Module 10 :: INSEL in MATLAB and Simulink

Sorry, but this Module is not yet complete as INSEL 8.2 is released (April 2014). Two things are important to know, hoewever.

First, if you intend to use the INSEL Renewable Energy Blockset in Simulink, copy the file `startup.m` from `resources\inselSimulink` to your MATLAB installation's directory `toolbox\local` or append its content if you should have a `startup.m` file there already.

Second, it is recommended to have a look at the examples in the `inselSimulink\examples\blocks` directory. Since some of the examples use relative paths to files make this directory the current directory in MATLAB's commnd window before you start.

Enjoy!

10.1 What is MATLAB?

MATLAB is a high-level computer language, developed by the company The Mathworks, Inc. during the 1980's. The acronym MATLAB stands for matrix laboratory. With its interactive environment the software can be used for algorithm development, data visualisation, data analysis, and for numeric computation. It contains mathematical, statistical, and engineering functions, like

- Matrix manipulation and linear algebra
- Polynomials and interpolation
- Fourier analysis and filtering
- Data analysis and statistics
- Optimisation and numerical integration
- Ordinary differential equations (ODEs)
- Partial differential equations (PDEs)
- Sparse matrix operations

Additional toolboxes provide specialised mathematical computing functions for areas including signal processing, optimisation, statistics, symbolic math, partial differential equation solving, and curve fitting.

When the program is started the MATLAB default desktop opens, as shown in Figure 10.1.

Below a standard menu and toolbar four windows are shown:

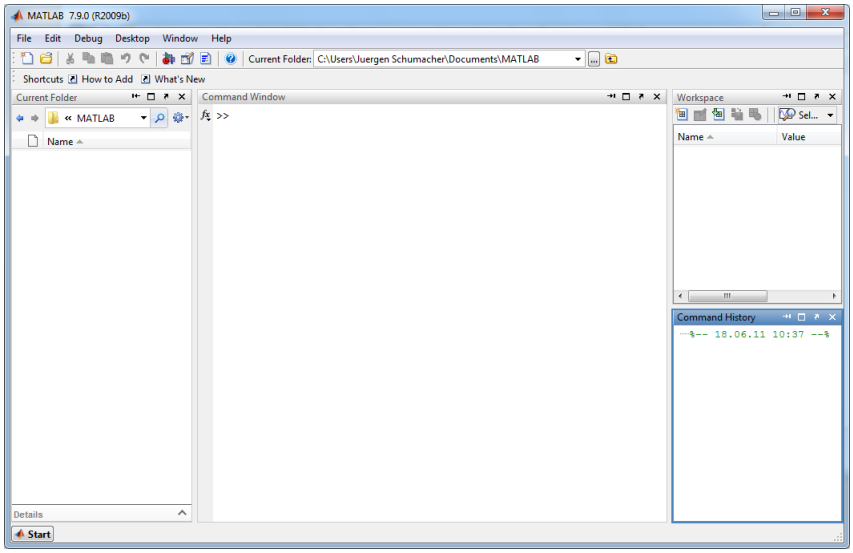


Figure 10.1: Default MATLAB desktop layout.

- The Current Folder window at the left side displays the content of the assigned current folder. The current folder can be changed anytime from the pull-down menu in the toolbar.
- The window displayed in the center is the Command Window. It shows a prompt `>>` where MATLAB commands and functions can be executed interactively.
- The Workspace and Command History windows are displayed at the right side of the MATLAB desktop.

Command window

There are several ways how to communicate with MATLAB. The most direct access is to type in commands at the command promt. For example, typing `magic(4)` makes MATLAB answer with

```
>> magic(4)
ans =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

If we wish to know how `magic` works, we can ask MATLAB for help:

```
>> help magic
 MAGIC  Magic square.
```

```
MAGIC(N) is an N-by-N matrix constructed from the integers
1 through N^2 with equal row, column, and diagonal sums.
Produces valid magic squares for all N > 0 except N = 2.

Reference page in Help browser
   doc magic
```

The link `doc magic` leads directly to the online help browser for more information.

We could ask MATLAB to calculate the sums of the four colums:

```
>> sum(magic(4))
ans =
    34    34    34    34
```

M-files

Beside the interactive computational environment MATLAB provides a powerful programming language. Files that contain code in the MATLAB language are called M-files. Two kinds of M-files can be written:

- Scripts operate on data in the workspace. They do not accept input arguments, nor do they return output arguments.
- Function do not access data in the workspace but internal variables. Data exchange with the workspace is possible through input arguments and through return output arguments.

hello.m

A simple script which displays the hello-world string has only one line of code:

```
'Hello World!'
```

When the script is saved in a file named `hello.m` to the current folder, it can be executed by just typing the name of the script at the command prompt:

```
>> hello
ans =
Hello World!
```

sum12N.m

A simple function which sums up integers from one to a variable n and returns the result in a variable named y is

```
function y = sum1toN(n)
y = sum(1:n)
```

The result is

```
>> sum1toN(10);
y =
    55
```

It is possible to write functions for MATLAB in C or Fortran, too. The minimal C and Fortran prototypes are

helloC.c

```
#include "mex.h"
void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[]) {
   mexPrintf("Hello C!\n");
}
```

and

helloF.f

```
      INCLUDE "FINTRF.H"
      SUBROUTINE MEXFUNCTION(NLHS,PLHS,NRHS,PRHS)
      IMPLICIT   NONE
      INTEGER    NLHS,NRHS
      MWPOINTER  PLHS(*),PRHS(*)
      MEXPRINTF('Hello Fortran!')
      RETURN
      END
```

Simulink A graphical MATLAB user interface named Simulink is available to model, simulate, and analyse dynamic systems by building models as block diagrams. Simulink supports linear and nonlinear systems, modeled in continuous time, sampled time, or a combination of both.

The block libraries are fully customisable and blocksets are available for fixed-point modeling, event-based modeling, physical modeling, control system design and analysis, signal processing and communications, code generation, rapid prototyping and hardware-in-the-loop simulation, verification and validation, and simulation graphics and reporting, to mention just the main application fields.

In the context of this manual, it will be described, how blocks, which were originally written for the simulation environment INSEL, are implemented in the Renewable Energy blockset. The process involves two successive steps:

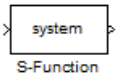
(1) Programming of a universal S-function, adapted to the definition of a general INSEL block.

(2) Programming of Ruby scripts for the automated generation of masked S-function implementations in a Simulink library.

10.2 S-functions in Simulink

System functions or S-functions are computer language descriptions of Simulink blocks. They can be written in M (the MATLAB language), C/C++, or Fortran. Code written in one of the latter two languages must be compiled as MEX-files using the mex utility, which is provided by MATLAB. When needed, these MEX-files are dynamically linked into MATLAB.

S-functions require a special calling syntax so that the code can interact with Simulink's equation solvers. The form of S-functions is very general and can accomodate continuous, discrete, and hybrid systems.



Once written and compiled, an S-function can be incorporated into a Simulink model. Simlink provides an S-function block. It can be found in the User-Defined Functions block library. Once dragged to the drawing area, a double-click opens the S-function dialog box as shown in Figure 10.2

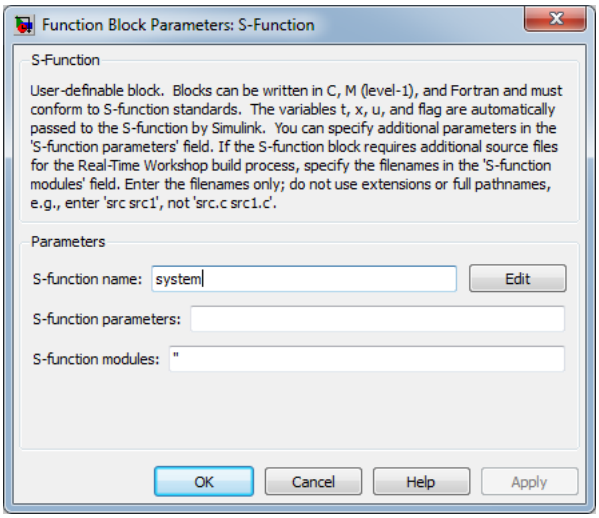


Figure 10.2: S-function dialog box.

The name of the S-function can be specified in the S-function name parameter. The Simulink default name is `system`, the INSEL block S-function is named `SinselBlock`. Parameters from the S-function parameters parameter will be passed directly to the S-function. The S-function parameters can be MATLAB expressions or variables separated by commas. The third S-function modules parameter applies only in the context of the Real-Time-Workshop software, which is of no interest here.

If we save a file which just contains the default S-function block, Simulink writes a lot of ASCII data to a file with extension mdl. Beside plenty of overhead the S-function description is similar to:

Empty S-function

We can identify some interesting keywords: The S-function is implemented as a “Block” with attributes like BlockType (S-Function), its FunctionName (system) etc. Under “System” we see some more keywords which deal with the location of the block in the Simulink file, color definitions etc and finally, the implementation of the “Block”, being of “Blocktype” S-Function named “S-Function”, having an SID 1 and one input and one output port.

Mask editor

Since we wish to implement INSEL blocks similar to their representation in VSEit we now look at some possibilities to improve the appearance of S-functions in Simulink. Via a right-click on the S-function the Mask Editor presented in Figure 10.2 can be opened.

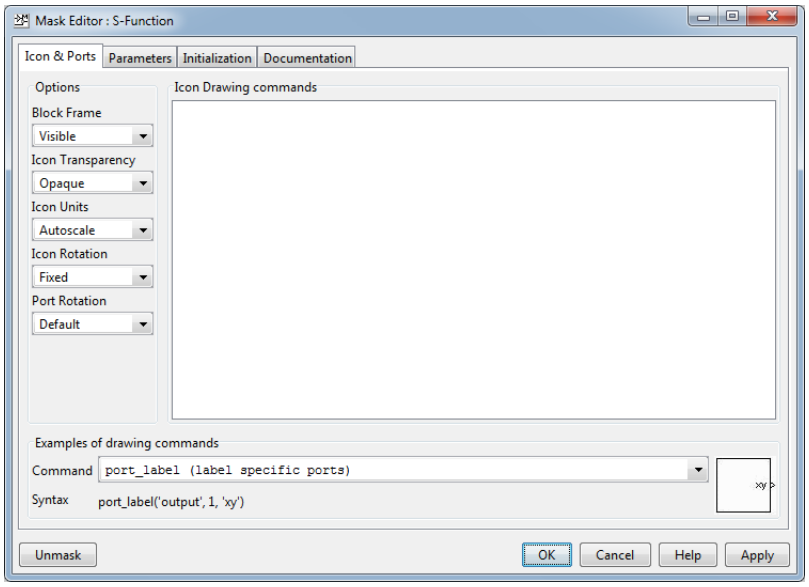


Figure 10.3: S-function mask editor.

Individual interfaces can be defined for each S-function via four tabbed panes of the mask editor. The Icon & Ports pane enables the definition of a block icon, via the Parameter pane mask dialog box parameter prompts and variable names for the individual parameters can be defined. It is possible to define initialization commands for dialog variables of the S-function via the Initialization pane and to provide some documentation of the S-function via the Documentation pane.

In the Renewable Energy blockset, each INSEL block will get an own icon – exactly the same icon as it appears in INSEL itself. The syntax is

```
image(imread('geng.png','png','BackgroundColor', [1 1 1]))
```

The mask drawing command `image` is used to display an image on the icon of the masked S-function. The MATLAB function `imread` reads an image from a graphics file, `geng.png` in our example. The problem how Simulink finds the path to the icon files will be discussed later (page 194). The string `'png'` specifies the format of the graphics file by its standard file extension. MATLAB supports different file formats, we restrict ourselves to portable network graphics. Finally,

the transparency of the icon's pixels can be defined through the BackgroundColor parameter by a three-element vector whose values must be in the range between zero and one.

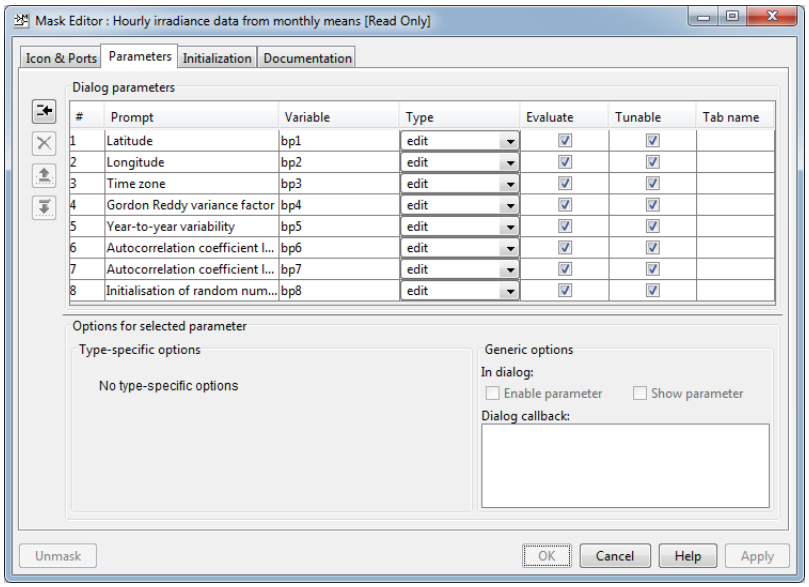


Figure 10.4: Parameter definition in the S-function mask editor. The example is taken from the INSEL block library and represents the GENG block, which can be used to generate meteorological data of solar irradiance in hourly resolution.

Figure 10.2 shows an example for the parameter definition of an S-function. The text specified in the Prompt column will be displayed in the mask dialog box. The variable names follow the convention bp$n$, indicating that they are "block parameters" (a naming convention in INSEL for numerical parameters), numbered from 1 to the total number of bp's. It is also possible to have "string parameters" named `sp`n, accordingly.

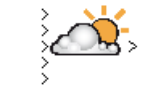
Finally, in the Documentation pane three different strings can be specified, a Mask type a Mask description and a Mask help string. The mask type string ("Hourly irradiance data from monthly means", for example) will be displayed in the mask's margin. The mask description ("The GENG block generates a series of hourly global radiation data from monthly mean values.", for example) will be displayed at the top of the mask.

The mask help string can contain just a literal string or HTML text, and it is possible to specify commands which enable the link to a URL passed to the default web browser by Simulink. Another option is offered through the `eval`

command, which is then passed to MATLAB and evaluated. In INSEL the documentation is completely based on PDF files. An executable named `inselHelp` accepts an INSEL block name as parameter and opens the block reference at the corresponding page. Hence, all INSEL-related S-function masks use the string

```
eval('!inselHelp BN'))
```

inselHelp moechte auch gefunden werden (Windows Path)! Dummerweise fuehrt MATLAB zwar einen eigenen search path, ueberlaesst das finden von executables dann aber doch offenbar Windows selbst.



where `BN` stands for the individual block name, GENG, for example. The exclamation point preceeding the executable name is a MATLAB convention which initiates a shell escape function so that the command is directly performed by the operating system. Figure 10.2 shows the open S-function mask for the INSEL block GENG with the concrete implementation as described above.

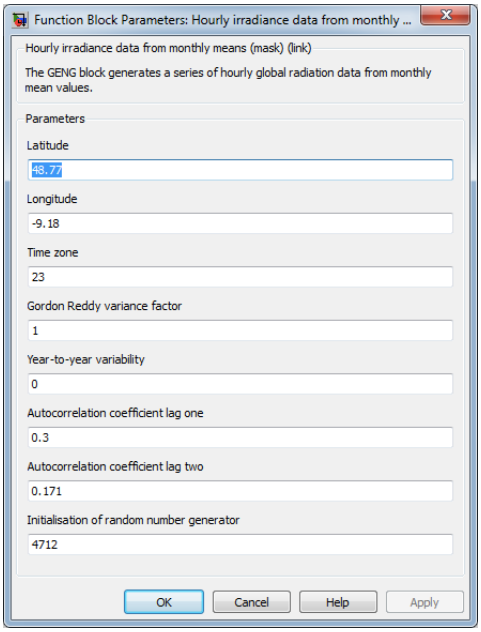


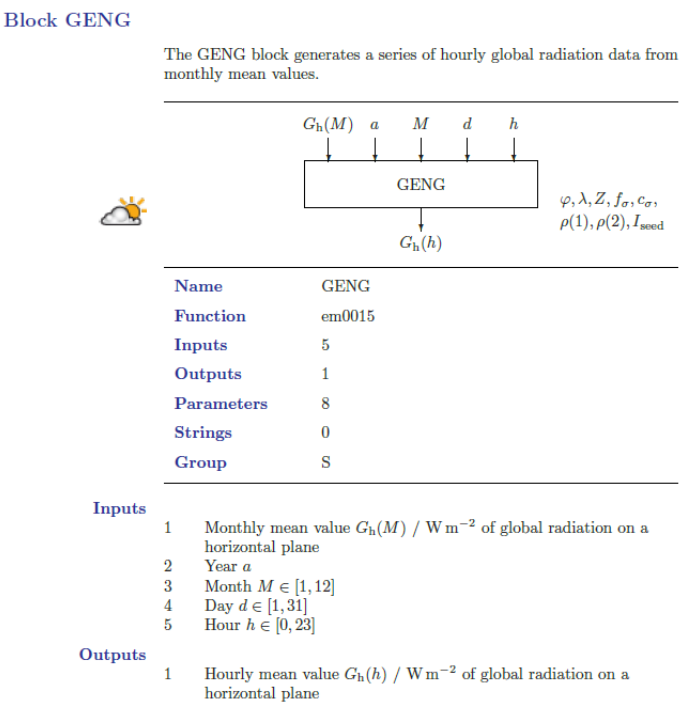
Figure 10.5: S-function mask of the INSEL block GENG.

GENG reference

GENG S-function

Please notice that we have skipped the description of the initialization of the parameters. soll das noch nachgeholt werden?

As mentioned before, a click on the Help button opens the INSEL block reference manual page as shown in Figure 10.2



Block GENG

The GENG block generates a series of hourly global radiation data from monthly mean values.

Name	GENG
Function	em0015
Inputs	5
Outputs	1
Parameters	8
Strings	0
Group	S

Inputs

1 Monthly mean value $G_h(M)$ / $W\,m^{-2}$ of global radiation on a horizontal plane
2 Year a
3 Month $M \in [1, 12]$
4 Day $d \in [1, 31]$
5 Hour $h \in [0, 23]$

Outputs

1 Hourly mean value $G_h(h)$ / $W\,m^{-2}$ of global radiation on a horizontal plane

Figure 10.6: Block reference manual page of the INSEL block GENG (extract).

So, this feels quite like INSEL already. Let us now look at the implementation of the S-function itself.

10.3 The S-function SinselBlock

A look under the mask of the GENG block implementation shows the use of the SinselBlock S-function – see Figure 10.3.

The parameters of the S-function SinselBlock fix the number of block inputs (five), block outputs (one), the name of the INSEL block (GENG) and the parameter list named bp1 ... bp8, as mentioned above.

10.4 Getting Started

10.4.1 Installer

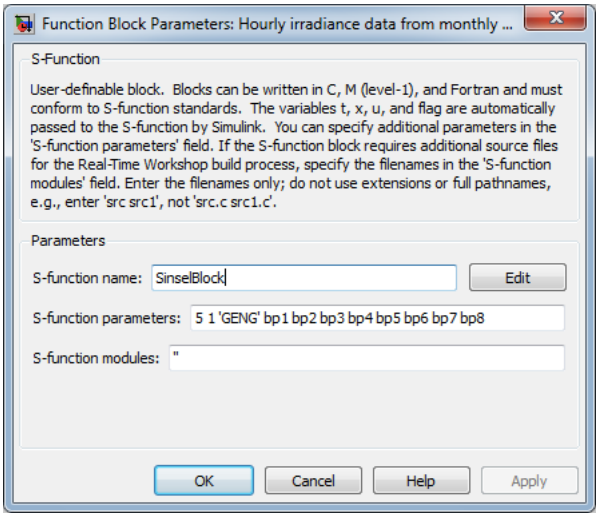


Figure 10.7: S-function dialog box for the GENG block implementation.

We want to integrate the INSEL Renewable Energy blockset with the Simulink Library Browser in such a way that users are allowed to access the blockset in the same way as they access MathWorks products. Therefore, we should

(1) Use the `addpath` command as described in Using MATLAB: Development Environment: Search Path of the Help Browser

(2) Create a `Contents.m` file so that MATLAB displays information about INSEL when `help INSEL` is entered at the command prompt and that it is listed in the response to `ver`.

(3) Create an `slblocks.m` file to define how the blockset should appear in the Simulink library browser.

Ad (1) MATLAB does not use the Windows environment variable `%PATH%` to find files but a special concept, named search path. The search path is a subset of all the folders in the file system. MATLAB can access all files in the folders on the search path.

It is not possible to specify file names relative to a directory in the search path, i. e. if `matlabroot/mydir` is in the search path and `sub` is a subdirectory of `mydir` then files located in `sub` cannot be addressed via `sub/etc`.

MATLAB provides several mechanisms so that users can modify the search path. Most of them are available in the MATLAB command window, but not available programmatically. This means if we wish to inform MATLAB about a new INSEL installation the installer can write a file named `startup.m` with information about

new search path directories. Here comes an example which fulfills the needs of INSEL:

startup.m

```
path('C:\Program Files\insel 8\resources',path)
path('C:\Program Files\insel 8\resources\icons',path)
path('C:\Program Files\insel 8\resources\simulink',path)
```

One possibility to place it in MATLAB's search path is to copy the file to `matlabroot/toolbox/local`. It is however unclear, whether this is the best solution. When MATLAB is replaced by a new installation, the file will be lost and MATLAB and Simulink can no longer access the INSEL blockset.

Suchreihenfolge: 1. matlab search path, 2. in toolbox/local

pfad zu icons in createSinselBlocks auf angepasst (werden jetzt in resources/icons gefunden)

`C:\Program Files\insel 8\resources` muss im Pfad stehen, damit inselHelp.exe, SinselBlock mex32 etc gefunden wird. Alternativ windows/system???

blockDoc.dat – wo soll das ding liegen und wie gefunden werden? Antwort: im insel installationsverzeichnis unter resources. Gefunden wird die Datei von bn2fn mittels der inselroot Funktion, die in inselTools.dll liegt.

Ad (2) Write `Contents.m` in `C:\Program Files\insel 8\resources` with content

```
% INSEL
% Version 8.2 05-Aug-2013
```

When `ver` is typed in MATLAB's Command Window it displays

```
-------------------------------------------------------------------------------
MATLAB Version 7.9.0.529 (R2009b)
MATLAB License Number: XXXXXX
Operating System: Microsoft Windows Vista Version 6.1 (Build 7600)
Java VM Version: Java 1.6.0_12-b04 (...) Java HotSpot(TM) Client VM mixed mode
-------------------------------------------------------------------------------
MATLAB                                                Version 7.9        (R2009b)
Simulink                                              Version 7.4        (R2009b)
INSEL                                                 Version 8.2
```

or something similar.

Ad (3) Write `slblocks.m` in `C:\Program Files\insel 8\resources` with content

```
function blkStruct = slblocks
%SLBLOCKS Defines the block library for a specific Toolbox or Blockset.
%   SLBLOCKS returns information about a Blockset to Simulink.  The
%   information returned is in the form of a BlocksetStruct with the
%   following fields:
```

```
%
%     Name         Name of the Blockset in the Simulink block library
%                  Blocksets & Toolboxes subsystem.
%     OpenFcn      MATLAB expression (function) to call when you
%                  double-click on the block in the Blocksets & Toolboxes
%                  subsystem.
%     MaskDisplay  Optional field that specifies the Mask Display commands
%                  to use for the block in the Blocksets & Toolboxes
%                  subsystem.
%     Browser      Array of Simulink Library Browser structures, described
%                  below.
%
%   The Simulink Library Browser needs to know which libraries in your
%   Blockset it should show, and what names to give them.  To provide
%   this information, define an array of Browser data structures with one
%   array element for each library to display in the Simulink Library
%   Browser.  Each array element has two fields:
%
%     Library      File name of the library (mdl-file) to include in the
%                  Library Browser.
%     Name         Name displayed for the library in the Library Browser
%                  window.  Note that the Name is not required to be the
%                  same as the mdl-file name.
%
%   Example:
%
%      %
%      % Define the BlocksetStruct for the Simulink block libraries
%      % Only simulink_extras shows up in Blocksets & Toolboxes
%      %
%      blkStruct.Name        = ['Simulink' sprintf('\n' Extras];
%      blkStruct.OpenFcn     = simulink_extras;
%      blkStruct.MaskDisplay = disp('Simulink\nExtras');
%
%      %
%      % Both simulink3 and simulink_extras show up in the Library Browser.
%      %
%      blkStruct.Browser(1).Library = 'simulink3';
%      blkStruct.Browser(1).Name    = 'Simulink';
%      blkStruct.Browser(2).Library = 'simulink_extras';
%      blkStruct.Browser(2).Name    = 'Simulink Extras';
%
%   See also FINDBLIB, LIBBROWSE.

%   Copyright 1990-2001 The MathWorks, Inc.
%   $Revision: 1.17 $

%
% Name of the subsystem which will show up in the Simulink Blocksets
% and Toolboxes subsystem.
%
blkStruct.Name = ['Simulink' sprintf('\n') 'Extras'];

%
```

```
% The function that will be called when the user double-clicks on
% this icon.
%
blkStruct.OpenFcn = 'simulink_extras';

%
% The argument to be set as the Mask Display for the subsystem.  You
% may comment this line out if no specific mask is desired.
% Example:  blkStruct.MaskDisplay = 'plot([0:2*pi],sin([0:2*pi]));';
% No display for Simulink Extras.
%
blkStruct.MaskDisplay = '';

%
% Define the Browser structure array, the first element contains the
% information for the Simulink block library and the second for the
% Simulink Extras block library.
%
Browser(1).Library = 'INSEL';
Browser(1).Name    = 'INSEL Renewable Energy';
Browser(1).IsFlat  = 0;% Is this library "flat" (i.e. no subsystems)?

blkStruct.Browser = Browser;

% End of slblocks
```

10.4.2 Link vs. simple copy

Breaking a link to a library block

(verbatim copy of Simulink documentation): You can break the link between a reference block and its library block to cause the reference block to become a simple copy of the library block, unlinked to the library block. Changes to the library block no longer affect the block. Breaking links to library blocks may enable you to transport a Masked Subsystem Example model as a standalone model, without the libraries.

To break the link between a reference block and its library block, first disable the link. Then select the block and choose Break Link from the Link Options menu. You can also break the link between a reference block and its library block from the command line by changing the value of the LinkStatus parameter to 'none' using this command:

```
set_param('refblock', 'LinkStatus', 'none')
```

You can also break links to library blocks when saving the model, by supplying arguments to the `save_system` command. See `save_system` in the Simulink reference documentation.

Breaking library links in a model does not guarantee that you can run the model standalone, especially if the model includes blocks from third-party libraries or optional Simulink blocksets. It is possible that a library block invokes functions

supplied with the library and hence can run only if the library is installed on the system running the model. Further, breaking a link can cause a model to fail when you install a new version of the library on a system. For example, suppose a block invokes a function that is supplied with the library. Now suppose that a new version of the library eliminates the function. Running a model with an unlinked copy of the block results in invocation of a now nonexistent function, causing the simulation to fail. To avoid such problems, you should generally avoid breaking links to third-party libraries and optional Simulink blocksets.

Fixing unresolved library links

If Simulink is unable to find either the library block or the source library on your MATLAB path when it attempts to update the reference block, the link becomes unresolved. Simulink issues an error message and displays these blocks using red dashed lines. The error message is

Failed to find block “source-block-name” in library “source-library-name” referenced by block “reference-block-path”.

The unresolved reference block appears like this (colored red).

To fix a bad link, you must do one of the following:

- Delete the unlinked reference block and copy the library block back into your model.
- Add the directory that contains the required library to the MATLAB path and select Update Diagram from the Edit menu.
- Double-click the unlinked reference block to open its dialog box (see the Bad Link block reference page). On the dialog box that appears, correct the pathname in the Source block field and click OK.

10.4.3 Enumerations and operation modes

I don’t know how often I have thought about shooting the guys who had the idea, that counting indexes should start at zero instead of one. I have never seen a child starting to learn to count fingers with a closed fist representing zero, but showing the thumb (okay – the Japanese start counting with their pinkie). Everybody – except those C guys - wants to have the first item as one and not as zero.

As a very early idea, INSEL provided the option to have similar designed blocks organised in one subroutine and distinguish them by the operation mode parameter – of course, starting with one for the first operation mode, two for the second, and so on. A similar case occurs with a parameter, which enumerates diverse options, like option one, two, and so on. So parameter definitions of INSEL blocks with enumeration-type parameters started with one, followed by two, and so forth.

Then in the mid-90's HP VEE came across INSEL, providing pull-down objects to nicely specify enum objects in a graphical environment. So, the "Default" case was invented in INSEL 5, introducing some "artificial" meaning of the enum-value zero and leaving the logics of enum-parameter interpretation as it was in INSEL before.

Then came VSEit, the great graphical Java interface for insel 8. Since VSEit uses the convention to index enum objects from zero to n, it was decided to follow the standard-C convention and – for God's sake – start to count enum objects at ozero.

Finally, in 2011 we started to deploy INSEL blocks with MATLAB & Simulink. The one-vs.-zero horror returned, when we found out that Simulink indicates enum objects from one to n. Well ...

Since some INSEL blocks use enum-object parameters – and since we didn't want to waste an additional IP parameter on this, we decided to incorporate the "zero-vs.-one" difference in "overloading" the third operation-mode paramter IP(3) – or IP[2], for the start-at-zero fans. Hence, when the operation mode is positive, any enum parameters are interpreted between one and n. If the operation mode is negative the enum parameters are interpreted to start at zero.

Sorry for the mess.