FAQ 298

Best Practice for Handling S-Functions, Libraries and Additional Source Code

Keywords

additional paths; S-Function source files; user makefile; USR.MK file; rtwmakecfg.m; Libraries; user makefile; USR-MK File; Custom Code Page; Simulink Coder®; Real-Time Workshop®

Question

I have organized my S-Function MEX DLL files, source files and header files in separate folders. When using an S-Function in a model I would like to automatically add the respective source paths to the build process. How can I do that?

My custom code references object files or libraries. How can I use these libraries?

Relevant Products

Real-Time Interface (RTI) and ConfigurationDesk, differences between Real-Time Interface (RTI) and ConfigurationDesk are explicitly pointed to

Solution

The standard “Implicit Build Support” for S-Functions takes the following rules into account (excerpt from the Simulink Coder documentation, chapter *Build Support for S-Functions*)**:**

> *Implicit Build Support*
>
> *When building models with S-Functions, the code generator automatically adds rules, include paths, and source filenames to the generated makefile. For this to occur, the source files (.h, .c, and .cpp) for the S-Function must be in the same folder as the S-Function MEX-file. The code generator propagates this information through the token expansion mechanism of converting a template makefile (TMF) to a makefile. The propagation requires the TMF to support the tokens.*
>
> *Details of the implicit build support follow:*
>
> - *If the file sfcnname.h exists in the same folder as the S-Function MEX-file (for example, sfcnname.mexext), the folder is added to the include path*
> - *If the file sfcnname.c or sfcnname.cpp exists in the same folder as the S-Function MEX-file, the Simulink Coder code generator adds a makefile rule for compiling files from that folder.*
> - *When an S-Function is not inlined with a TLC file, the Simulink Coder code generator must compile the S-Function's source file. To determine the name of the source file to add to the list of files to compile, the code generator searches for sfcnname.cpp on the MATLAB path.*
>   *If the source file is found, the code generator adds the source filename to the makefile. If sfcnname.cpp is not found on the path, the code generator adds the filename sfcnname.c to the makefile, whether or not it is on the MATLAB path.*

According to the Simulink Coder’s (formerly Real-Time Workshop) standard handling the following ► use scenarios have been worked out. The order of the presented methods reflects the best practice dSPACE considers to be helpful in a specific use scenario.

It is assumed that the S-Function is named `sfun` and the related MEX-DLL `sfun.mexw32` has been generated with the MEX-Compiler from the S-Function source file `sfun.c`. If your S-Function has a different name, adapt the instructions accordingly.
As a second precondition the MEX-DLL `sfun.mexw32` has to persist on the MATLAB search path.

► Use scenario 1: C-coded S-Function (non-inlined S-Function) `sfun.c` and the MEX-DLL `sfun.mexw32` in the SAME directory

The search path is automatically expanded.

No additional actions are necessary.

► Use scenario 2: C-coded S-Function (non-inlined S-Function) `sfun.c` and the MEX-DLL `sfun.mexw32` in DIFFERENT directories

The path of the S-Function source file `sfun.c` has to be specified either

1. at `makeInfo.sourcePath` in **the rtwmakecfg.m file**.
   Please find further information at **Example of a rtwmakecfg.m file**, or

2. .at “Include list of additional…Include directories” (space separated list) on the **Code Generation (formerly Real-Time Workshop) – Custom Code** pane of the Configuration Parameters dialog, or

3. at “Search paths” (semicolon separated list) of the Custom Code Options of the ConfigurationDesk application’s **Build Configuration Set** for the application process the specific behavior model is assigned to (only ConfigurationDesk, since dSPACE Release 2013-A), or

4. at the macro SFCN_DIR (space separated list) in the model’s **user makefile template** <modelname>_usr.mk (only RTI).

   For example

```
…
# Directories where S-Function C source files are stored.
SFCN_DIR = "\project one\sfcns"
…
```

► Use scenario 3: additional source files

The names of additional source files have to be specified either

1. at `makeInfo.sources` in the **rtwmakecfg.m file**.
   Please find further information at **Example of a rtwmakecfg.m file**, or

2. at the **S-Function block dialog** property S-Function modules, if the additional source files are necessary for an specific S-Function block, or

3. at “Include list of additional…Source files” (space separated list) on the **Code Generation (formerly Real-Time Workshop) – Custom Code** pane of the Configuration Parameters dialog, or

4. at “Custom source files“(semicolon separated list) of the Custom Code Options of the ConfigurationDesk application’s **Build Configuration Set** for the application process the specific behavior model is assigned to (only ConfigurationDesk, since dSPACE Release 2013-A), or

5. at the macro `USER_SRCS` (space separated list) and the paths at the macro `USER_SRCS_DIR` (space separated list) in the model’s **user makefile template** `<modelname>_usr.mk` (only RTI).

   For example

```
…
# Additional C source files to be compiled (file name extension .c).
USER_SRCS = file1.c file2.c file3.c
…
# Directories where additional C and assembler source files are
stored.
USER_SRCS_DIR = "\project one\sfcns" "\project two\sfcns"
…
```

If the additional source files are not located in the same directory as the S-Function source file `sfun.c`, the path has to be expanded as described in use scenario 2 (except point 5.).

	If the Custom Code pane defines source files (option “Include list of additional… Source files”), which do not exist in the folders defined by the “Include list of additional… Include directories” option, the path will not be expanded accordingly. To circumvent this, the name of the S-Function (sfun.c) can be specified as an additional source file.

► Use scenario 4: additional precompiled libraries and object files

Precompiled libraries (object libraries) are hardware platform-specific. To reference a library, it must have been created by the related archiving tool of the compiler that corresponds to the dSPACE hardware platform you are using.

If dSPACE RTLib function calls or function calls generated from the Simulink Coder® (formerly Real-Time Workshop) are part of the object libraries, it is mandatory to use identical MATLAB® and dSPACE Releases. Otherwise compatibility problems may arise.

Furthermore if the libraries integrate Simulink Coder generated function calls, the libraries must correspond to the timer task mode of the model. (*FAQ 012*).

The name and the path of the library files have to be specified either

1. at cell array `makeInfo.linkLibsObjs` in the **rtwmakecfg.m file**.
   Please find further information at **Example of a rtwmakecfg.m file**
   For example

   ```
   …
   makeInfo.linkLibsObjs = {fullfile(pwd, 'libdir',' user_library.lib')};
   …
   ```

   or

2. at “Include list of additional…Libraries” (space separated list) on the **Code Generation (formerly Real-Time Workshop) – Custom Code** pane of the Configuration Parameters dialog, or

3. at “Custom libraries” (semicolon separated list) of the Custom Code Options of the ConfigurationDesk application’s **Build Configuration Set** for the application process the specific behavior model is assigned to (only ConfigurationDesk, since dSPACE Release 2013-A)

   For example,
   `libcustom1.a; .\libs\libcustom2.a,` or

4. at macro `USER_LIBS` (space separated list) for libraries and
   at macro `USER_OBJS` (space separated list) for object files
   in the model’s **user makefile template** `<modelname>_usr.mk` (only RTI).
   For example

   ```
   …
   # Additional user libraries to be linked.
   USER_LIBS = user_library.lib
   …
   ```

	If you want the model to be compatible to both the single timer task mode and the multiple timer task mode, you must include the correct libraries for each timer task mode. You can do this via the `$(OBJ_EXT)` expression, which automatically expands into `.oXX or .mXX` depending on the current timer task mode of the model. The user makefile entry therefore looks like this: `USER_LIBS = user_library_$(OBJ_EXT).lib`

Additional information regarding the S-Functions based on the S-Function Builder Block

The mentioned methods also apply for the S-Function Builder Block. While building the S-Function, additional source files are added to the S-Function Block Properties (Property S-Function modules). Depending on the used properties of the S-Function Builder block, an rtwmakecfg.m file is generated automatically.

Conclusive information regarding the model's user makefile

The model's user makefile has been part of the RTI build process for a very long time. It had been used to define paths and names of additional files, which are custom specific and which cannot be defined in the model. The user make file provides additional options, which are not available on the Code Generation (formerly Real-Time Workshop) – Custom Code pane of the Configuration Parameters dialog. For early MATLAB Releases and dSPACE Release combinations the model's user makefile had been the only way to expand search paths and to add additional files to the build process.
But the model's user makefile is a separate file, which needs to be maintained. This is different to the Configuration Parameters, which are part of the model, and obviously a disadvantage.

Example of a rtwmakecfg.m file

1. Create a `rtwmakecfg.m` file in the folder that holds the MEX DLL files. If a model contains an S-Function with the MEX DLL file in this folder, the `rtwmakecfg.m` file provides the build process with the information about what folders contain the S-Function source and header files.

2. Open the `rtwmakecfg.m` file in an editor and create an `rtwmakecfg()` function, which returns a struct with an `includePath` field and a `sourcePath` field. The example shown below can be used as a template.

For further information, refer to *Using the rtwmakecfg.m API to Customize Generated Makefiles* search for the keyword *rtwmakecfg.m* in the Simulink Coder User's Guide (formerly real-Time Workshop User'S Guide) by The MathWorks

Assume that you have the following directory structure:

```
C:\Projects\SFunctions\
rtwmakecfg.m
sfun1.mexw32 (up to MATLAB® R14SP2 sfun1.dll)
sfun2.mexw32 (up to MATLAB® R14SP2 sfun2.dll)

.\Sources\
sfun1.c
sfun2.c

.\Headers\
sfun1.h
sfun2.h
sfun_common.h
```

Then the related `rtwmakecfg.m` file looks like this:

```
function makeInfo = rtwmakecfg()
% RTWMAKECFG Add include and source directories to
% Simulink Coder make files.
% makeInfo = RTWMAKECFG returns a structured array containing
% following fields:

% makeInfo.includePath - cell array containing additional include
% directories. Those directories will be expanded into include
% instructions of Simulink Coder generated make files.

% makeInfo.sourcePath - cell array containing additional source
% directories. Those directories will be expanded into rules of
% Simulink Coder generated make files.

% makeInfo.library - structure containing additional runtime library
% names and module objects. This information will be expanded into
% rules of generated make files.
% ... .library(1).Name - name of runtime library
% ... .library(1).Modules - cell array containing source file names
% for the runtime library
% ... .library(2).Name
% ... .library(2).Modules

% This RTWMAKECFG file must be located in the same directory as the
% related S-Function MEX-DLL(s). If one or more S-Functions of the
% directory are referenced by a model Simulink Coder will evaluate
% RTWMAKECFG to obtain the additional include and source directories.

% To examine more RTWMAKECFG files in your installation issue at the
% MATLAB prompt:
% >> which RTWMAKECFG -all

% Issue a message.
seperatorLine = char(ones(1,70) * '~');
fprintf('\n');
fprintf('%s\n', seperatorLine);
fprintf(' %s\n', which(mfilename));
fprintf(' Adding source and include directories to make process.\n')

% Setting up the return structure with
% - source directories:
% C:\Projects\SFunctions\Sources
makeInfo.sourcePath = { ...
'C:\Projects\SFunctions\Sources' ...
};

% - include directories
% C:\Projects\SFunctions\Headers
makeInfo.includePath = { ...
'C:\Projects\SFunctions\Headers' ...
};

% Display contents.
fprintf(' - additional source directories:\n');
fprintf(' %s\n', makeInfo.sourcePath{:});
fprintf(' - additional include directories:\n');
fprintf(' %s\n', makeInfo.includePath{:});
fprintf('%s\n', seperatorLine);

% [EOF] rtwmakecfg.m
```

Related FAQs

- -

How to Contact dSPACE Support

dSPACE GmbH
Rathenaustraße 26
33102 Paderborn
Germany

++49 5251 1638-941

mailto:support@dspace.de
http://www.dspace.com/support

dSPACE recommends that you use the support request form on the Internet to contact dSPACE Support.
It is available at:

- http://www.dspace.com/go/supportrequest

Software Updates and Patches

dSPACE strongly recommends that you download and install the most recent patches for your current dSPACE installation. Visit http://www.dspace.com/go/patches for software updates and patches.

FAQ

FAQ documents are available at http://www.dspace.com/go/faq.

Important Notice