state pointer for this* **Interface**(p. 26).

- double **getInputSampleTime** () const

  *Get the input port sample time in seconds.*

- double **getInputSampleOffset** () const

  *Get the input port sample time offset in seconds.*

- double **getOutputSampleTime** () const

  *Get the output port sample time in seconds.*

- double **getOutputSampleOffset** () const

  *Get the output port sample time offset in seconds.*

### Public Attributes

- **SimDiscreteStateVector x**

  *State vector.*

## Protected Attributes

- double **m_inputSampleTime**
- double **m_inputSampleOffset**
- double **m_outputSampleTime**
- double **m_outputSampleOffset**

### 6.2.1 Detailed Description

DiscreteInterface class.

**Author:**
Peter Mendham

The DiscreteInterface class extends the **Interface**(p. 26) class to define an interface to Simulink with a with next state values, which are accessible via nx( ).

A ContiuousInterface is assumed to require continuous sample time.

A new interface protocol can be 'requested' via InterfaceContainer.newInterface( ).

**See also:**
**Interface**(p. 26), **Sim**(p. 40), **InterfaceContainer**(p. 30)

### 6.2.2 Constructor & Destructor Documentation

#### 6.2.2.1 DiscreteInterface::DiscreteInterface (InterfaceContainer * *p_interfaceContainer*, const char * *p_name*, const unsigned int *p_inputs*, const unsigned int *p_outputs*, const unsigned int *p_states*, const unsigned int *p_params*, const double *p_inputSampleTime*, const double *p_outputSampleTime*, const double *p_inputSampleOffset*, const double *p_outputSampleOffset*)

Constructor for DiscreteInterface class.

**Parameters:**
***p_interfaceContainer*** Pointer to the **InterfaceContainer**(p. 30) class.

***p_name*** Name of interface

***p_inputs*** Number of inputs required.

***p_outputs*** Number of outputs required.

***p_states*** Number of states required.

***p_params*** Number of params required.

***p_inputSampleTime*** Input sample time in seconds

***p_outputSampleTime*** Output sample time in seconds

***p_inputSampleOffset*** Input sample time offset in seconds

***p_outputSampleOffset*** Output sample time offset in seconds

#### 6.2.2.2 DiscreteInterface::DiscreteInterface (InterfaceContainer * *p_interfaceContainer*, const char * *p_name*, const unsigned int *p_inputs*, const unsigned int *p_outputs*, const unsigned int *p_states*, const unsigned int *p_params*, const double *p_sampleTime*)

Constructor for DiscreteInterface class.

**Parameters:**

***p_interfaceContainer*** Pointer to the **InterfaceContainer**(p. 30) class.

***p_name*** Name of interface

***p_inputs*** Number of inputs required.

***p_outputs*** Number of outputs required.

***p_states*** Number of states required.

***p_params*** Number of params required.

***p_sampleTime*** Input and output sample time in seconds

### 6.2.3 Member Function Documentation

#### 6.2.3.1 double DiscreteInterface::getInputSampleOffset () const [inline]

Get the input port sample time offset in seconds.

#### 6.2.3.2 double DiscreteInterface::getInputSampleTime () const [inline]

Get the input port sample time in seconds.

#### 6.2.3.3 virtual unsigned int DiscreteInterface::getNumStates () const [inline, virtual]

Return the number of states for this interface.

Implements **Interface** (p. 28).

#### 6.2.3.4 double DiscreteInterface::getOutputSampleOffset () const [inline]

Get the output port sample time offset in seconds.

#### 6.2.3.5 double DiscreteInterface::getOutputSampleTime () const [inline]

Get the output port sample time in seconds.

#### 6.2.3.6 virtual void DiscreteInterface::invalidateX () [inline, virtual]

Invalidate the state pointer for this **Interface**(p. 26).

Implements **Interface** (p. 28).

### 6.2.4 Member Data Documentation

#### 6.2.4.1 double DiscreteInterface::m_inputSampleOffset [protected]

#### 6.2.4.2 double DiscreteInterface::m_inputSampleTime [protected]

#### 6.2.4.3 double DiscreteInterface::m_outputSampleOffset [protected]

#### 6.2.4.4 double DiscreteInterface::m_outputSampleTime [protected]

#### 6.2.4.5 SimDiscreteStateVector DiscreteInterface::x

State vector.

The documentation for this class was generated from the following files:

- code/include/**sim_interface.h**
- code/src/**sim_interface.cpp**

## 6.3 Exception Class Reference

Exception class.

`#include <sim_exception.h>`

### Public Member Functions

- **Exception** ()

  *Exception class constructor.*

- **Exception** (const char *p_string)

  *Class constructor with string input.*

- **kill** (SimStruct *p_S)

### Protected Attributes

- string **m_exception_name**

  *Exception description (defaulted to "General Exception").*

### 6.3.1 Detailed Description

Exception class.

**Author:**
Lee Netherton

The Exception class is used to throw exceptions which will halt the simulation. For example:

```
// Check for range error
if(value<min_range || value>max_range)
                throw Exception();
```

By default Exception class is equiped with a `string` member variable equal to "General Exception". This `string` will be printed when the simulaton halts. To provive a more discriptive exception, the exeption class can be inherited from and then thrown:

```
class RangeException : Exception
{
        // Constructor
        RangeException();
};
RangeException::RangeException() : m_exception_name("General Exception") {}

...

// Check for range error
if(value<min_range || value>max_range)
        throw RangeException();
```

### 6.3.2 Constructor & Destructor Documentation

#### 6.3.2.1 Exception::Exception ()

Exception class constructor.

#### 6.3.2.2 Exception::Exception (const char * *p_string*)

Class constructor with string input.

### 6.3.3 Member Function Documentation

#### 6.3.3.1 Exception::kill (SimStruct * *p_S*)

Kill the simulation and print the exception description (m_exception_name) to screen.

**Parameters:**
***p_S*** Pointer to SimStruct.

### 6.3.4 Member Data Documentation

#### 6.3.4.1 string Exception::m_exception_name [protected]

Exception description (defaulted to "General Exception").

The documentation for this class was generated from the following files:

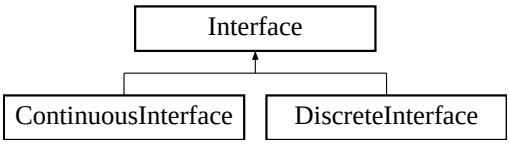
- code/include/**sim_exception.h**
- code/src/**sim_exception.cpp**

## 6.4 Interface Class Reference

Interface class.

`#include <sim_interface.h>`

Inheritance diagram for Interface::




### Public Member Functions

- **Interface** (**InterfaceContainer** *p_interfaceContainer, const char *p_name, const unsigned int p_inputs, const unsigned int p_outputs, const unsigned int p_params)

  *Constructor for Interface class.*

- unsigned int **getNumInputs** () const

  *Return the number of inputs for this interface.*

- unsigned int **getNumOutputs** () const

  *Return the number of outputs for this interface.*

- virtual unsigned int **getNumStates** () const =0

  *Return the number of states for this interface.*

- unsigned int **getNumParams** () const

  *Return the number of params for this interface.*

- **InterfaceContainer** * **getInterfaceContainer** ()

  *Returns the* **InterfaceContainer**(p. 30) *pointer.*

- string **getName** () const

  *Returns the interface name.*

- virtual void **printInfo** () const

  *Prints information about the interface.*

- void **invalidateU** ()

  *Invalidate the input pointer for this Interface.*

- void **invalidateY** ()

  *Invalidate the output pointer for this Interface.*

- virtual void **invalidateX** ()=0

  *Invalidate the state pointer for this Interface.*

- void **invalidateParam** ()

  *Invalidate the parameter pointer for this Interface.*

## Public Attributes

- **SimInputVector u**

  *Input vector.*

- **SimOutputVector y**

  *Output vector.*

- **SimParamVector param**

  *Parameters vector.*

### 6.4.1 Detailed Description

Interface class.

**Author:**
Lee Netherton

The Interface class defines an generic interface to Simulink with a number of:

- Inputs. Accessible via **u( )**(p. 29)
- Outputs. Accessible via **y( )**(p. 29)
- States. Accessible via x( )
- Parameters. Accessible via **param( )**(p. 29)

The Interface class is never created, one of its child classes **ContinuousInterface**(p. 17) or **DiscreteInterface**(p. 20) is always used

**See also:**
**Sim**(p. 40), **InterfaceContainer**(p. 30), **ContinuousInterface**(p. 17), **DiscreteInterface**(p. 20)

### 6.4.2 Constructor & Destructor Documentation

#### 6.4.2.1 Interface::Interface (InterfaceContainer * *p_interfaceContainer*, const char * *p_name*, const unsigned int *p_inputs*, const unsigned int *p_outputs*, const unsigned int *p_params*)

Constructor for Interface class.

**Parameters:**
***p_interfaceContainer*** Pointer to the **InterfaceContainer**(p. 30) class.

***p_name*** Name of interface

***p_inputs*** Number of inputs required.

***p_outputs*** Number of outputs required.

***p_params*** Number of params required.

### 6.4.3 Member Function Documentation

#### 6.4.3.1 InterfaceContainer* Interface::getInterfaceContainer () [inline]

Returns the **InterfaceContainer**(p. 30) pointer.

#### 6.4.3.2 string Interface::getName () const [inline]

Returns the interface name.

#### 6.4.3.3 unsigned int Interface::getNumInputs () const [inline]

Return the number of inputs for this interface.

#### 6.4.3.4 unsigned int Interface::getNumOutputs () const [inline]

Return the number of outputs for this interface.

#### 6.4.3.5 unsigned int Interface::getNumParams () const [inline]

Return the number of params for this interface.

#### 6.4.3.6 virtual unsigned int Interface::getNumStates () const [pure virtual]

Return the number of states for this interface.
Implemented in **ContinuousInterface** (p. 18), and **DiscreteInterface** (p. 22).

#### 6.4.3.7 void Interface::invalidateParam () [inline]

Invalidate the parameter pointer for this Interface.

#### 6.4.3.8 void Interface::invalidateU () [inline]

Invalidate the input pointer for this Interface.

#### 6.4.3.9 virtual void Interface::invalidateX () [pure virtual]

Invalidate the state pointer for this Interface.
Implemented in **ContinuousInterface** (p. 19), and **DiscreteInterface** (p. 22).

#### 6.4.3.10 void Interface::invalidateY () [inline]

Invalidate the output pointer for this Interface.

#### 6.4.3.11 void Interface::printInfo () const [virtual]

Prints information about the interface.

### 6.4.4 Member Data Documentation

#### 6.4.4.1 SimParamVector Interface::param

Parameters vector.

#### 6.4.4.2 SimInputVector Interface::u

Input vector.

#### 6.4.4.3 SimOutputVector Interface::y

Output vector.

The documentation for this class was generated from the following files:

- code/include/**sim_interface.h**
- code/src/**sim_interface.cpp**

## 6.5 InterfaceContainer Class Reference

Simulation interface container class.

`#include <sim_interface_container.h>`

### Public Member Functions

- **InterfaceContainer** (**Sim** *p_sim)

  *InterfaceContainer constructor.*

- virtual ~**InterfaceContainer** ()
- **Sim** * **getSim** () const

  *Return a pointer to the* **Sim**(p. 40) *class.*

- **ContinuousInterface** * **addContinuousInterface** (const char *p_name, const unsigned int p_inputs, const unsigned int p_outputs, const unsigned int p_states, const unsigned int p_params)

  *Add a new* **ContinuousInterface**(p. 17) *object, and return a pointer to it.*

- **DiscreteInterface** * **addDiscreteInterface** (const char *p_name, const unsigned int p_inputs, const unsigned int p_outputs, const unsigned int p_states, const unsigned int p_params, const double p_inputSampleTime, const double p_outputSampleTime, const double p_inputSampleOffset, const double p_outputSampleOffset)

  *Add a new* **DiscreteInterface**(p. 20) *object, and return a pointer to it.*

- **DiscreteInterface** * **addDiscreteInterface** (const char *p_name, const unsigned int p_inputs, const unsigned int p_outputs, const unsigned int p_states, const unsigned int p_params, const double p_sampleTime)

  *Add a new* **DiscreteInterface**(p. 20) *object, and return a pointer to it.*

- unsigned int **getNumInputs** ()

  *Return the total number of inputs.*

- unsigned int **getNumInputs** (const unsigned int interfaceIndex)

  *Return the number of input ports for one interface.*

- unsigned int **getNumActiveInputPorts** ()

  *Return the total number of input ports with a width > 0.*

- unsigned int **getActiveInputPortWidth** (const unsigned int portIndex)

  *Return the width of the ith active input port.*

- unsigned int **getNumOutputs** ()

  *Return the total number of output ports.*

- unsigned int **getNumOutputs** (const unsigned int interfaceIndex)

  *Return the number of output ports for one interface.*

- unsigned int **getNumActiveOutputPorts** ()

*Return the total number of output ports with a width > 0.*

- unsigned int **getActiveOutputPortWidth** (const unsigned int portIndex)
  *Return the width of the ith active output port.*
- unsigned int **getNumContinuousStates** ()
  *Return the total number of continuous states.*
- unsigned int **getNumContinuousStates** (const unsigned int interfaceIndex)
  *Return the number of continuous states for one interface.*
- unsigned int **getNumDiscreteStates** ()
  *Return the total number of discrete states.*
- unsigned int **getNumDiscreteStates** (const unsigned int interfaceIndex)
  *Return the number of discrete states for one interface.*
- unsigned int **getNumParams** ()
  *Return the total number of parameters.*
- unsigned int **getNumParams** (const unsigned int interfaceIndex)
  *Return the number of parameters for one interface.*
- unsigned int **getNumActiveParams** ()
  *Return the total number of parameters with a width > 0.*
- unsigned int **getActiveParamWidth** (const unsigned int portIndex)
  *Return the width of the ith parameter port.*
- unsigned int **getNumContinuousInterfaces** ()
  *Return the number of continuous interfaces.*
- unsigned int **getNumDiscreteInterfaces** ()
  *Return the number of discrete interfaces.*
- unsigned int **getNumInterfaces** ()
  *Return the total number of interfaces.*
- double **getActiveInputPortSampleTime** (const unsigned int portIndex)
  *Return the sample time for a given active input port.*
- double **getActiveInputPortSampleOffset** (const unsigned int portIndex)
  *Return the sample time offset for a given active input port.*
- double **getActiveOutputPortSampleTime** (const unsigned int portIndex)
  *Return the sample time for a given active output port.*
- double **getActiveOutputPortSampleOffset** (const unsigned int portIndex)
  *Return the sample time offset for a given active output port.*

- void **printInfo** ()

  *Prints an* **Interface**(p. 26) *map for all interfaces.*

- const real_T * **getU** (**Interface** *interface)
- real_T * **getY** (**Interface** *interface)
- real_T * **getContinuousX** (**ContinuousInterface** *interface)
- real_T * **getDiscreteX** (**DiscreteInterface** *interface)
- real_T * **getDx** (**ContinuousInterface** *interface)
- const real_T * **getParam** (**Interface** *interface)
- void **invalidateU** ()

  *Invalidate the input pointer for all Interfaces.*

- void **invalidateY** ()

  *Invalidate the output pointer for all Interfaces.*

- void **invalidateX** ()

  *Invalidate the state pointer for all Interfaces.*

- void **invalidateDx** ()

  *Invalidate the state derivative pointer for all ContinuousInterfaces.*

- void **invalidateParam** ()

  *Invalidate the parameter pointer for all Interfaces.*

- void **invalidateAll** ()

  *Invalidate all pointers for all Interfaces.*

## Static Public Attributes

- const double **SAMPLE_TIME_CONTINUOUS** = -1.0
- const double **SAMPLE_OFFSET_CONTINUOUS** = -1.0

### 6.5.1 Detailed Description

Simulation interface container class.

**Author:**
Lee Netherton

The InterfaceContainer class acts as a container for all the inputs, outputs, states and parameters for the S-Function. An **Interface**(p. 26) can be 'requested' via the **addContinuousInterface()**(p. 33) and **addDiscreteInterface()**(p. 34) member fuctions. Each **Interface**(p. 26) has its own unique set of IOs which can be accessed as public members of the **Interface**(p. 26) classes.

## 6.5.2 Constructor & Destructor Documentation

### 6.5.2.1 InterfaceContainer::InterfaceContainer (Sim * p_sim)

InterfaceContainer constructor.

**Parameters:**

***p_sim*** Pointer to **Sim**(p. 40)

### 6.5.2.2 InterfaceContainer::~InterfaceContainer () [virtual]

## 6.5.3 Member Function Documentation

### 6.5.3.1 ContinuousInterface * InterfaceContainer::addContinuousInterface (const char * p_name, const unsigned int p_inputs, const unsigned int p_outputs, const unsigned int p_states, const unsigned int p_params)

Add a new **ContinuousInterface**(p. 17) object, and return a pointer to it.

**Parameters:**

***p_name*** Name of the interface.

***p_inputs*** Number of inputs required.

***p_outputs*** Number of outputs required.

***p_states*** Number of states required.

***p_params*** Number of params required.

**Returns:**

Pointer to **ContinuousInterface**(p. 17).

**See also:**

**ContinuousInterface**(p. 17)

### 6.5.3.2 DiscreteInterface * InterfaceContainer::addDiscreteInterface (const char * p_name, const unsigned int p_inputs, const unsigned int p_outputs, const unsigned int p_states, const unsigned int p_params, const double p_sampleTime)

Add a new **DiscreteInterface**(p. 20) object, and return a pointer to it.

**Parameters:**

***p_name*** Name of interface

***p_inputs*** Number of inputs required.

***p_outputs*** Number of outputs required.

***p_states*** Number of states required.

***p_params*** Number of params required.

***p_sampleTime*** Input and output sample time in seconds

**Returns:**
Pointer to **DiscreteInterface**(p. 20).

**See also:**
**DiscreteInterface**(p. 20)

#### 6.5.3.3 DiscreteInterface * InterfaceContainer::addDiscreteInterface (const char * *p_name*, const unsigned int *p_inputs*, const unsigned int *p_outputs*, const unsigned int *p_states*, const unsigned int *p_params*, const double *p_inputSampleTime*, const double *p_outputSampleTime*, const double *p_inputSampleOffset*, const double *p_outputSampleOffset*)

Add a new **DiscreteInterface**(p. 20) object, and return a pointer to it.

**Parameters:**
***p_name*** Name of interface

***p_inputs*** Number of inputs required.

***p_outputs*** Number of outputs required.

***p_states*** Number of states required.

***p_params*** Number of params required.

***p_inputSampleTime*** Input sample time in seconds

***p_outputSampleTime*** Output sample time in seconds

***p_inputSampleOffset*** Input sample time offset in seconds

***p_outputSampleOffset*** Output sample time offset in seconds

**Returns:**
Pointer to **DiscreteInterface**(p. 20).

**See also:**
**DiscreteInterface**(p. 20)

#### 6.5.3.4 double InterfaceContainer::getActiveInputPortSampleOffset (const unsigned int *portIndex*)

Return the sample time offset for a given active input port.

#### 6.5.3.5 double InterfaceContainer::getActiveInputPortSampleTime (const unsigned int *portIndex*)

Return the sample time for a given active input port.

#### 6.5.3.6 unsigned int InterfaceContainer::getActiveInputPortWidth (const unsigned int *portIndex*)

Return the width of the ith active input port.

#### 6.5.3.7 double InterfaceContainer::getActiveOutputPortSampleOffset (const unsigned int *portIndex*)

Return the sample time offset for a given active output port.

#### 6.5.3.8 double InterfaceContainer::getActiveOutputPortSampleTime (const unsigned int *portIndex*)

Return the sample time for a given active output port.

#### 6.5.3.9 unsigned int InterfaceContainer::getActiveOutputPortWidth (const unsigned int *portIndex*)

Return the width of the ith active output port.

#### 6.5.3.10 unsigned int InterfaceContainer::getActiveParamWidth (const unsigned int *portIndex*)

Return the width of the ith parameter port.

#### 6.5.3.11 real_T * InterfaceContainer::getContinuousX (ContinuousInterface * *interface*)

Get a pointer to the state data for a particular **Interface**(p. 26).

**Parameters:**
*interface* Pointer to the **Interface**(p. 26)

**Returns:**
A pointer to the data array

#### 6.5.3.12 real_T * InterfaceContainer::getDiscreteX (DiscreteInterface * *interface*)

Get a pointer to the state data for a particular **Interface**(p. 26).

**Parameters:**
*interface* Pointer to the **Interface**(p. 26)

**Returns:**
A pointer to the data array

#### 6.5.3.13 real_T * InterfaceContainer::getDx (ContinuousInterface * *interface*)

Get a pointer to the state derivative data for a particular **ContinuousInterface**(p. 17).

**Parameters:**
*interface* Pointer to the **Interface**(p. 26)

**Returns:**
A pointer to the data array

#### 6.5.3.14 unsigned int InterfaceContainer::getNumActiveInputPorts ()

Return the total number of input ports with a width > 0.

#### 6.5.3.15 unsigned int InterfaceContainer::getNumActiveOutputPorts ()

Return the total number of output ports with a width > 0.

#### 6.5.3.16 unsigned int InterfaceContainer::getNumActiveParams ()

Return the total number of parameters with a width > 0.

#### 6.5.3.17 unsigned int InterfaceContainer::getNumContinuousInterfaces () [inline]

Return the number of continuous interfaces.

#### 6.5.3.18 unsigned int InterfaceContainer::getNumContinuousStates (const unsigned int *interfaceIndex*)

Return the number of continuous states for one interface.

#### 6.5.3.19 unsigned int InterfaceContainer::getNumContinuousStates ()

Return the total number of continuous states.

#### 6.5.3.20 unsigned int InterfaceContainer::getNumDiscreteInterfaces () [inline]

Return the number of discrete interfaces.

#### 6.5.3.21 unsigned int InterfaceContainer::getNumDiscreteStates (const unsigned int *interfaceIndex*)

Return the number of discrete states for one interface.

#### 6.5.3.22 unsigned int InterfaceContainer::getNumDiscreteStates ()

Return the total number of discrete states.

#### 6.5.3.23 unsigned int InterfaceContainer::getNumInputs (const unsigned int *interfaceIndex*)

Return the number of input ports for one interface.

#### 6.5.3.24 unsigned int InterfaceContainer::getNumInputs ()

Return the total number of inputs.

#### 6.5.3.25 unsigned int InterfaceContainer::getNumInterfaces () [inline]

Return the total number of interfaces.

#### 6.5.3.26 unsigned int InterfaceContainer::getNumOutputs (const unsigned int *interfaceIndex*)

Return the number of output ports for one interface.

#### 6.5.3.27 unsigned int InterfaceContainer::getNumOutputs ()

Return the total number of output ports.

#### 6.5.3.28 unsigned int InterfaceContainer::getNumParams (const unsigned int *interfaceIndex*)

Return the number of parameters for one interface.

#### 6.5.3.29 unsigned int InterfaceContainer::getNumParams ()

Return the total number of parameters.

#### 6.5.3.30 const real_T * InterfaceContainer::getParam (Interface * *interface*)

Get a pointer to the simulation parameters data for a particular **Interface**(p. 26).

**Parameters:**
*interface* Pointer to the **Interface**(p. 26)

**Returns:**
A pointer to the data array

#### 6.5.3.31 Sim* InterfaceContainer::getSim () const [inline]

Return a pointer to the **Sim**(p. 40) class.

#### 6.5.3.32 const real_T * InterfaceContainer::getU (Interface * *interface*)

Get a pointer to the input data for a particular **Interface**(p. 26).

**Parameters:**
*interface* Pointer to the **Interface**(p. 26)

**Returns:**
A pointer to the data array

#### 6.5.3.33 real_T * InterfaceContainer::getY (Interface * *interface*)

Get a pointer to the ouput data for a particular **Interface**(p. 26).

**Parameters:**
*interface* Pointer to the **Interface**(p. 26)

**Returns:**
A pointer to the data array

#### 6.5.3.34 void InterfaceContainer::invalidateAll ()

Invalidate all pointers for all Interfaces.

#### 6.5.3.35 void InterfaceContainer::invalidateDx ()

Invalidate the state derivative pointer for all ContinuousInterfaces.

#### 6.5.3.36 void InterfaceContainer::invalidateParam ()

Invalidate the parameter pointer for all Interfaces.

#### 6.5.3.37 void InterfaceContainer::invalidateU ()

Invalidate the input pointer for all Interfaces.

#### 6.5.3.38 void InterfaceContainer::invalidateX ()

Invalidate the state pointer for all Interfaces.

#### 6.5.3.39 void InterfaceContainer::invalidateY ()

Invalidate the output pointer for all Interfaces.

#### 6.5.3.40 void InterfaceContainer::printInfo ()

Prints an **Interface**(p. 26) map for all interfaces.

### 6.5.4 Member Data Documentation

#### 6.5.4.1 const double InterfaceContainer::SAMPLE_OFFSET_CONTINUOUS = -1.0 [static]

#### 6.5.4.2 const double InterfaceContainer::SAMPLE_TIME_CONTINUOUS = -1.0 [static]

The documentation for this class was generated from the following files:

- code/include/**sim_interface_container.h**
- code/src/**sim_interface_container.cpp**

## 6.6 Sim Class Reference

Simulation class.

`#include <sim_class.h>`

### Public Member Functions

- **Sim** (SimStruct *s)

  *Sim class constructor.*

- virtual ~**Sim** ()

  *Sim class Destructor.*

- virtual void **start** ()=0

  *Used to set up initial conditions.*

- virtual void **init_cond** ()=0

  *Used to set up initial conditions.*

- virtual void **update** ()=0

  *Called one evey major integration time-step.*

- virtual void **derivative** ()=0

  *Used to compute the derivatives witch should be placed in Interface.dx().*

- virtual void **output** ()=0

  *Used to send values to S-Function output via* **Interface.y()**(p. 29).

- virtual void **terminate** ()=0

  *Called one at the end of the simulation to initiate any termination routines.*

- SimStruct * **getS** ()

  *Get the pointer to the SimStruct.*

- **InterfaceContainer** * **getInterfaceContainer** ()

  *Get a pointer to the* **InterfaceContainer**(p. 30) *class object.*

- **ContinuousInterface** * **addContinuousInterface** (const char *p_name, const unsigned int p_inputs, const unsigned int p_outputs, const unsigned int p_states, const unsigned int p_params)

  *Add a new* **ContinuousInterface**(p. 17) *object, and return a pointer to it.*

- **DiscreteInterface** * **addDiscreteInterface** (const char *p_name, const unsigned int p_inputs, const unsigned int p_outputs, const unsigned int p_states, const unsigned int p_params, const double p_inputSampleTime, const double p_outputSampleTime, const double p_inputSampleOffset=0.0, const double p_outputSampleOffset=0.0)

  *Add a new* **DiscreteInterface**(p. 20) *object, and return a pointer to it.*

- **DiscreteInterface** * **addDiscreteInterface** (const char *p_name, const unsigned int p_inputs, const unsigned int p_outputs, const unsigned int p_states, const unsigned int p_params, const double p_sampleTime)

  *Add a new* **DiscreteInterface**(p. 20) *object, and return a pointer to it.*

- double **getSimTime** () const
- bool **runOnce** (const double time) const
- bool **runOnce** () const

### 6.6.1 Detailed Description

Simulation class.

**Author:**
Lee Netherton

The Sim class acts as a continer for the entire simulation. The following functions are called whils the simulation is running:

- **init_cond( )**(p. 43)
- **update( )**(p. 44)
- **derivative( )**(p. 43)
- **output( )**(p. 43)
- **terminate( )**(p. 44) (see inividual functions for their uses)

Simulation inputs, ouputs, states, and parameter vectors are accessed via **Interface**(p. 26) classes. These **Interface**(p. 26) classes are instantiated and kept in an **InterfaceContainer**(p. 30) class. New interfaces can be registered with the **InterfaceContainer**(p. 30) class.

The **Exception**(p. 24) class can be used to flag exeptions. The **Exception::kill()**(p. 25) function will stop the simulation and print the exeption that caused it to the screen.

### 6.6.2 Constructor & Destructor Documentation

#### 6.6.2.1 Sim::Sim (SimStruct * *s*)

Sim class constructor.

**Parameters:**
***s*** Pointer to SimStruct structure (stored in m_S).

#### 6.6.2.2 Sim::~Sim () [virtual]

Sim class Destructor.

Virtual - the derived class's destructor will always need to be called, as it may allocate memory.

### 6.6.3 Member Function Documentation

#### 6.6.3.1 ContinuousInterface* Sim::addContinuousInterface (const char * *p_ name*, const unsigned int *p_ inputs*, const unsigned int *p_ outputs*, const unsigned int *p_ states*, const unsigned int *p_ params*) [inline]

Add a new **ContinuousInterface**(p. 17) object, and return a pointer to it.

**Parameters:**

***p_ name*** Name of the interface.

***p_ inputs*** Number of inputs required.

***p_ outputs*** Number of outputs required.

***p_ states*** Number of states required.

***p_ params*** Number of params required.

**Returns:**

Pointer to **ContinuousInterface**(p. 17).

**See also:**

**ContinuousInterface**(p. 17)

#### 6.6.3.2 DiscreteInterface* Sim::addDiscreteInterface (const char * *p_ name*, const unsigned int *p_ inputs*, const unsigned int *p_ outputs*, const unsigned int *p_ states*, const unsigned int *p_ params*, const double *p_ sampleTime*) [inline]

Add a new **DiscreteInterface**(p. 20) object, and return a pointer to it.

**Parameters:**

***p_ name*** Name of interface

***p_ inputs*** Number of inputs required.

***p_ outputs*** Number of outputs required.

***p_ states*** Number of states required.

***p_ params*** Number of params required.

***p_ sampleTime*** Input and output sample time in seconds

**Returns:**

Pointer to **DiscreteInterface**(p. 20).

**See also:**

**DiscreteInterface**(p. 20)

#### 6.6.3.3 DiscreteInterface* Sim::addDiscreteInterface (const char * *p_ name*, const unsigned int *p_ inputs*, const unsigned int *p_ outputs*, const unsigned int *p_ states*, const unsigned int *p_ params*, const double *p_ inputSampleTime*, const double *p_ outputSampleTime*, const double *p_ inputSampleOffset* = 0.0, const double *p_ outputSampleOffset* = 0.0) [inline]

Add a new **DiscreteInterface**(p. 20) object, and return a pointer to it.

**Parameters:**

***p_name*** Name of interface

***p_inputs*** Number of inputs required.

***p_outputs*** Number of outputs required.

***p_states*** Number of states required.

***p_params*** Number of params required.

***p_inputSampleTime*** Input sample time in seconds

***p_outputSampleTime*** Output sample time in seconds

***p_inputSampleOffset*** Input sample time offset in seconds

***p_outputSampleOffset*** Output sample time offset in seconds

**Returns:**

Pointer to **DiscreteInterface**(p. 20).

**See also:**

**DiscreteInterface**(p. 20)

#### 6.6.3.4 virtual void Sim::derivative () [pure virtual]

Used to compute the derivatives witch should be placed in Interface.dx().

#### 6.6.3.5 InterfaceContainer* Sim::getInterfaceContainer () [inline]

Get a pointer to the **InterfaceContainer**(p. 30) class object.

**Returns:**

Pointer to **InterfaceContainer**(p. 30) object

#### 6.6.3.6 SimStruct* Sim::getS () [inline]

Get the pointer to the SimStruct.

**Returns:**

Pointer to SimStruct

#### 6.6.3.7 double Sim::getSimTime () const [inline]

#### 6.6.3.8 virtual void Sim::init_cond () [pure virtual]

Used to set up initial conditions.

#### 6.6.3.9 virtual void Sim::output () [pure virtual]

Used to send values to S-Function output via **Interface.y()**(p. 29).

#### 6.6.3.10 bool Sim::runOnce () const [inline]

#### 6.6.3.11 bool Sim::runOnce (const double *time*) const

#### 6.6.3.12 virtual void Sim::start () [pure virtual]

Used to set up initial conditions.

#### 6.6.3.13 virtual void Sim::terminate () [pure virtual]

Called one at the end of the simulation to initiate any termination routines.

#### 6.6.3.14 virtual void Sim::update () [pure virtual]

Called one evey major integration time-step.

The documentation for this class was generated from the following files:

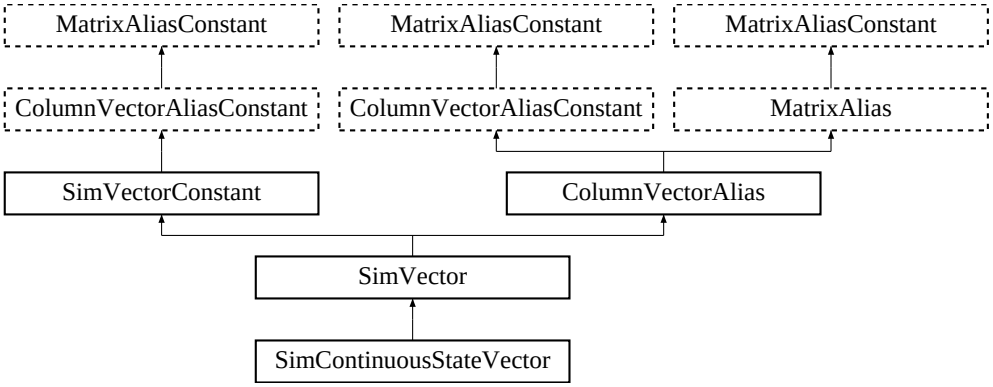
- code/include/**sim_class.h**
- code/src/**sim_class.cpp**

## 6.7 SimContinuousStateVector Class Reference

Simulation continuous state vector.

`#include <sim_vector.h>`

Inheritance diagram for SimContinuousStateVector::




### Public Member Functions

- **SimContinuousStateVector** (**ContinuousInterface** *parentInterface, const unsigned int size)

  *Sized Constructor.*

- **SimContinuousStateVector** & **operator**= (const **MatrixAliasConstant** &copy)

  *Base class assignment operator.*

- **SimContinuousStateVector** & **operator**= (const **SimContinuousStateVector** &copy)

  *Assignment operator.*

- virtual const double * **getSimIoPointer** () const

  *Get an up-to-date version of the data pointer.*

### 6.7.1 Detailed Description

Simulation continuous state vector.

**Author:**
Lee Netherton

Derived from **SimVector**(p. 59), this class is used soley for defining how the data pointer is retrieved via **getSimIoPointer()**(p. 46).

## 6.7.2 Constructor & Destructor Documentation

### 6.7.2.1 SimContinuousStateVector::SimContinuousStateVector (ContinuousInterface * *parentInterface*, const unsigned int *size*)

Sized Constructor.

To construct the full class.

**Parameters:**

***parentInterface*** Pointer to the owner class

***size*** Size of the vector

## 6.7.3 Member Function Documentation

### 6.7.3.1 const double * SimContinuousStateVector::getSimIoPointer () const [virtual]

Get an up-to-date version of the data pointer.

Implements **SimVectorConstant** (p. 62).

### 6.7.3.2 SimContinuousStateVector& SimContinuousStateVector::operator= (const SimContinuousStateVector & *copy*) [inline]

Assignment operator.

### 6.7.3.3 SimContinuousStateVector& SimContinuousStateVector::operator= (const MatrixAliasConstant & *copy*) [inline]

Base class assignment operator.

Reimplemented from **ColumnVectorAlias**.

The documentation for this class was generated from the following files:

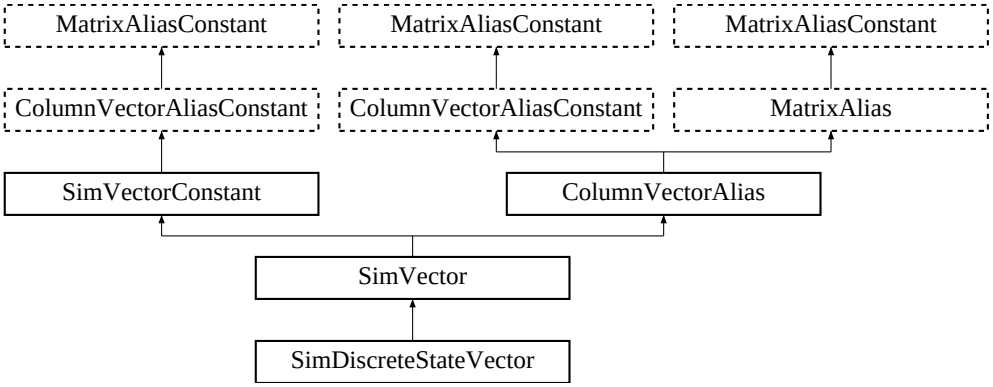
- code/include/**sim_vector.h**
- code/src/**sim_vector.cpp**

# 6.8 SimDiscreteStateVector Class Reference

Simulation discrete state vector.

`#include <sim_vector.h>`

Inheritance diagram for SimDiscreteStateVector::




## Public Member Functions

- **SimDiscreteStateVector** (**DiscreteInterface** *parentInterface, const unsigned int size)
  *Sized Constructor.*
- **SimDiscreteStateVector** & **operator**= (const **MatrixAliasConstant** &copy)
  *Base class assignment operator.*
- **SimDiscreteStateVector** & **operator**= (const **SimDiscreteStateVector** &copy)
  *Assignment operator.*
- virtual const double * **getSimIoPointer** () const
  *Get an up-to-date version of the data pointer.*

### 6.8.1 Detailed Description

Simulation discrete state vector.

**Author:**
Lee Netherton

Derived from **SimVector**(p. 59), this class is used soley for defining how the data pointer is retrieved via **getSimIoPointer()**(p. 48).

### 6.8.2 Constructor & Destructor Documentation

#### 6.8.2.1 SimDiscreteStateVector::SimDiscreteStateVector (DiscreteInterface * *parentInterface*, const unsigned int *size*)

Sized Constructor.

To construct the full class.

**Parameters:**

***parentInterface*** Pointer to the owner class

***size*** Size of the vector

### 6.8.3 Member Function Documentation

#### 6.8.3.1 const double * SimDiscreteStateVector::getSimIoPointer () const [virtual]

Get an up-to-date version of the data pointer.

Implements **SimVectorConstant** (p. 62).

#### 6.8.3.2 SimDiscreteStateVector& SimDiscreteStateVector::operator= (const SimDiscreteStateVector & *copy*) [inline]

Assignment operator.

#### 6.8.3.3 SimDiscreteStateVector& SimDiscreteStateVector::operator= (const MatrixAliasConstant & *copy*) [inline]

Base class assignment operator.

Reimplemented from **ColumnVectorAlias**.

The documentation for this class was generated from the following files:

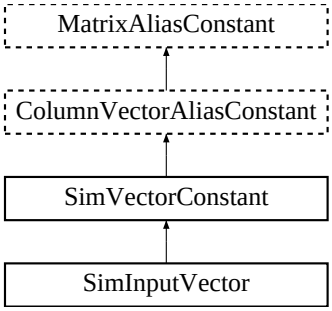
- code/include/**sim_vector.h**
- code/src/**sim_vector.cpp**

## 6.9 SimInputVector Class Reference

Simulation input vector.

`#include <sim_vector.h>`

Inheritance diagram for SimInputVector::




### Public Member Functions

- **SimInputVector** (**Interface** *parentInterface, const unsigned int size)

  *Sized Constructor.*

- virtual const double * **getSimIoPointer** () const

  *Get an up-to-date version of the data pointer.*

### 6.9.1 Detailed Description

Simulation input vector.

**Author:**
Lee Netherton

Derived from **SimVectorConstant**(p. 61), this class is used soley for defining how the data pointer is retrieved via **getSimIoPointer()**(p. 50).

### 6.9.2 Constructor & Destructor Documentation

#### 6.9.2.1 SimInputVector::SimInputVector (Interface * *parentInterface*, const unsigned int *size*) `[inline]`

Sized Constructor.

To construct the full class.

**Parameters:**
***parentInterface*** Pointer to the owner class

***size*** Size of the vector

### 6.9.3 Member Function Documentation

#### 6.9.3.1 const double * SimInputVector::getSimIoPointer () const [virtual]

Get an up-to-date version of the data pointer.

Implements **SimVectorConstant** (p. 62).

The documentation for this class was generated from the following files:

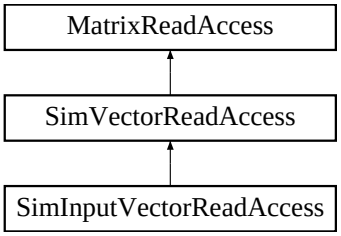
- code/include/**sim_vector.h**
- code/src/**sim_vector.cpp**

# 6.10 SimInputVectorReadAccess Class Reference

Read access for the **SimInputVector**(p. 49) class.

`#include <sim_vector.h>`

Inheritance diagram for SimInputVectorReadAccess::




## Public Member Functions

- **SimInputVectorReadAccess** (**SimVectorConstant** *SVC)

  *Data pointer valid flag Constructor.*

- virtual const double & **readElement** (const unsigned int row, const unsigned int column)
- virtual const double & **readElement** (const unsigned int index)

## 6.10.1 Detailed Description

Read access for the **SimInputVector**(p. 49) class.

**Author:**
Lee Netherton

The SimInputVectorReadAccess class provides virtual overloaded functions for the **readElement()**(p. 52) member of **MatrixReadAccess**. This is needed as Simulink input vectors are stored as double** rather than double*. The extra indirection is provided with these functions.

## 6.10.2 Constructor & Destructor Documentation

### 6.10.2.1 SimInputVectorReadAccess::SimInputVectorReadAccess (SimVectorConstant * *SVC*) [inline]

Data pointer valid flag Constructor.

Constructs the SimInputVectorReadAccess class. This is the only constructor.

**Parameters:**
***SVC*** Pointer to owner vector

### 6.10.3 Member Function Documentation

#### 6.10.3.1 const double & SimInputVectorReadAccess::readElement (const unsigned int *index*) [virtual]

Returns a read-only reference to the data member at specified position

**Parameters:**

***index*** Row-wise position of element (zero-indexed)

Reimplemented from **MatrixReadAccess**.

#### 6.10.3.2 const double & SimInputVectorReadAccess::readElement (const unsigned int *row*, const unsigned int *column*) [virtual]

Returns a read-only reference to the data member at specified position

**Parameters:**

***row*** Row position of desired element (zero indexed)

***column*** Column position of desired element (zero indexed)

Reimplemented from **MatrixReadAccess**.

The documentation for this class was generated from the following files:

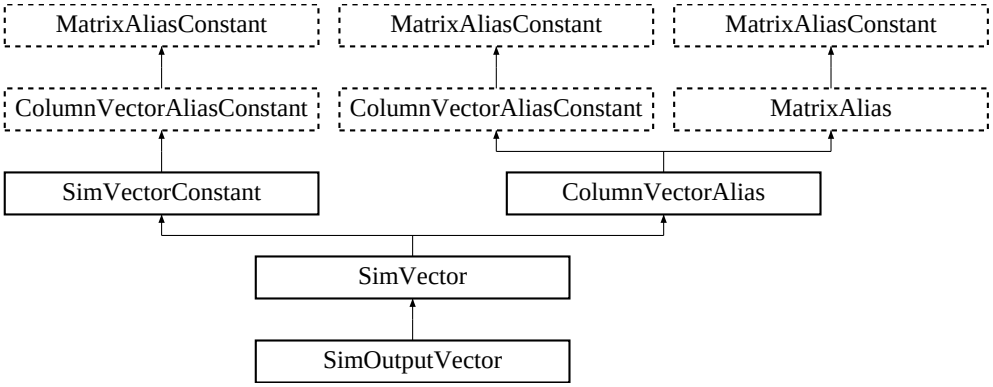
- code/include/**sim_vector.h**
- code/src/**sim_vector.cpp**

## 6.11 SimOutputVector Class Reference

Simulation output vector.

`#include <sim_vector.h>`

Inheritance diagram for SimOutputVector::




### Public Member Functions

- **SimOutputVector** (**Interface** *parentInterface, const unsigned int size)
  *Sized Constructor.*
- **SimOutputVector** & **operator**= (const **MatrixAliasConstant** &copy)
  *Base class assignment operator.*
- **SimOutputVector** & **operator**= (const **SimOutputVector** &copy)
  *Assignment operator.*
- virtual const double * **getSimIoPointer** () const
  *Get an up-to-date version of the data pointer.*

### 6.11.1 Detailed Description

Simulation output vector.

**Author:**
Lee Netherton

Derived from **SimVector**(p. 59), this class is used soley for defining how the data pointer is retrieved via **getSimIoPointer()**(p. 54).

### 6.11.2 Constructor & Destructor Documentation

#### 6.11.2.1 SimOutputVector::SimOutputVector (Interface * *parentInterface*, const unsigned int *size*) [inline]

Sized Constructor.

To construct the full class.

**Parameters:**

***parentInterface*** Pointer to the owner class

***size*** Size of the vector

### 6.11.3 Member Function Documentation

#### 6.11.3.1 const double * SimOutputVector::getSimIoPointer () const [virtual]

Get an up-to-date version of the data pointer.

Implements **SimVectorConstant** (p. 62).

#### 6.11.3.2 SimOutputVector& SimOutputVector::operator= (const SimOutputVector & *copy*) [inline]

Assignment operator.

#### 6.11.3.3 SimOutputVector& SimOutputVector::operator= (const MatrixAliasConstant & *copy*) [inline]

Base class assignment operator.

Reimplemented from **ColumnVectorAlias**.

The documentation for this class was generated from the following files:

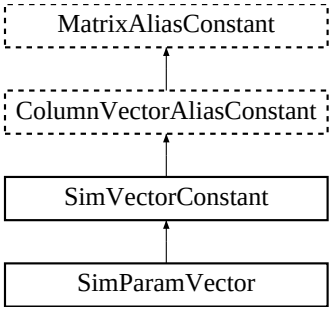
- code/include/**sim_vector.h**
- code/src/**sim_vector.cpp**

## 6.12 SimParamVector Class Reference

Simulation parameters vector.

`#include <sim_vector.h>`

Inheritance diagram for SimParamVector::




### Public Member Functions

- **SimParamVector** (**Interface** *parentInterface, const unsigned int size)

  *Sized Constructor.*

- virtual const double * **getSimIoPointer** () const

  *Get an up-to-date version of the data pointer.*

### 6.12.1 Detailed Description

Simulation parameters vector.

**Author:**
Lee Netherton

Derived from **SimVectorConstant**(p. 61), this class is used soley for defining how the data pointer is retrieved via **getSimIoPointer()**(p. 56).

### 6.12.2 Constructor & Destructor Documentation

#### 6.12.2.1 SimParamVector::SimParamVector (Interface * *parentInterface*, const unsigned int *size*) `[inline]`

Sized Constructor.

To construct the full class.

**Parameters:**
***parentInterface*** Pointer to the owner class

***size*** Size of the vector

### 6.12.3 Member Function Documentation

#### 6.12.3.1 const double * SimParamVector::getSimIoPointer () const [virtual]

Get an up-to-date version of the data pointer.

Implements **SimVectorConstant** (p. 62).

The documentation for this class was generated from the following files:

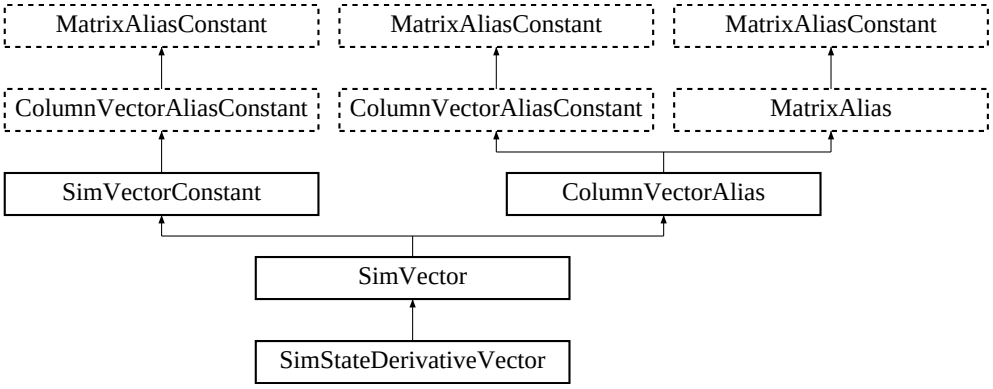
- code/include/**sim_vector.h**
- code/src/**sim_vector.cpp**

## 6.13 SimStateDerivativeVector Class Reference

Simulation state derivative vector.

`#include <sim_vector.h>`

Inheritance diagram for SimStateDerivativeVector::




### Public Member Functions

- **SimStateDerivativeVector** (**ContinuousInterface** *parentInterface, const unsigned int size)

  *Sized Constructor.*

- **SimStateDerivativeVector** & **operator**= (const **MatrixAliasConstant** &copy)

  *Base class assignment operator.*

- **SimStateDerivativeVector** & **operator**= (const **SimStateDerivativeVector** &copy)

  *Assignment operator.*

- virtual const double * **getSimIoPointer** () const

  *Get an up-to-date version of the data pointer.*

### 6.13.1 Detailed Description

Simulation state derivative vector.

**Author:**
Lee Netherton

Derived from **SimVector**(p. 59), this class is used soley for defining how the data pointer is retrieved via **getSimIoPointer()**(p. 58).

## 6.13.2 Constructor & Destructor Documentation

### 6.13.2.1 SimStateDerivativeVector::SimStateDerivativeVector (ContinuousInterface * *parentInterface*, const unsigned int *size*)

Sized Constructor.

To construct the full class.

**Parameters:**

***parentInterface*** Pointer to the owner class

***size*** Size of the vector

## 6.13.3 Member Function Documentation

### 6.13.3.1 const double * SimStateDerivativeVector::getSimIoPointer () const [virtual]

Get an up-to-date version of the data pointer.

Implements **SimVectorConstant** (p. 62).

### 6.13.3.2 SimStateDerivativeVector& SimStateDerivativeVector::operator= (const SimStateDerivativeVector & *copy*) [inline]

Assignment operator.

### 6.13.3.3 SimStateDerivativeVector& SimStateDerivativeVector::operator= (const MatrixAliasConstant & *copy*) [inline]

Base class assignment operator.

Reimplemented from **ColumnVectorAlias**.

The documentation for this class was generated from the following files:

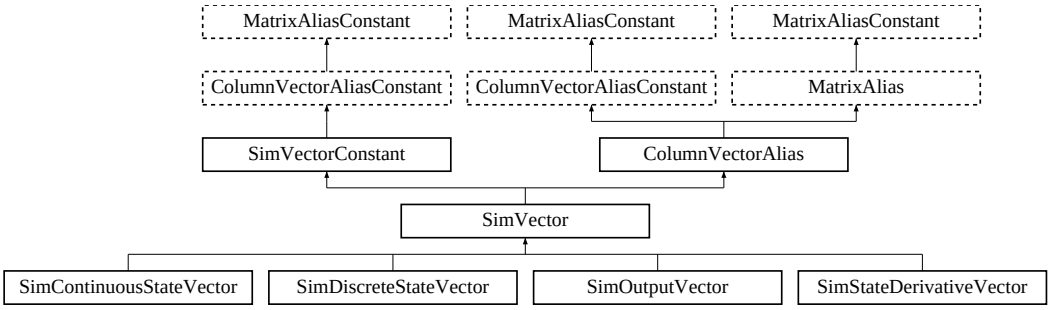
- code/include/**sim_vector.h**
- code/src/**sim_vector.cpp**

## 6.14 SimVector Class Reference

Base class for SimVector descendants.

`#include <sim_vector.h>`

Inheritance diagram for SimVector::




### Public Member Functions

- **SimVector** (**Interface** *parentInterface, const unsigned int size)
  *Sized Constructor.*
- virtual ~**SimVector** ()
  *SimVector Destructor.*
- virtual void **invalidate** ()
  *Invalidate the data pointer.*

### Protected Member Functions

- void **_constructSimVector** (**Interface** *parentInterface)
  *Sized Constructor.*

### Protected Attributes

- **SimVectorWriteAccess** * **m_simVectorWriteAccess**
  *Pointer to the WriteAccess object.*

### 6.14.1 Detailed Description

Base class for SimVector descendants.

**Author:**
Lee Netherton

This is an abstract class which provides a base for read/write SimVector's like **SimStateDerivativeVector**(p. 57).

### 6.14.2 Constructor & Destructor Documentation

#### 6.14.2.1 SimVector::SimVector (Interface * *parentInterface*, const unsigned int *size*) [inline]

Sized Constructor.

To construct the full class.

**Parameters:**

***parentInterface*** Pointer to the owner class

***size*** Size of the vector

#### 6.14.2.2 SimVector::~SimVector () [virtual]

SimVector Destructor.

Virtual - the lowest derived class will always need to be called, as they all allocate memory in ther own ways.

### 6.14.3 Member Function Documentation

#### 6.14.3.1 void SimVector::_constructSimVector (Interface * *parentInterface*) [protected]

Sized Constructor.

To construct the full class manually.

**Parameters:**

***parentInterface*** Pointer to the owner class

#### 6.14.3.2 virtual void SimVector::invalidate () [inline, virtual]

Invalidate the data pointer.

Reimplemented from **SimVectorConstant** (p. 63).

### 6.14.4 Member Data Documentation

#### 6.14.4.1 SimVectorWriteAccess* SimVector::m_simVectorWriteAccess [protected]

Pointer to the WriteAccess object.

The documentation for this class was generated from the following files:

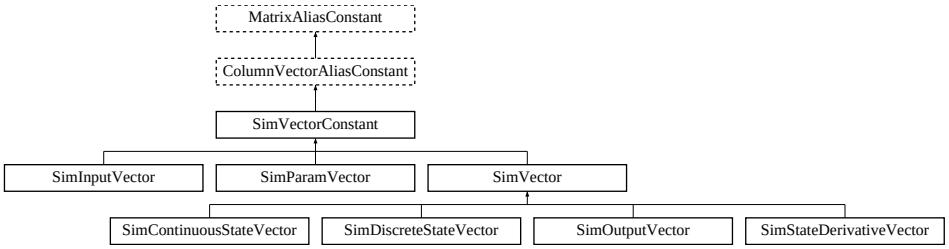
- code/include/**sim_vector.h**
- code/src/**sim_vector.cpp**

## 6.15 SimVectorConstant Class Reference

Base class for SimVectorConstant descendants.

`#include <sim_vector.h>`

Inheritance diagram for SimVectorConstant::




### Public Member Functions

- **SimVectorConstant** (**Interface** *parentInterface, const unsigned int size)
  *Sized Constructor.*
- virtual ~**SimVectorConstant** ()
  *SimVectorConstant Destructor.*
- virtual void **invalidate** ()
  *Invalidate the data pointer.*
- virtual const double * **getSimIoPointer** () const =0
  *Get an up-to-date version of the data pointer.*

### Protected Member Functions

- void **_constructSimVectorConstant** (**Interface** *parentInterface)
  *Sized Constructor.*

### Protected Attributes

- **InterfaceContainer** * **m_interfaceContainer**
  *Pointer to the* **InterfaceContainer**(p. 30) *class.*
- **Interface** * **m_pInterface**
  *Pointer to* **Interface**(p. 26) *class in which this vector exists.*
- **SimVectorReadAccess** * **m_simVectorReadAccess**
  *Pointer to the ReadAccess object.*

### 6.15.1 Detailed Description

Base class for SimVectorConstant descendants.

**Author:**
Lee Netherton

This is an abstract class which provides a base for read only SimVector's like **SimInputVector**(p. 49). The main functionality that it provides is the ability to obtain its own data pointer from the simulation structure. This is provided via **getSimIoPointer()**(p. 62). To aid the retreval of this pointer, the class is equiped with a pointer to the **InterfaceContainer**(p. 30) class (m_interfaceContainer) and the id number of the **Interface**(p. 26) (m_interfaceId).

### 6.15.2 Constructor & Destructor Documentation

#### 6.15.2.1 SimVectorConstant::SimVectorConstant (Interface * *parentInterface*, const unsigned int *size*) [inline]

Sized Constructor.

To construct the full class.

**Parameters:**
***parentInterface*** Pointer to the owner class
***size*** Size of the vector

#### 6.15.2.2 SimVectorConstant::~SimVectorConstant () [virtual]

SimVectorConstant Destructor.

Virtual - the lowest derived class will always need to be called, as they all allocate memory in ther own ways.

### 6.15.3 Member Function Documentation

#### 6.15.3.1 void SimVectorConstant::_constructSimVectorConstant (Interface * *parentInterface*) [protected]

Sized Constructor.

To construct the full class manually.

**Parameters:**
***parentInterface*** Pointer to the owner class

#### 6.15.3.2 virtual const double* SimVectorConstant::getSimIoPointer () const [pure virtual]

Get an up-to-date version of the data pointer.

Implemented in **SimInputVector** (p. 50), **SimParamVector** (p. 56), **SimOutputVector** (p. 54), **SimContinuousStateVector** (p. 46), **SimDiscreteStateVector** (p. 48), and **SimStateDerivativeVector** (p. 58).

#### 6.15.3.3 virtual void SimVectorConstant::invalidate () [inline, virtual]

Invalidate the data pointer.

Reimplemented in **SimVector** (p. 60).

### 6.15.4 Member Data Documentation

#### 6.15.4.1 InterfaceContainer* SimVectorConstant::m_interfaceContainer [protected]

Pointer to the **InterfaceContainer**(p. 30) class.

#### 6.15.4.2 Interface* SimVectorConstant::m_pInterface [protected]

Pointer to **Interface**(p. 26) class in which this vector exists.

#### 6.15.4.3 SimVectorReadAccess* SimVectorConstant::m_simVectorReadAccess [protected]

Pointer to the ReadAccess object.

The documentation for this class was generated from the following files:

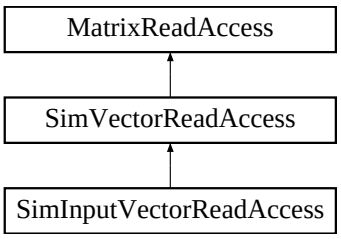
- code/include/**sim_vector.h**
- code/src/**sim_vector.cpp**

## 6.16 SimVectorReadAccess Class Reference

Read access for the **SimVectorConstant**(p. 61) and **SimVector**(p. 59) classes.

`#include <sim_vector.h>`

Inheritance diagram for SimVectorReadAccess::




### Public Member Functions

- **SimVectorReadAccess** (**SimVectorConstant** *SVC)
  *Constructor.*
- void **invalidate** ()
  *Invalidates the data pointer.*
- virtual const double * **getDataPointer** ()
  *Returns the a pointer to the data array.*
- void **updateDataPointer** ()
  *Updates the data pointer.*

### Protected Attributes

- **SimVectorConstant** * **m_simVectorConstant**
  *Poiner to the owner class.*
- bool **m_dataPointerOk**
  *Data pointer valid flag.*

### 6.16.1 Detailed Description

Read access for the **SimVectorConstant**(p. 61) and **SimVector**(p. 59) classes.

**Author:**
Lee Netherton

The SimvectorReadAccess class provides a means of ensuring that the SimVector's data pointer is up-to-date. Validity is maintained automatically using the **getDataPointer()**(p. 65) function.

To in validate the data pointer the **invalidate()**(p. 65) function can be used. The pointer will then be automatically updated upon the next **getDataPointer()**(p. 65) call. Manual updating of the pointer can be forced using the **updateDataPointer()**(p. 65) function.

## 6.16.2 Constructor & Destructor Documentation

### 6.16.2.1 SimVectorReadAccess::SimVectorReadAccess (SimVectorConstant * *SVC*) [inline]

Constructor.

Constructs the SimVectorReadAccess class. This is the only constructor.

**Parameters:**

***SVC*** Pointer to owner vector

## 6.16.3 Member Function Documentation

### 6.16.3.1 const double * SimVectorReadAccess::getDataPointer () [virtual]

Returns the a pointer to the data array.

Reimplemented from **MatrixReadAccess**.

### 6.16.3.2 void SimVectorReadAccess::invalidate () [inline]

Invalidates the data pointer.

### 6.16.3.3 void SimVectorReadAccess::updateDataPointer ()

Updates the data pointer.

## 6.16.4 Member Data Documentation

### 6.16.4.1 bool SimVectorReadAccess::m_dataPointerOk [protected]

Data pointer valid flag.

### 6.16.4.2 SimVectorConstant* SimVectorReadAccess::m_simVectorConstant [protected]

Poiner to the owner class.

The documentation for this class was generated from the following files:

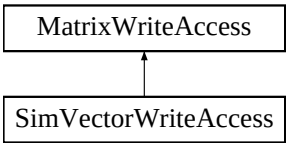
- code/include/**sim_vector.h**
- code/src/**sim_vector.cpp**

## 6.17 SimVectorWriteAccess Class Reference

Write access for the **SimVector**(p. 59) classes.

`#include <sim_vector.h>`

Inheritance diagram for SimVectorWriteAccess::




### Public Member Functions

- **SimVectorWriteAccess** (**SimVector** *SV)
  *Constructor.*
- void **invalidate** ()
  *Invalidates the data pointer.*
- virtual double * **getDataPointer** ()
  *Returns the a pointer to the data array.*
- void **updateDataPointer** ()
  *Updates the data pointer.*

### 6.17.1 Detailed Description

Write access for the **SimVector**(p. 59) classes.

**Author:**
Lee Netherton

The SimvectorWriteAccess class provides a means of ensuring that the SimVector's data pointer is up-to-date. Validity is maintained automatically using the **getDataPointer()**(p. 67) function. To in validate the data pointer the **invalidate()**(p. 67) function can be used. The pointer will then be automatically updated upon the next **getDataPointer()**(p. 67) call. Manual updating of the pointer can be forced using the **updateDataPointer()**(p. 67) function.

### 6.17.2 Constructor & Destructor Documentation

#### 6.17.2.1 SimVectorWriteAccess::SimVectorWriteAccess (SimVector * *SV*) [inline]

Constructor.

Constructs the SimVectorWriteAccess class. This is the only constructor.

**Parameters:**
***SV*** Pointer to owner vector

## 6.17.3 Member Function Documentation

### 6.17.3.1 double * SimVectorWriteAccess::getDataPointer () [virtual]

Returns the a pointer to the data array.

Reimplemented from **MatrixWriteAccess**.

### 6.17.3.2 void SimVectorWriteAccess::invalidate () [inline]

Invalidates the data pointer.

### 6.17.3.3 void SimVectorWriteAccess::updateDataPointer ()

Updates the data pointer.

The documentation for this class was generated from the following files:

- code/include/**sim_vector.h**
- code/src/**sim_vector.cpp**

# Chapter 7

# Sim File Documentation

## 7.1 code/include/sim.h File Reference

```
#include "iostream.h"
#include "simstruc.h"
#include "sim_class.h"
#include "sim_interface_container.h"
#include "sim_interface.h"
#include "sim_exception.h"
#include "sim_functions.h"
#include "sim_vector.h"
```

### Functions

- **Sim** ∗ **createSim** (SimStruct ∗S)

### 7.1.1 Function Documentation

#### 7.1.1.1 Sim∗ createSim (SimStruct ∗ *S*)

## 7.2 code/include/sim_class.h File Reference

```
#include "sim_interface_container.h"
```

### Classes

- class **Sim**

  *Simulation class.*

# 7.3 code/include/sim_exception.h File Reference

```
#include "sim.h"
```

## Classes

- class **Exception**

  *Exception class.*

## 7.4 code/include/sim_functions.h File Reference

```
#include "simstruc.h"
```

### Functions

- void **SimInitializeSizes** (SimStruct *S, **Sim** *sim_temp)
- void **SimInitializeSampleTimes** (SimStruct *S)
- void **SimStart** (SimStruct *S)
- void **SimInitializeConditions** (SimStruct *S)
- void **SimUpdate** (SimStruct *S, int_T tid)
- void **SimDerivatives** (SimStruct *S)
- void **SimOutputs** (SimStruct *S, int_T tid)
- void **SimTerminate** (SimStruct *S)

### 7.4.1 Function Documentation

#### 7.4.1.1 void SimDerivatives (SimStruct * *S*)

#### 7.4.1.2 void SimInitializeConditions (SimStruct * *S*)

#### 7.4.1.3 void SimInitializeSampleTimes (SimStruct * *S*)

#### 7.4.1.4 void SimInitializeSizes (SimStruct * *S*, Sim * *sim_ temp*)

#### 7.4.1.5 void SimOutputs (SimStruct * *S*, int_T *tid*)

#### 7.4.1.6 void SimStart (SimStruct * *S*)

#### 7.4.1.7 void SimTerminate (SimStruct * *S*)

#### 7.4.1.8 void SimUpdate (SimStruct * *S*, int_T *tid*)

# 7.5 code/include/sim_interface.h File Reference

```
#include "iostream.h"
#include "string.h"
#include "simstruc.h"
#include "sim_vector.h"
```

## Classes

- class **Interface**
  *Interface class.*
- class **ContinuousInterface**
  *ContinuousInterface class.*
- class **DiscreteInterface**
  *DiscreteInterface class.*

## 7.6 code/include/sim_interface_container.h File Reference

```
#include "sim_interface.h"
#include "iostream.h"
#include "vector.h"
#include "simstruc.h"
```

### Classes

- class **InterfaceContainer**

  *Simulation interface container class.*

# 7.7 code/include/sim_vector.h File Reference

```
#include "../../../MatrixClassLib/code/include/matrix.h"
#include "../../../MatrixClassLib/code/include/matrix_container.h"
#include <iostream>
```

## Classes

- class **SimVectorReadAccess**
  *Read access for the* **SimVectorConstant**(p. 61) *and* **SimVector**(p. 59) *classes.*
- class **SimInputVectorReadAccess**
  *Read access for the* **SimInputVector**(p. 49) *class.*
- class **SimVectorWriteAccess**
  *Write access for the* **SimVector**(p. 59) *classes.*
- class **SimVectorConstant**
  *Base class for SimVectorConstant descendants.*
- class **SimVector**
  *Base class for SimVector descendants.*
- class **SimInputVector**
  *Simulation input vector.*
- class **SimParamVector**
  *Simulation parameters vector.*
- class **SimOutputVector**
  *Simulation output vector.*
- class **SimContinuousStateVector**
  *Simulation continuous state vector.*
- class **SimDiscreteStateVector**
  *Simulation discrete state vector.*
- class **SimStateDerivativeVector**
  *Simulation state derivative vector.*

## 7.8 code/src/sim_class.cpp File Reference

```
#include "../include/sim_class.h"
```

## 7.9 code/src/sim_exception.cpp File Reference

```
#include "../include/sim_exception.h"
#include "../include/sim.h"
```

## 7.10 code/src/sim_functions.cpp File Reference

```
#include "../include/sim.h"
```

### Defines

- #define **Sim_START**
- #define **Sim_INITIALIZE_CONDITIONS**
- #define **Sim_UPDATE**
- #define **Sim_DERIVATIVES**

### Functions

- void **SimInitializeSizes** (SimStruct *S, **Sim** *sim_temp)
- void **SimInitializeSampleTimes** (SimStruct *S)
- void **SimStart** (SimStruct *S)
- void **SimInitializeConditions** (SimStruct *S)
- void **SimUpdate** (SimStruct *S, int_T tid)
- void **SimDerivatives** (SimStruct *S)
- void **SimOutputs** (SimStruct *S, int_T tid)
- void **SimTerminate** (SimStruct *S)

### 7.10.1 Define Documentation

#### 7.10.1.1 #define Sim_DERIVATIVES

#### 7.10.1.2 #define Sim_INITIALIZE_CONDITIONS

#### 7.10.1.3 #define Sim_START

#### 7.10.1.4 #define Sim_UPDATE

### 7.10.2 Function Documentation

#### 7.10.2.1 void SimDerivatives (SimStruct * *S*)

#### 7.10.2.2 void SimInitializeConditions (SimStruct * *S*)

#### 7.10.2.3 void SimInitializeSampleTimes (SimStruct * *S*)

#### 7.10.2.4 void SimInitializeSizes (SimStruct * *S*, Sim * *sim_temp*)

#### 7.10.2.5 void SimOutputs (SimStruct * *S*, int_T *tid*)

#### 7.10.2.6 void SimStart (SimStruct * *S*)

#### 7.10.2.7 void SimTerminate (SimStruct * *S*)

#### 7.10.2.8 void SimUpdate (SimStruct * *S*, int_T *tid*)

## 7.11 code/src/sim_interface.cpp File Reference

```
#include "../include/sim_interface.h"
#include "../include/sim_class.h"
#include "../include/sim_exception.h"
```

## 7.12 code/src/sim_interface_container.cpp File Reference

```
#include "../include/sim_interface_container.h"
#include "../include/sim_interface.h"
#include "../include/sim_exception.h"
#include "../include/sim_class.h"
```

## 7.13 code/src/sim_vector.cpp File Reference

```
#include "../include/sim_vector.h"
#include "../include/sim_interface_container.h"
#include "../include/sim_interface.h"
```

# Index