Author:
Nitin S (nitinskandan@gmail.com)
Specialist Engineer
www.tataelxsi.com

SIMULINK BLOCKSET development GUIDE

About this document

World over SIMULINK has established as defacto standard for graphical modeling. One of the main advantage of SIMULINK based development is the capability of create customized blocksets there by allowing reusability of components developed with SIMULINK. As you can see on the SIMULINK library browser there is lot many tool boxes. But how do we create one?

Here I am going to talk about professional blockset development using SIMULINK. This is a whitepaper that compiles the information regarding blockset development.

Block set development is a professional area and it is difficult to newbies to work with. This mainly due to the convergence of various know-how of MATLAB domains that is needed to develop a blockset. Such knowledge can only be build up over a period of time. There have been many organizations and people who have come up with professional blocksets in SIMULINK but so far no one has thought of publishing some kind of tutorial in this topic. So let me do my own bit for helping the users out there. This document does not provide you all the details, but it surely give some hints and directions on which way you can proceed.

The paper is written keeping in mind the following audience categories

1. People who want to develop their own specific toolboxes in SIMULINK
2. People who want to provide exception handling (restrict the usage of their blocks in specific manner) in their blocksets.
3. People who want to provide code generation support for their blocksets.

4. People who want to develop SIMULINK blocks to interface their hardware with SIMULINK.
5. And for all those who want to know more of SIMULINK

1. Introduction

If you want to develop an advanced and professional SIMULINK blockset you should be familiar with following areas / skillsets.

1. Development of C- S functions
2. Development of GUI
3. M-script programming for block building and to access data / configuration from model
4. Writing TLC to support code generation with RTW.

In the file exchange page itself you can easily locate tutorials and example of each of these areas.

A generic SIMULINK blockset consists of the following components.

1. A library (mdl) file with SIMULINK blocks.
2. Block S-function (functionality is done here)
3. Each block will have a GUI for configuring the block (simple GUI with SIMULINK mask / a complex GUI made with GUIDE tool)
4. Block building scripts (m-scripts) to do the automatic configuration of blocks based on user configuration settings.
5. Block TLC file to generate inlined code from the block when you generate code of model with RTW.
6. A database (local) for storing information pertaining to a block and another database (global) for storing information to be accessed by all the blocks.
7. An exception handling mechanism to restrict the usage of blocks in a specific manner (depends on the blockset requirements)
8. Mechanisms for integrating the blockset with MATLAB/SIMULINK.

In preceding sections of this document we shall see each of these components in detail and will discuss points to be considered while working with them.

2. Block functionality implementation

The block functionality could be implemented with

1. Basic SIMULINK blocks
2. With S-function

S-functions are to be used only if the functionality cannot be made with basic SIMULINK blocks. Go for C- S functions due to better execution time performance. There are separate tutorials available on how to develop S-functions so I am not planning to cover that topic here.

However, I would just elaborate some important points to be accounted.

- Whether there will be multiple instances of the same library block in a model? . If the answer is YES, then the S-function should use work vectors for global variables used for information exchanges across mdlXXX functions.
- Whether the S-function is to be inlined? If its to be inlined some S-function parameters whose values are required at the time of code generation has to be written to *.RTW file in the mdlRTW() S – function method. Please note that code generation mechanism parses the *.RTW file created from the model. So any information of the model that is required for code generation must be present in *.RTW file.
- The S-functions should have proper exception handling mechanisms in place to avoid invalid data formats or other situations that can lead to faulty behavior of S-function.
- If Block TLC for inlining needs data exchange across different TLC functions then corresponding work vectors must be created in the S-function itself. As it is not possible to do the same inside TLC. (These are situations where work vectors has to be declared in S-function even though they are not used inside S-function)
- It may be required to have varying number (dynamic) of input ports for S-function. In such cases the S-function input ports should appear before block building inside the library block. The number of input ports could be changed only by executing the S-function. The challenge here is to ensure the S-function is

executed before any block building programs is executed. Some points to take care of this are given in section.

- There may be operations like those which have to be done only once during the model execution such operations are to be done from mdlStart() callback of S-function.
- In some situations the block might have to send data to SIMULINK during simulation and to some real time hardware during real time simulation. In such situations it is possible to have the S-function take care of both cases (without inlining). Use #ifdef MATLAB_MEX_FILE preprocessor directive to distinguish code section for real-time section and simulation section. Ex:

```
#ifdef MATLAB_MEX_FILE
        Write code for PC simulation here
#else
        Write code for real-time simulation here
#endif
```

If you are already planning for S-function inlining then don't include any real-time simulation related code anywhere in the S-function (this will be provided in block TLC)

3. Block GUI development

To configure blocks GUI are needed. GUI should open when user double-clicks a block. GUI can be provided to block by 2 mechanisms

1. SIMULINK GUI
2. GUIDE based M-GUI
3. GUI made in VC, .NET, FLASH

When thinking about the mechanism to be used for a particular block always go for method 1→2→3. If 1 is sufficient to meet your block configuration requirements use that. If that does not work think of 2 and then 3.

3.1 SIMULINK GUI

SIMULINK GUI can be made with Mask editor of SIMULINK block. The data entered by user in the SL GUI is stored as block parameter which is a string variable. This is the easiest way to develop a GUI to configure the block. However only simple GUI's can be made this way.

3.2 M-GUI

GUI could be developed with GUIDE tool of MATLAB. Complex GUI's can be constructed in this fashion (with Multiple tabs, Panels etc).

All M-GUI's to have the following features

1. OK-APPLY-HELP button's and associated support features
2. Handling of KeyPress events (ESC, ENTER) (implemented in KeyPressFcn)
3. Validation of user data entered in GUI fields. If data is not valid it should not be accepted and user must be intimated on this. (Implemented in GUI script)
4. GUI to be locked if opened from Library. (To be implemented in GUI script after checking the library link status of block)
5. GUI should not accept data if model is already running. (Implemented in GUI script after checking Model 'SimulationStatus')
6. A separate and unique GUI instance should open for each block instance or each block instance in the model should have GUI associated. (The block handle stored in block GUI can be used to avoid multiple GUI for same block. "gui_Singleton" can be set to 0 in GUI opening fcn callback if there should be separate instance for each block)
7. GUI of block should close automatically if it is already open and the parent block is deleted. (To be done in delete callback of the block)
8. GUI should close if model is closed (To be done in Model close callback of block)
9. GUI should close if the blocks parent susbsystem is closed (To be done in Parent Close callback of the block)
10. GUI should save the user configuration on closing. (So that user configuration settings is not lost)

11. GUI when it opens must display the last saved user configuration

3.2.1 Integrating M-GUI and SIMULINK block

Integration of GUI involves accomplishing the following functionalities

1. The data in GUI must be stored in the block when the GUI closes, so that it is not lost when GUI closes.
2. The data from block must be used to update the GUI fields when GUI opens so that GUI opens with last saved data.
3. GUI opens when user double clicks a block.
4. Block building should be initiated from GUI (user clicks ok/apply)

3.2.1.1 Data exchange method

The data collected in the GUI must be stored in the block after the validation. Block provides two data elements to store GUI data.

1. Block data parameter (BDP)
2. Block User Data (BUD)

The BDP is a string type variable (limitation of BDP) and is created using mask editor of block. If the GUI is a simple one with limited fields then its better to use BDP.

If GUI is complex the data like a structure it could be stored as in 'UserData' parameters of block. This is the most common method.

Sometimes a combination of BUD and BDP shall be needed. (Reason explained in section)

In the opening function of GUI the content of BUD / BDP is updated to respective GUI fields. Similarly BUD/BDP is updated when user presses APPLY/OK buttons in GUI. It is necessary to set default data for BUD for the block in library to ensure that library block GUI opens with default block configuration.

3.2.1.2 Initiating block building from GUI

In some blocks block building should happen when user presses APPLY/ OK buttons for ex: additional port should appear, port name should change etc.

Block building is essentially done in MASKINIT callback of the block. So the MASKINIT callback section shall be triggered from the GUI. The only way to accomplish this from GUI is to set a BDP with a changed value (if value set is same as

currently stored MASKINIT is not invoked). If block does not have BDP, dummy BDP will be required.

3.2.1.3 Opening of the block

When user double clicks a block, the block open function callback will be executed. Scripts to invoke GUI of the block shall be provided here. The handle of the block shall be passed as argument to the GUI program and it should be stored in GUI (to associate GUI with a block instance as explained in the beginning)

3.2.2 Advanced topics

3.2.2.1 Distribution

The GUIDE based GUI is represented by a XXX.FIG file and XXX.M file. It is possible to merge these two files (using export option) into a common M file. This common m file can in turn be converted to P file and used for distribution of the blockset.

3.2.2.2 GUI blurred during opening

When FIG file and M file is merged to an M file. There can be some issues with visualization of GUI when it opens. The user will be able to see GUI creation when GUI opens. To avoid this the merged m file has to be manually modified so that the GUI is made visible only after GUI building is fully over.

3.3 Other GUI's

It is possible integrate GUI developed with .NET, Adobe flash etc to be integrated with SIMULINK block. This is possible via the activeX interface of these platforms. For details refer to details on interfacing activeX objects with MATLAB.

3.4 Tab based GUI

GUI's with multiple pages and tab-based format has to be created sometimes. These can be made using panels in GUIDE. Each panel will be made visible / invisible based on tab selected. Only one panel will be visible at a time.

4. Exception handling mechanism

Exception handling is a very essential component of blockset. It prevents the user from faulty usage blocks while making SIMULINK model which may in turn lead to unpredictable model behavior. For ex: A block can have only one instance in a model. These kinds of situations are to be handled by exception handler.

Exceptions have to be shown during various events some are mentioned here: -

1. When a block is introduced (paste, move, undo-delete, new block added from library etc)
2. When block initializes
3. When simulation starts

Exception handling programs must do the following operation

1. Identify exception situations
2. Report exceptions to user (warning in MATLAB workspace or pop up dialog)
3. Take action when exception is found. (Stop model initialization)

4.1 Implementing exception checks

Exception checks are implemented in block callbacks and block mask callbacks. The place where a particular check is to be implemented depends on the event that should trigger the exception check. For ex: exception checks to be done during block introduction should be done in block copy callback of the block, exception to be checked during block initialization shall be implemented in block init callback. ***Most of the exception checks are done in BLOCK INIT***.

Please take a detailed look at the callbacks and when it is executed to identify the location of place an exception check.

4.2 Multiple exception messages

All block in the library shall have exception checks implemented. Hence it is possible that multiple exception messages or different instances of same exception message are reported when model simulation starts / when model is initialized as block init callback of all blocks shall be executed during that time. Ideally there exception should be thrown one at time.

A global flag can be used to signal that an exception had occurred. If this flag is already set then no more exception checks would be done anywhere.

But the challenge here is to reset the flag. The flag should be reset when the init callback of last block in the model is executed. But how do you find the last block? SIMUINK does not have any fixed order for calling block callbacks. The only way is to have a count, compare the count with the count variable stored in global data.

4.3 Selective exception checks

Most exception checks need not be done every time, instead it sufficient to do check only when there has been any changes in model. A Global flag can be used to denote the need for exception check. The flag would be set from block callbacks that are likely to detect changes in model (copy, undo delete, delete). The flag could be reset once the exception check is over and no error has been found.

4.4 Caution

When implementing exception check logic do not rely on the callback sequence. SIMULINK does not specify the sequence of callback execution for ex: there is block load callback and block mask init callback it cannot be concluded that block init callback shall be executed before block mask init is executed during all circumstances. So while preparing design for exception checks and block building keep this in mind so that the program is independent of callback sequence.

5. Block building mechanism

Based on the user configuration it will seldom be required to reconfigure the blocks from its default state inside the block library. This may involve

1. Placing new blocks
2. Making changes in existing signal connections or adding new connections.
3. Changing parameters of existing blocks
4. Deleting/replacing existing blocks

Block building needs to be done on various occasions like

1. When model is loaded (opened) all library blocks will have the default settings as in library, even if the user has configured the block and saved model. So its necessary to build the library blocks at the time of model loading so that existing user configurations are preserved.
2. When user clicks OK button in configuration GUI.
3. When user initializes the model.

5.1 Block building method

The block building has to be implemented in MASKINT callback section of respective block. It is possible to do block building from other block callbacks as well, ***but never do it as there are some limitations and is not a good practice.*** The MASKINIT of the block

is executed during several occasions and hence block building will be done frequently. To avoid this, a block parameter flag could be provided in the block. The flag will be set if block building needs to be done and on successful completion of build the flag will be reset.

In some cases where the S-function has dynamic inputs, it may be necessary to execute S-function before block building starts. In such case it is necessary to forcefully execute the S-function before the block building starts. So the script to accomplish this should also be part of block building program. To execute S-function it is necessary to set the dialog parameter of the S-function with the updated parameter value set.

5.2 Block building on model loading

When the model loads the library block will load with the default settings as in the library. So block building needs to be done based on block configuration before the model opens. MASKINIT of all blocks must be executed for this to happen properly. To ensure this, it is necessary to do a find_system call to get the handle of all blocks in the model from model postload function. If this is not performed the signal connection settings with the library block will be lost.

6. Code generation support

When deciding on code generation support the following questions should be asked.

1. Does the block for which code generation support is needed use user defined S-function?
2. If answer to point 1 is YES then, does the application for which the blockset is made demand that blockset's code is to be generated?
3. If answer to point 2 is YES then, Does the blockset has to support code generation with RTW- Embedded coder / has the code generated from block to be optimal?

If answer to questions 1, 2 and 3 are YES then, you have to inline block by providing Block TLC. If 1 and 2 are YES and 3 is NO then no TLC need to be provided. The S-function itself is sufficient for code generation support. Please refer to section.

Various tutorials are available in central file exchange on TLC's, I shall be updating some existing t.

7. Toolbox integration with MATLAB

Integration with MATLAB implies accomplishing the following tasks.

1. User must be able to use toolbox in MATLAB irrespective of the current working path.
2. The toolbox must be accessible from the SIMULINK library browser
3. Toolbox must be accessible from MATLAB start menu
4. Toolbox and its working version installed shall be shown with MATLAB "version" command.

The following files are needed for performing these tasks. Templates of these files are already part of this distribution.

Note:

Name of all files mentioned below should not be changed.

7.1 Startup file

Toolbox operations that are to be automatically executed when MATLAB loads can be included in this m-file. In the template the file in turns calls another file mytoolboxconfig.m. This architecture is preferable.

The file sets path of Toolbox distribution if it is not already in the MATLAB path. This makes sure that toolbox is accessible from MATLAB command prompt.

7.2 Info.xml

This file is responsible for making the toolbox accessible through MATLAB start menu. The template file provided along with the package can be customized for the any toolbox. Each item has a callback section where user can include executable commands (m-script commands) to be invoked when user selects the corresponding menu item. Any image icons to be displayed in start menu can also be included in XML file.

7.3 Contents.m

The first two lines to this template is to be customized to display blockset name and version when user types 'ver' command in MATLAB.

7.4 slblocks.m

This template is to be customized to access the blockset from SIMULINK library browser. Sections of the template that are to be modified are mentioned in the script itself.

7.5 sl_customization.m

This template is to be customized to access the library from the SIMULINK model menu as in figure below.

8. Other software components

8.1 Need for Global data

A model will have large number of library blocks. To do exception checks, block building and other related functionalities it is necessary to get that handle of these blocks. The handle would in turn be used to access other block parameters.

The direct method to get handles is via find_system command. The use of this command can have a negative impact on block usage characteristics (as it will slowdown things). A work around is to the store important information (or frequently accessed information) of each block in Global structure variable in MATLAB workspace. The structure has to be maintained and updated during the coarse of model updation / building.

8.2 Need for error handler

Error handler is a unified mechanism to route all exceptions thrown by the library. Such a mechanism can impart modularity to the software. The error handler does the following

1. The expection handler passes on the following information to error handler
    a. Error message
    b. Type of error – this determines the type of response
    c. Block details
2. Based on these information error handler takes the appropriate action to alert the user and decides on the subsequent action (program termination, warning and program continues)

9. Distribution of toolbox

This section we will see how the library can be distributed across teams or made part of installation package.

The primary task here is to make a folder structure.

LOCAL – refers to local folder of MATLAB that is located at $://MATLAB ROOT / Toolbox/ Local
Startup.m file should be kept here.

MYTOOLBOX sub folder - contains all executable M file, P file, MDL file needed for the toolbox to work. This is the main folder where most of toolbox files are located.

RTWR- this folder contains the TLC files needed for code generation support of the blocks, S-functions which implement the functionality of block, C files & Library files which are needed for working of block S-function and block code generation mechanism.

9.1 Source code hiding

In order to make sure that other people don't tamper with the toolbox source code it is necessary to restrict the modification of the same.

M-files – are to be distributed only after converting them to P-files using command P-code. If the toolbox is intended to work with multiple MATLAB versions then, p code conversion shall be done in the lowest (oldest) MATLAB version supported.

GUI files – are to be merged (fig, M file to be merged) to a single M-file and this can in turn be converted to P file.

Note:

Since the whole blockset is on MATLAB path all MATLAB executable files in the toolbox folders will be accessible from MATLAB command prompt. There may be situation where you may to make a few files as private i.e. these files are need by our toolbox programs but should not be accessible to user directly.

The way to do this is to make a folder named “Private” in the root folder. For ex: If a “private” folder is made inside “MYTOOLBOX” all programs residing in MYTOOLBOX can access the programs inside “private” folder but programs in RTWR folder cannot access them.