Keysight Technologies

SystemVue

Electronic System-Level Design

Application Note

Creating C++ Algorithms in SystemVue Using Model Builder

KEYSIGHT TECHNOLOGIES

Overview

The SystemVue C++ Model Builder interface provides a powerful mechanism for exploring signal processing algorithms for communications system design. Modeling in SystemVue can be performed several ways: using native models in the SystemVue Graphical User Interface (GUI), programming with inline (interpreted) math language, or programming in a high-level compiled language (e.g., C++). Modeling in a high-level language offers several advantages. It:

– Provides the highest simulation speed

– Enables greater transportability to/from other environments, with fewer licensing constraints

– Allows existing Intellectual Property (IP) to be used at a system-level, thereby enabling co-verification of IP across Baseband, RF, and Test & Measurement domains

This application note outlines the process of packaging a C++ model for use in SystemVue. This process consists of just a few simple steps. Supported SystemVue users may download the workspace associated with this application note and access additional resources at:

– Keysight Technologies, Inc. EEsof EDA Knowledge Center (login required):

  http://edocs.soco.keysight.com/display/eesofkcsysvue/Using+the+Model+Builder+Interface

To follow this application note using SystemVue, the following software is suggested:

– W1461BP SystemVue core environment, or higher bundle

  – SystemVue release 2012.06, or higher

  – The W1718EP C++ Code Generator is not required, but may be helpful

– Microsoft Visual C++2010 Express Edition (free download), or a more complete Microsoft Visual Studio environment

When to use the SystemVue Model Builder Interface

– To include an existing C++ model in a simulation

– To make a model easy to re-use, or to create a company library

– To externalize key parameters or hide levels of hierarchy, for ease-of-use

– To protect custom IP

– For improved simulation speed, versus graphical or interpreted languages

Model Builder Architecture

SystemVue uses CMake to create Model Builder projects and Microsoft Visual Studio solutions. CMake is a cross-platform build system that allows users to build source code and generate projects for commonly used Integrated Development Environments (IDEs) like Visual Studio and Eclipse. CMake uses the concept of out-of-source builds. SystemVue C++ Model Builder is layered on top of CMake (Figure 1).

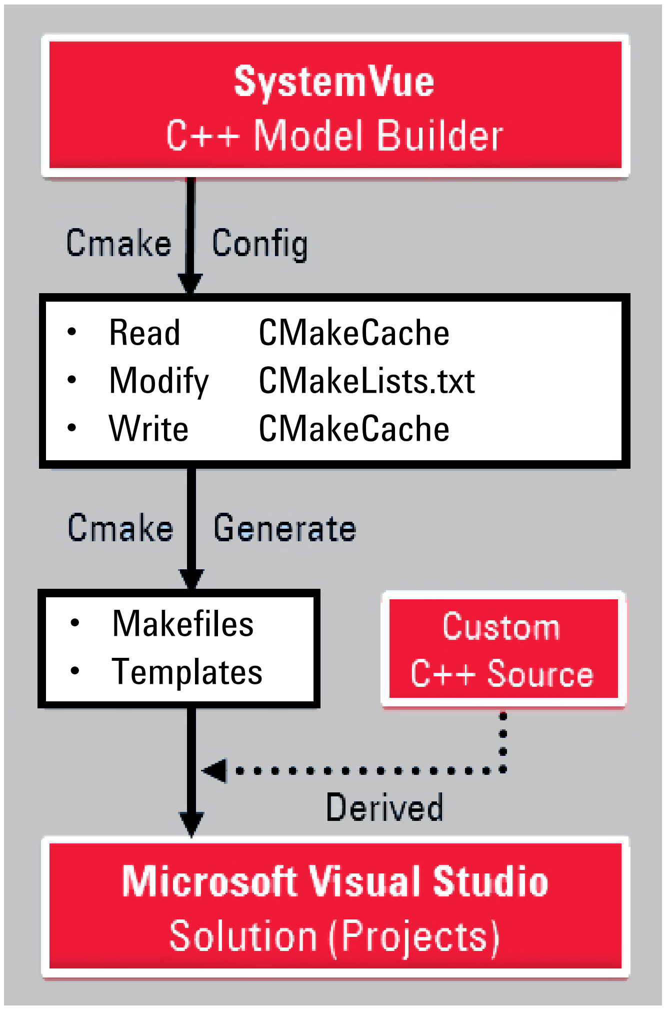


Figure 1. The SystemVue model building flow. The focus of this application note is highlighted in color.

About this C++ Model

This application note instantiates a C++ model for a Quadrature Amplitude Modulation (QAM) modulation mapper that creates QAM-1024, QAM-2048 and QAM-4096 symbols. SystemVue workspaces were created to use and test this new model (Figure 2). Additionally, I and Q waveforms were captured at the output of the model for verification. The resulting constellations are shown in Figure 3.

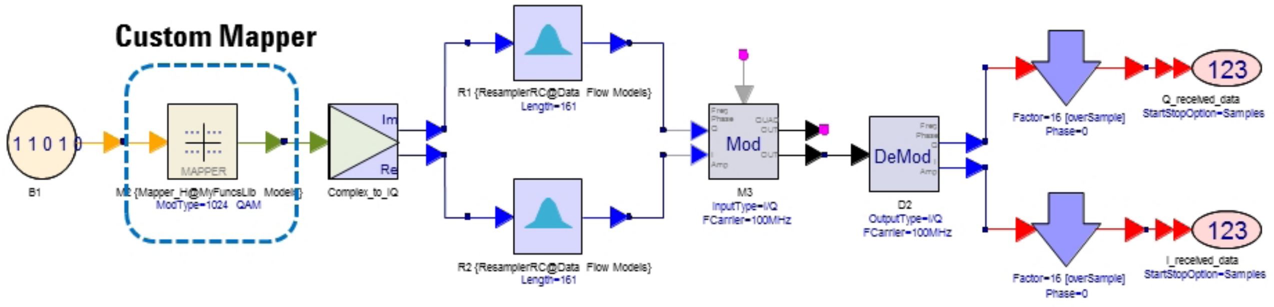


Figure 2. This application note considers the additional steps necessary to package a modulation algorithm, written in C++, for use as a custom model in SystemVue. The example itself is a symbol mapping algorithm for higher order QAM.

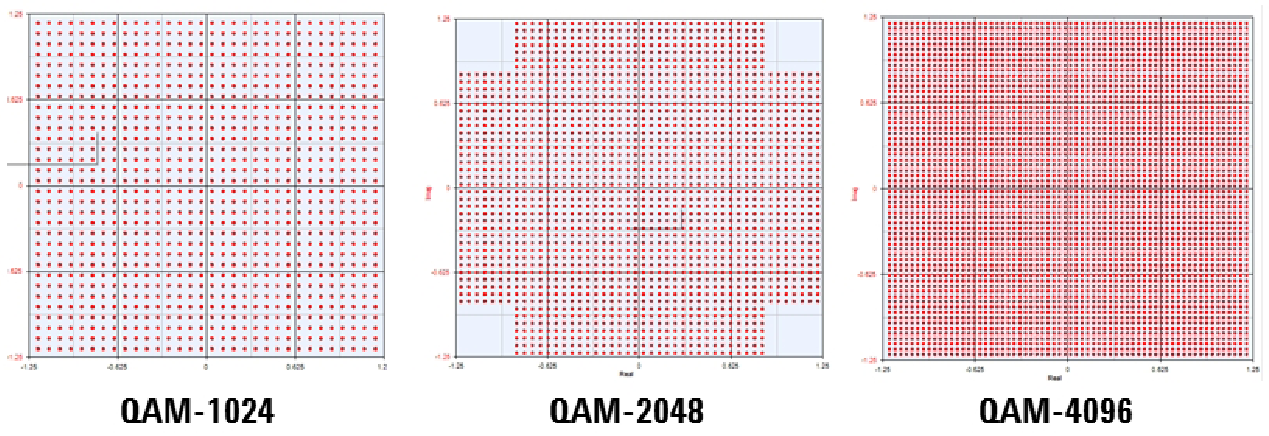


Figure 3. Shown here are the output constellations of the new C++ algorithm, as seen from a system-level analysis.

The C++ Model Building Process in SystemVue

The model building process encompasses three main steps. The remainder of this application discusses these steps at a high level to better illustrate the process. The steps are:

1. Create a SystemVue Model Builder project
2. Write a C++ dataflow model, accounting for any header and template files
3. Use the new library in SystemVue

Step 1.
Create a SystemVue Model Builder Project

1. From the **Action** menu, select **Create Model Builder Project**.
2. Choose a name for the project, such as **MyFuncsLib**, and enter it into the **Name** field. Next, choose a location for the project.
3. Click **Create** to start. A new instance of Visual Studio will be opened with the created Model Builder Project solution **MyFuncsLib.sln**.

Step 2.
Write C++ Dataflow Model

Creating Source and Header Template Files

1. On the Solution Explorer window in Visual Studio, open the **CMakeLists.txt** file in the **SystemVue-MyFuncsLib** project.
2. Update this file by uncommenting the following lines. Then, change the name of the model.

Before changes	After changes

`#SVU_CREATE_MODEL`
`( MyCustomModel )`
 	
`#   MyCustomModel.h`
`#   MyCustomModel.cpp`
	`SVU_CREATE_MODEL`
	`( Mapper_H )`
	
	`Mapper_H.h`
	`Mapper_H.cpp`

3. Save the change. Right click on the **Install** project in the Solution Explorer window and click **Rebuild**. Accept all warning messages to reload **SystemVue-MyFuncsLib** project.
4. The SystemVue Model Builder can now automatically generate two template sources (.h and .cpp) for the user's easy modification.

Writing the Header File

1. On the Solution Explorer window in Visual Studio, open the **Mapper_H.h** template file in the **SystemVue-MyFuncsLib** project **Header Files** folder. The content of **Mapper_H.h** is as follows:

```
#pragma once
#include "ModelBuilder.h"

class Mapper_H : public KeysightEEsof::DFModel
{
public:
    // This Macro is required for all classes derived from DFModel
    DECLARE_MODEL_INTERFACE( Mapper_H );

    // Constructor to initialize parameters
    Mapper_H();

    //-------- Function Overloads --------
    virtual bool       Run();

    // Ports
    KeysightEEsof::CircularBuffer< double > input, output;

    // Parameter
    double Gain;
};
```

2. To modify a **Mapper_H.h** file, perform the following steps:

   Step 1.
   The class must be derived from "`KeysightEEsof::DFModel`".

   Step 2.
   Add a macro with the public access level,
   `DECLARE_MODEL_INTERFACE( ClassName );`

   Change the ClassName to the name of the class in this example, "Mapper_H." The declaration should be
   `DECLARE_MODEL_INTERFACE( Mapper_H );`

Step 3.
Add any data member needed, with the following guidlines:

I/O ports
If the data member acts as an input or output port, then it must be among one of the following types:

```
double, double *, int, int*, std::complex<double>,
std::complex<double> *,
```

or a supported `CircularBuffer` data type.

Within the SystemVue documentation, more information about the `CircularBuffer` data type is available in the section entitled "C++ User Compiled Models." The pointer data members will act as multi-rate ports, whereas scalar data members will act as uni-rate ports, and all CircularBuffer data types can act as multi-rate port with default rate of 1. Optionally, the user can also add an unsigned rate variable, or call the `SetRate()` method of a `CircularBuffer` data type.

Parameters
If the data member acts as a parameter, then it must be among one of the following types:

```
double, double *, int, int*, std::complex<double>,
       std::complex<double> *, and char *.
```

The pointer data members (except `char *`) will act as an array parameter, `char *` will act as string parameter. Scalar data members will act as non-array parameters. For each array parameter, choose an unsigned variable that holds the number of elements in the array (refer to the documentation for further details).

Otherwise, choose any valid C++ data type for a member that is not an I/O port or parameter.

Step 4.
At the very least, override the public virtual method `bool Run()` of the base class. The `Run()` method is called for each execution of the model during simulation. Optionally, the user could override virtual methods, `Setup()`, `Initialize()`, and `Finalize()`.

Step 5.
As shown in the modified header file below, the `GetTableIndex` and `Normalize` protected member functions were added. Three private member variables were also added in this model implementation. The user can add any other methods that may be desired.

Step 6.
Three complex array type variables were declared for the mapping table with external linkage. They are visible from files other than the one in which it's defined.

3. The modified header file looks like the following:

```
#pragma once
#include "ModelBuilder.h"

class Mapper_H : public KeysightEEsof::DFModel
{
public:
    DECLARE_MODEL_INTERFACE ( Mapper_H )

    //-------- Function Overloads --------
    bool Run();
    bool Initialize();
    virtual bool Setup();

    enum ModulationType {QAM1024, QAM2048 , QAM4096};

    //-------- Ports and Parameters --------
    KeysightEEsof::CircularBuffer<bool> m_input;
    KeysightEEsof::CircularBuffer< std::complex<double > > m_output;

    int nSymbolLength;
    ModulationType ModType;

protected:
    int GetTableIndex( int iSymbolLength );
    double Normalize( int iSymbolLength, std::complex<double>* pxTable );

private:
    double m_dQAM1024Norm, m_dQAM2048Norm, m_dQAM4096Norm; };

extern   std::complex<double> QAM_1024[256];
extern   std::complex<double> QAM_2048[512];
extern   std::complex<double> QAM_4096[1024];
```

Using Custom Models Inside SystemVue

Before the user proceeds to modify the **Mapper_H.cpp** C++ source code, it may be helpful to review the context of an individual model within the SystemVue runtime simulation process.

The code in `DEFINE_MODEL_INTERFACE` instantiates the model information, including such things as ports, model names, and descriptions. This allows SystemVue to draw the schematic symbol and display the part properties. In the SystemVue dataflow simulator, the values of the model parameters are set based on the user's input on each schematic instance. As shown in Figure 4, the `setup()` method of each model is then invoked to tell the dataflow simulator the input and output data rates, as well as the sample rate, if the model is a timed source. Once SystemVue knows the data rates of each model, the dataflow scheduler computes a static schedule of the system and allocates sufficient buffers for inter-model communication. Then, before simulation, the `initialize()` method is called for each model to allocate memory space and initialize values, among other things. Next, the `run()` method is called, according to the schedule. During a `run()`, a model reads samples from its input circular buffers, performs its scientific computation, and then writes output samples to its output circular buffers. After simulation, `finalize()` is called for each model to clean up their allocated memory.

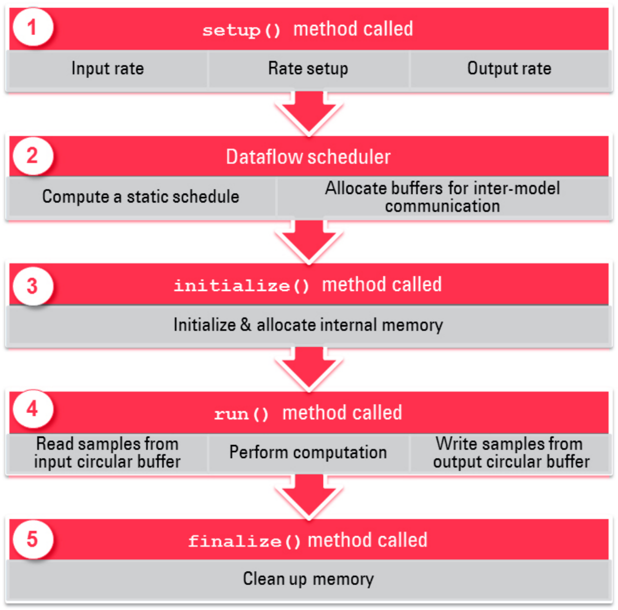

1
setup() method called
Input rate
Rate setup
Output rate
2
Dataflow scheduler
Compute a static schedule
Allocate buffers for inter-model communication
3
initialize() method called
Initialize & allocate internal memory
4
run() method called
Read samples from input circular buffer
Perform computation
Write samples from output circular buffer
5
finalize() method called
Clean up memory

Figure 4. The execution flow of a C++ model within the dataflow simulator, at runtime.

Writing the .cpp Source File

1. On the Solution Explorer window in Visual Studio, open the **Mapper_H.cpp** file in the **SystemVue-MyFuncsLib** project **Source Files** folder. The initial content of **Mapper_H.cpp** was generated by SystemVue from a template and is as follows:

```
#include "Mapper_H.h"

#ifndef SV_CODE_GEN
DEFINE_MODEL_INTERFACE ( Mapper_H )
{
    ADD_MODEL_INPUT( input );
    ADD_MODEL_OUTPUT( output );
    ADD_MODEL_PARAMETER( Gain );
    return true;
}
#endif

Mapper_H::Mapper_H()
{
    Gain = 0;
}

//-----------------------------------------------------------------
//  Run

//       Here we do the math
//-----------------------------------------------------------------
bool Mapper_H::Run()
{
    output[0] = Gain * input[0];
    return true;
}
```

2. This **Mapper_H.cpp** file must be modified by performing the following steps:

   Step 1.
   Include the Mapper_H.h header file and any other headers that the code requires. Then, use the following macro:

```
DEFINE_MODEL_INTERFACE(ClassName)
                  {
                  }
```

   to attach data members to ports or parameters (e.g., to define the interface to the simulator). Where ClassName is the name of the class in question (e.g., the user's class), the following macros can be used to attach a data member to a port or a parameter:

```
ADD_MODEL_INPUT(class_data_member);
// to attach class_data_member to an input port

ADD_MODEL_OUTPUT(class_data_member);
// to attach class_data_member to an output port

ADD_MODEL_ENUM_PARAM(class_data_member);
// to attach class_data_member
```

Step 2.
Once the headers and classes have been declared, the functionality of the user's model should be added in the Run() method.

3. If the reader is following along with these instructions, a modified **Mapper_H.cpp** file can be downloaded from the Keysight EEsof EDA Knowledge Center at:

   http://edocs.soco.keysight.com/display/eesofkcsysvue/Using+the+Model+Builder+Interface

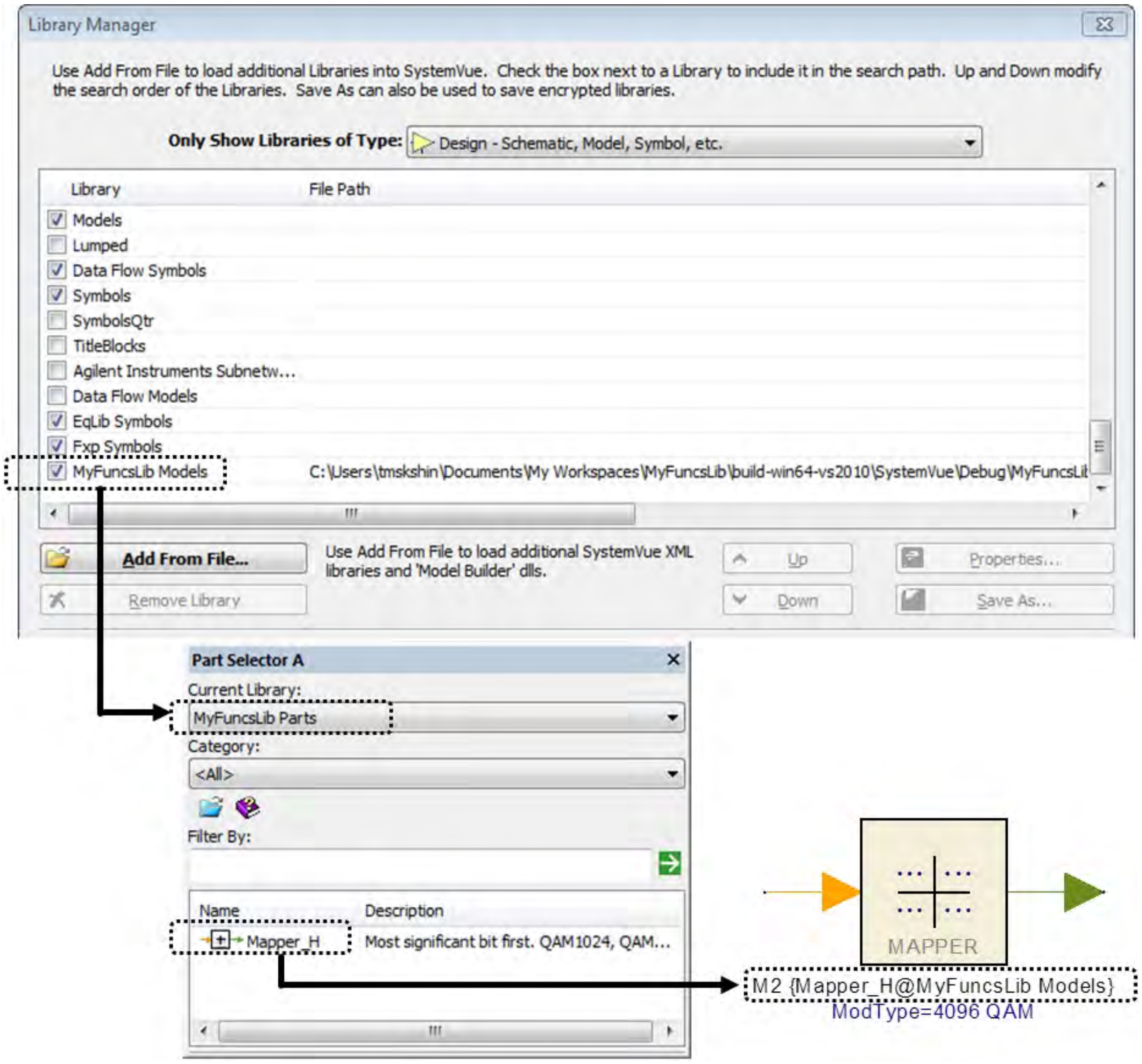
4. Recompile the Dynamic Link Library (DLL) by right-clicking mouse on **INSTALL** and selecting **Build** from the dropdown menu.

Step 3.

Use the New Library in SystemVue

Use the SystemVue Library Manager to load the new DLL and use the custom model. This can be done by first opening the SystemVue Library Manager from the **Tools/ Library Manager** menu. When the DLL has been loaded, the SystemVue Parts Selector shows the new library in the list of libraries. Select this library and place the new model on a schematic, as shown in Figure 5. More complete documentation is available on the Library Manager, or by watching the "Discovering Model Builder" video at:

www.youtube.com/watch?v=p-H-2a2L1JI.



Library Manager

Use Add From File to load additional Libraries into SystemVue. Check the box next to a Library to include it in the search path. Up and Down modify the search order of the Libraries. Save As can also be used to save encrypted libraries.

Only Show Libraries of Type: Design - Schematic, Model, Symbol, etc.

Library	File Path
✓ Models	
Lumped	
✓ Data Flow Symbols	
✓ Symbols	
SymbolsQtr	
TitleBlocks	
Agilent Instruments Subnetw...	
Data Flow Models	
✓ EqLib Symbols	
✓ Fxp Symbols	
✓ MyFuncsLib Models	C:\Users\tmskshin\Documents\My Workspaces\MyFuncsLib\build-win64-vs2010\SystemVue\Debug\MyFuncsLi

Use Add From File to load additional SystemVue XML libraries and 'Model Builder' dlls.

Part Selector A

Current Library: MyFuncsLib Parts

Category: <All>

Filter By:

Name	Description
Mapper_H	Most significant bit first. QAM1024, QAM...

Figure 5. After loading the DLL for the new library, it is added to the SystemVue parts selector. The new model is the ready to be placed on a schematic.

Conclusion

Writing a new signal processing model in C++ in SystemVue is easy and requires just a few steps. Once the "scientific" portion of the model has been defined (not shown in this document), the user simply needs to start a new model building project from the SystemVue GUI, update the templates with some model parameters (e.g., filenames, data rates and other basic information) and then merge the algorithm code into the modified template files. Using Cmake to control the C++ IDE (e.g., Visual Studio 2010) completes the compilation and prepares the DLL for SystemVue. The final steps involve importing the DLL into the SystemVue Library Manager and then verifying the model in simulation.

Further Reading

Workspace files

Supported users can download additional files from Knowledge Center (login required) at

http://edocs.soco.keysight.com/display/eesofkcsysvue/Using+the+Model+Builder+Interface

Videos

The following video tutorials videos may be of assistance:

Discovering Model Builder
www.youtube.com/watch?v=p-H-2a2L1JI

Discovering C++ Code Generation
www.youtube.com/watch?v=oTaUiVTyJeo

Discovering Model Configurations
www.youtube.com/watch?v=LEEibGvIDvc

Tutorials

In the documentation for SystemVue 2012.06 (or later), a more complete step-by-step tutorial is available under "Tutorials/Algorithm Design/C-plus-plus Model Development." Supported users can access this tutorial electronically at http://edocs.soco.keysight.com/display/sv201206/C+Plus+Plus+Model+Development, or within the installed documentation in the SystemVue release.

Documentation

Additional documentation is available for these and other modeling topics:

- Renaming ports/parameters/models
- Adding descriptions to parameters/models
- Adding multi-rate, array and string/file type parameters
- Adding simulation errors, warnings and other informative messages for the custom model
- Fixed-point modeling
- Pre- and post-processing the custom model before and after the simulation (e.g., setup, initialize, finalize methods)

For more information about SystemVue, please visit us on the web:

Product Information
www.keysight.com/find/eesof-systemvue

Request a 30-day Evaluation
www.keysight.com/find/eesof-systemvue-evaluation

Downloads
www.keysight.com/find/eesof-systemvue-latestdownloads

Helpful Videos
www.keysight.com/find/eesof-systemvue-videos

myKeysight

myKeysight

www.keysight.com/find/mykeysight
A personalized view into the information most relevant to you.

Three-Year Warranty

www.keysight.com/find/ThreeYearWarranty
Keysight's commitment to superior product quality and lower total cost of ownership. The only test and measurement company with three-year warranty standard on all instruments, worldwide.

Keysight Assurance Plans

www.keysight.com/find/AssurancePlans
Up to five years of protection and no budgetary surprises to ensure your instruments are operating to specification so you can rely on accurate measurements.

www.keysight.com/quality
Keysight Technologies, Inc.
DEKRA Certified ISO 9001:2008
Quality Management System

Keysight Channel Partners

www.keysight.com/find/channelpartners
Get the best of both worlds: Keysight's measurement expertise and product breadth, combined with channel partner convenience.

www.keysight.com/find/eesof-systemvue

For more information on Keysight Technologies' products, applications or services, please contact your local Keysight office. The complete list is available at:
www.keysight.com/find/contactus

Americas	
Canada	(877) 894 4414
Brazil	55 11 3351 7010
Mexico	001 800 254 2440
United States	(800) 829 4444
Asia Pacific	
Australia	1 800 629 485
China	800 810 0189
Hong Kong	800 938 693
India	1 800 112 929
Japan	0120 (421) 345
Korea	080 769 0800
Malaysia	1 800 888 848
Singapore	1 800 375 8100
Taiwan	0800 047 866
Other AP Countries	(65) 6375 8100
Europe & Middle East	
Austria	0800 001122
Belgium	0800 58580
Finland	0800 523252
France	0805 980333
Germany	0800 6270999
Ireland	1800 832700
Israel	1 809 343051
Italy	800 599100
Luxembourg	+32 800 58580
Netherlands	0800 0233200
Russia	8800 5009286
Spain	0800 000154
Sweden	0200 882255
Switzerland	0800 805353
	Opt. 1 (DE)
	Opt. 2 (FR)
	Opt. 3 (IT)
United Kingdom	0800 0260637

For other unlisted countries:
www.keysight.com/find/contactus
(BP-07-10-14)

KEYSIGHT TECHNOLOGIES