

Simulink® Coder™

参考



MATLAB® & SIMULINK®

R2022b



如何联系 MathWorks



最新动态: www.mathworks.com
销售和服务: www.mathworks.com/sales_and_services
用户社区: www.mathworks.com/matlabcentral
技术支持: www.mathworks.com/support/contact_us



电话: 010-59827000



迈斯沃克软件 (北京) 有限公司
北京市朝阳区望京东园四区 6 号楼
北望金辉大厦 16 层 1604

Simulink® Coder™ 参考

© COPYRIGHT 2011–2022 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

商标

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

专利

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

修订历史记录

2011 年 4 月	仅限在线版本	版本 8.0 (版本 2011a) 中的新增内容
2011 年 9 月	仅限在线版本	版本 8.1 (版本 2011b) 中的修订内容
2012 年 3 月	仅限在线版本	版本 8.2 (版本 2012a) 中的修订内容
2012 年 9 月	仅限在线版本	版本 8.3 (版本 2012b) 中的修订内容
2013 年 3 月	仅限在线版本	版本 8.4 (版本 2013a) 中的修订内容
2013 年 9 月	仅限在线版本	版本 8.5 (版本 2013b) 中的修订内容
2014 年 3 月	仅限在线版本	版本 8.6 (版本 2014a) 中的修订内容
2014 年 10 月	仅限在线版本	版本 8.7 (版本 2014b) 中的修订内容
2015 年 3 月	仅限在线版本	版本 8.8 (版本 2015a) 中的修订内容
2015 年 9 月	仅限在线版本	版本 8.9 (版本 2015b) 中的修订内容
2015 年 10 月	仅限在线版本	版本 8.8.1 (版本 2015aSP1) 中的再发布内容
2016 年 3 月	仅限在线版本	版本 8.10 (版本 2016a) 中的修订内容
2016 年 9 月	仅限在线版本	版本 8.11 (版本 2016b) 中的修订内容
2017 年 3 月	仅限在线版本	8.12 版 (版本 2017a) 中的修订内容
2017 年 9 月	仅限在线版本	版本 8.13 (版本 2017b) 中的修订内容
2018 年 3 月	仅限在线版本	版本 8.14 (版本 2018a) 中的修订内容
2018 年 9 月	仅限在线版本	9.0 版 (版本 2018b) 中的修订内容
2019 年 3 月	仅限在线版本	版本 9.1 (版本 2019a) 中的修订内容
2019 年 9 月	仅限在线版本	版本 9.2 (版本 2019b) 中的修订内容
2020 年 3 月	仅限在线版本	版本 9.3 (版本 2020a) 中的修订内容
2020 年 9 月	仅限在线版本	版本 9.4 (版本 2020b) 中的修订内容
2021 年 3 月	仅限在线版本	版本 9.5 (版本 2021a) 中的修订内容
2021 年 9 月	仅限在线版本	版本 9.6 (版本 2021b) 中的修订内容
2022 年 3 月	仅限在线版本	版本 9.7 (版本 2022a) 中的修订内容
2022 年 9 月	仅限在线版本	版本 9.8 (版本 2022b) 中的修订内容

查看 Bug 报告以确定并解决问题

Software is inherently complex and is not free of errors. The output of a code generator might contain bugs, some of which are not detected by a compiler. MathWorks reports critical known bugs brought to its attention on its Bug Report system at www.mathworks.com/support/bugreports/. In the search bar, type the phrase "Incorrect Code Generation" to obtain a report of known bugs that produce code that might compile and execute, but still produce wrong answers. To save a search, click Save Search.

The bug reports are an integral part of the documentation for each release. Examine periodically all bug reports for a release, as such reports may identify inconsistencies between the actual behavior of a release you are using and the behavior described in this documentation.

In addition to reviewing bug reports, you should implement a verification and validation strategy to identify potential bugs in your design, code, and tools.

1	Simulink 代码生成限制	
2	App	
3	Simulink Coder 函数	
4	Simulink Coder 模块	
5	仿真目标参数	
	自动调节	5-2
	描述	5-2
	设置	5-2
	依存关系	5-2
	命令行信息	5-2
6	GPU 加速参数	
7	代码生成参数：代码生成	
	模型配置参数：代码生成	7-2

浏览	7-5
描述	7-5
提示	7-5

8	代码生成参数：报告
---	-----------

9	代码生成参数：注释
---	-----------

10	代码生成参数：标识符
----	------------

模型配置参数：代码生成标识符	10-2
----------------------	------

11	代码生成参数：自定义代码
----	--------------

模型配置参数：代码生成自定义代码	11-2
------------------------	------

12	代码生成参数：接口
----	-----------

模型配置参数：代码生成接口	12-2
---------------------	------

共享代码位置	12-7
描述	12-7
设置	12-7
命令行信息	12-7
推荐的设置	12-7

支持: 非有限数	12-9
描述	12-9
设置	12-9
依存关系	12-9
命令行信息	12-9
推荐的设置	12-9

Generate C API for: signals	12-11
描述	12-11

设置	12-11
命令行信息	12-11
推荐的设置	12-11
Generate C API for: parameters	12-12
描述	12-12
设置	12-12
命令行信息	12-12
推荐的设置	12-12
生成用于根级 I/O 的 C API	12-13
描述	12-13
设置	12-13
命令行信息	12-13
推荐的设置	12-13
ASAP2 接口	12-14
描述	12-14
设置	12-14
命令行信息	12-14
推荐的设置	12-14
外部模式	12-15
描述	12-15
设置	12-15
命令行信息	12-15
推荐的设置	12-15
自动调节	12-16
描述	12-16
设置	12-16
依存关系	12-16
命令行信息	12-16
等间距设置的 LUT 对象结构体顺序	12-17
描述	12-17
设置	12-17
命令行信息	12-17
推荐的设置	12-17
动态大小字符串的缓冲区大小（以字节为单位）	12-18
描述	12-18
设置	12-18
命令行信息	12-18
推荐的设置	12-18
数组布局	12-19
描述	12-19
设置	12-19
命令行信息	12-19
推荐的设置	12-19
MAT 文件记录	12-21
描述	12-21
设置	12-21

依存关系	12-21
限制	12-21
命令行信息	12-22
推荐的设置	12-22

13 | 代码生成参数: GPU 代码

计算能力	13-2
描述	13-2
设置	13-2
依存关系	13-2
命令行信息	13-2

14 | Simulink Coder 参数: 高级参数

15 | Simulink 模型的配置参数

16 | 模型顾问检查

17 | 用于创建受保护模型的参数

18 | Simulink Coder 工具

19 | 优化参数

模型配置参数: 代码生成优化	19-2
----------------------	------

Simulink 代码生成限制

App

Simulink Coder

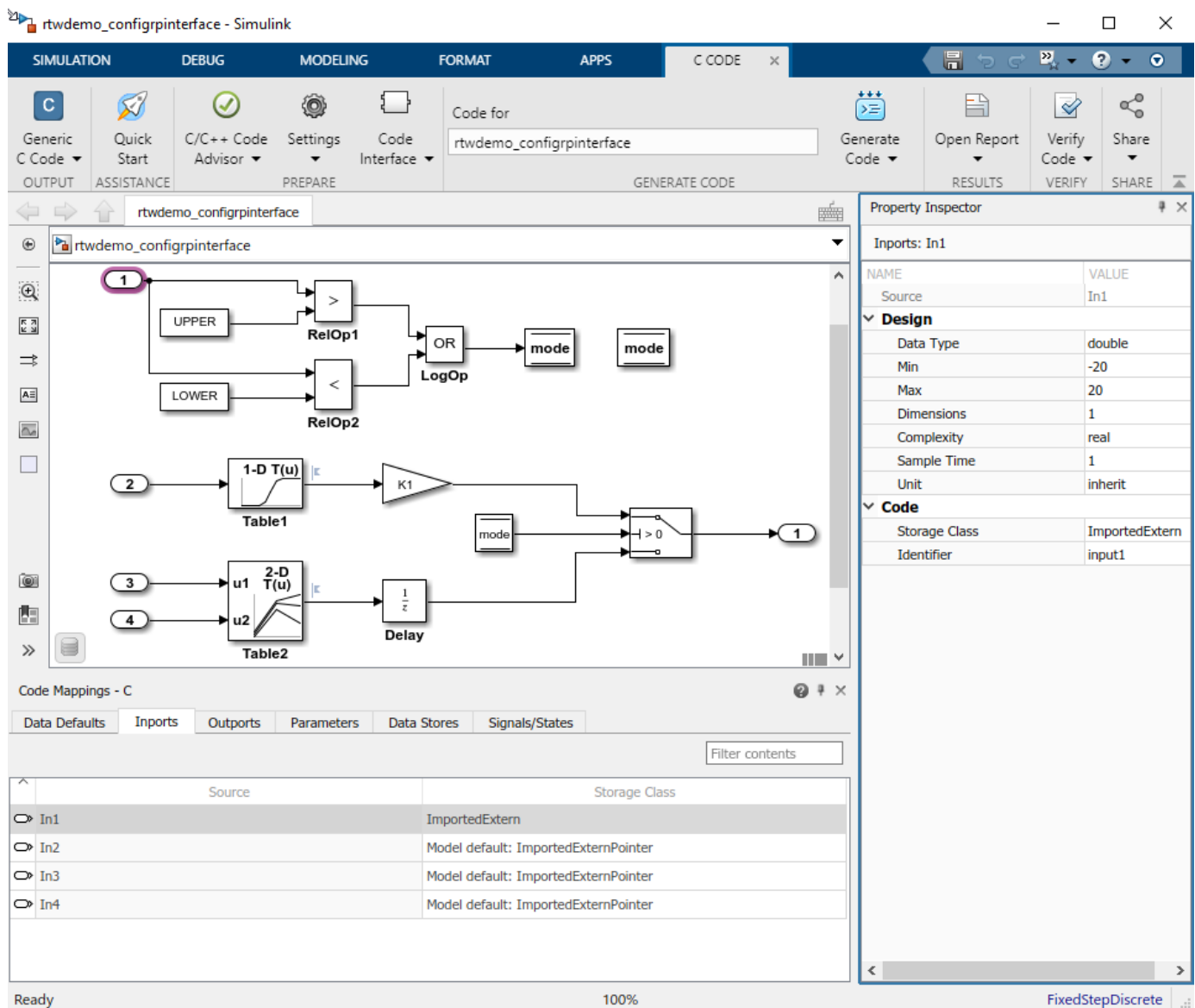
从 Simulink 模型、Stateflow 图和 MATLAB 函数中生成并执行 C 与 C++ 代码，用于仿真加速、快速原型构建和硬件在环 (HIL) 仿真等应用

说明

要从表示离散时间系统的模型中生成 C 或 C++ 代码，请使用 **Simulink Coder**。当您打开该 App 时，会在 Simulink 编辑器工具条中添加 **C 代码** 选项卡。**C 代码** 选项卡包含表示 Simulink Coder 工作流程步骤的各节。通过使用各个编辑器窗口面板，您可以完成所有的模型准备工作，以便进行代码生成、代码生成和编译以及代码验证。

使用该 App 可以：

- 在快速配置模型以进行代码生成的同时学习产品基础知识。如果您不熟悉 Simulink Coder，请使用 Simulink Coder 快速向导为代码生成准备模型。Simulink Coder 快速向导根据您的目标和应用选择基本代码生成设置。要打开 Simulink Coder 快速向导，请点击**快速向导**。
- 设置代码生成目标，并为代码生成准备模型。点击 **C/C++ 代码顾问**。
- 配置模型范围的代码生成参数设置。点击**设置**。
- 配置数据。选择**代码接口 > 默认代码映射** 或 **代码接口 > 个体元素代码映射**，这将打开代码映射编辑器和属性检查器。
- 生成代码或编译可执行程序。选择**编译 > 生成代码** 或 **编译 > 编译**。
- 审核生成的代码。点击**打开报告**。
- 创建一个用于仿真和代码生成的受保护模型，以便与第三方共享。选择**共享 > 生成受保护模型**。
- 打包生成的代码和工件，以便部署到另一个计算机系统。选择**共享 > 生成代码并打包**。



打开 Simulink Coder App

在 **App** 库中的**代码生成**下，点击 **Simulink Coder**。**C 代码**选项卡将打开。

示例

- “Generate Code by Using the Quick Start Tool”
- “使用 Simulink Coder 生成代码”
-

提示

- 如果您正在使用模型层次结构，请找到您要为其生成代码的层次结构的顶层模型，在对应的 Simulink 编辑器窗口中打开 **Simulink Coder**。在 **C 代码** 选项卡上，这些功能适用于在编辑器中打开的层次结构的顶层模型。
- 要为一个引用模型配置数据，请在层次结构中导航到该模型。通过选择**代码接口** > **默认代码映射**或**代码接口个体元素代码映射**，打开代码映射编辑器和属性检查器。这些工具适用于活动模型，该模型可以是顶层模型或引用模型。

版本历史记录

在 R2019b 中推出

主题

“Generate Code by Using the Quick Start Tool”

“使用 Simulink Coder 生成代码”

Simulink Coder 函数

codebuild

编译和链接生成的代码

语法

```
buildResults = codebuild(buildFolder)
codebuild(buildFolder, Name, Value)
codebuild(buildInfo, Name, Value)
```

说明

buildResults = codebuild(buildFolder) 从 **buildFolder** 中的 **buildInfo.mat** 文件加载数据，在 **buildFolder** 中生成联编文件，并使用指定的工具链或模板联编文件来编译在 **RTW.BuildInfo** 对象中注册的源代码。如果对象位于层次结构的顶层，该函数将对层次结构中的每个对象执行该过程。

该函数将编译工件（包括目标代码文件）保存在 **buildFolder** 中。

函数返回包含显示输出的对象。要查看输出，请运行 **disp(buildResults)**。

codebuild(buildFolder, Name, Value) 使用一个或多个名称-值对组指定附加选项。

codebuild(buildInfo, Name, Value) 使用一个或多个名称-值对组编译在 **buildInfo** 中指定的代码。

示例

转移并编译生成的代码

有关如何将生成的代码转移到另一个开发环境中并在其中进行编译的示例，请参阅在另一个开发环境中编译代码。

输入参数

buildFolder — 编译文件夹

字符向量 | 字符串

编译文件夹的路径，通常包含生成的源代码。该文件夹必须包含 **buildInfo.mat** 文件。

示例： **codebuild(pathToCodeFolder, 'BuildMethod', myToolchain)**

buildInfo — 编译信息对象

对象

包含编译和链接所生成代码的信息的 **RTW.BuildInfo** 对象。

示例： **codebuild(myBuildInfo, 'BuildMethod', 'CMake')**

名称-值对组参数

将可选的参数对组指定为 **Name1=Value1,...,NameN=ValueN**，其中 **Name** 是参数名称，**Value** 是对应的值。名称-值参数必须出现在其他参数后，但参数对组的顺序无关紧要。

在 R2021a 之前，使用逗号分隔每个名称和值，并用引号将 **Name** 引起来。

示例： `codebuild(pathToCodeFolder, 'BuildMethod', myToolchain)`

BuildMethod — 编译方法

字符向量 | 字符串

使用以下编译方法之一：

- 工具链 - 指定工具链的名称，例如 'GNU gcc/g++ | gmake (64-bit Linux)'。
- 模板联编文件 - 指定模板联编文件的路径。
- CMake - 指定 'cmake'，这将为 CMake 编译系统生成 CMakeLists.txt 配置文件。参数值不区分大小写。例如，您也可以指定 'Cmake' 或 'CMake'。

示例： `codebuild(pathToCodeFolder, 'BuildMethod', 'CMake')`

BuildVariant — 编译变体

'STANDALONE_EXECUTABLE' | 'MODEL_REFERENCE_CODER' | 'MEX_FILE' | 'SHARED_LIBRARY' | 'STATIC_LIBRARY'

指定编译输出的类型：

- 'STANDALONE_EXECUTABLE' - 生成独立可执行文件。
- 'MODEL_REFERENCE_CODER' - 生成静态库。
- 'MEX_FILE' - 生成 MEX 文件。此值仅用于编译仿真目标，例如，模型引用仿真目标 (ModelReferenceSimTarget) 和加速模式。
- 'SHARED_LIBRARY' - 生成动态库。
- 'STATIC_LIBRARY' - 生成静态库。

示例： `codebuild(pathToCodeFolder, 'BuildVariant', 'SHARED_LIBRARY')`

输出参数

buildResults — 编译结果

对象

从编译过程中捕获显示输出。要查看显示输出，请在命令行窗口中运行 `disp(buildResults)`。

版本历史记录

在 R2020b 中推出

另请参阅

slbuild | RTW.BuildInfo

主题

在另一个开发环境中编译代码

coder.codedescriptor.CodeDescriptor 类

包: `coder.codedescriptor`

返回有关生成代码的信息

描述

创建一个 `coder.codedescriptor.CodeDescriptor` 对象来访问在代码描述符 API 中定义的所有方法。`coder.codedescriptor.CodeDescriptor` 对象描述生成的代码中的数据接口、函数接口、全局数据存储、局部和全局参数。

创建对象

`codeDescObj = coder.getCodeDescriptor(model)` 为指定的模型创建 `coder.codedescriptor.CodeDescriptor` 对象。

`codeDescObj = coder.getCodeDescriptor(folder)` 为在 `folder` 中指定的编译文件夹中的模型创建 `coder.codedescriptor.CodeDescriptor` 对象。

属性

ModelName — 模型的名称

字符向量 (默认)

为其调用代码描述符对象的模型的名称。

示例: `'rtwdemo_comments'`

属性:

GetAccess **public**

BuildDir — 编译文件夹

字符向量 (默认)

编译模型的编译文件夹的路径。

示例: `'C:\Users\Desktop\Work\rtwdemo_comments_ert_rtw'`

属性:

GetAccess **public**

方法

公共方法

`getAllDataInterfaceTypes`

Return data interface types

`getAllFunctionInterfaceTypes`

Return function interface types

`getArrayLayout`

Return array layout of the generated code

`getDataInterfaceTypes`

Return data interface types in the generated code

getDataInterfaces	Return information of the specified data interface
getFunctionInterfaceTypes	Return function interface types in the generated code
getFunctionInterfaces	Return information of the specified function interface
getReferencedModelCodeDescriptor	Return coder.codedescriptor.CodeDescriptor object for the specified referenced model
getReferencedModelNames	Return names of the referenced models

示例

创建 coder.codedescriptor.CodeDescriptor 对象

- 1 编译模型。

```
slbuild('rtwdemo_comments')
```

- 2 为所需模型创建 coder.codedescriptor.CodeDescriptor 对象。

```
codeDescObj = coder.getCodeDescriptor('rtwdemo_comments')
```

```
    ModelName: 'rtwdemo_comments'
```

```
    BuildDir: 'C:\Users\Desktop\Work\rtwdemo_comments_ert_rtw'
```

- 3 返回一个包含所有可用函数接口类型的列表。

```
allFunctionInterfaceTypes = getAllFunctionInterfaceTypes(codeDescObj)
```

```
    {'Initialize'}
```

```
    {'Output' }
```

```
    {'Update' }
```

```
    {'Terminate' }
```

版本历史记录

在 R2018a 中推出

另请参阅

getCodeDescriptor | coder.descriptor.DataInterface | coder.descriptor.FunctionInterface

主题

"Get Code Description of Generated Code"

packNGo

将生成的代码打包为 zip 文件以进行转移

语法

packNGo(buildInfo,Name,Value)

说明

packNGo(buildInfo,Name,Value) 将代码文件打包为一个压缩 zip 文件，以便您可以将其转移到另一个开发环境中并解包和重新编译它们。名称-值对组列表是可选的。

ZIP 文件可以包含以下类型的文件：

- 源文件（例如，.c、.cu 和 .cpp 文件）
- 头文件（例如，.h、.cuh 和 .hpp 文件）
- 包含编译信息对象的 MAT 文件（.mat 文件）
- 最终可执行文件所需的非编译相关文件（例如，.dll 文件和 .txt 信息文件）
- 编译生成的二进制文件（例如，可执行的 .exe 文件或动态链接库 .dll）。

代码生成器在 zip 文件中包含编译生成的二进制文件（如果存在）。**ignoreFileMissing** 属性不适用于编译生成的二进制文件。

- CMake 配置文件 (CMakeLists.txt)，您可以用它来生成用于编译器环境的联编文件或工程。

使用此函数来转移文件，您之后可以针对特定的目标环境重新编译文件，或者在未安装 MATLAB® 的开发环境中重新编译文件。默认情况下，该函数将这些文件以扁平文件夹结构打包为代码生成文件夹中的一个 zip 文件。您可以通过指定名称-值对组来自定义输出。转移 zip 文件后，使用标准 zip 实用工具解包压缩文件。

packNGo 函数可能会修改传递到第一个 **packNGo** 参数的编译信息。在对代码打包的过程中，**packNGo** 可以根据编译信息中记录的源和包含路径查找其他文件。找到这些文件后，**packNGo** 会将它们添加到编译信息中。

要确保 **packNGo** 能找到头文件，请使用 **addIncludePaths** 函数将头文件的路径添加到 **buildInfo**。

注意 当使用 **codegen** 命令生成独立代码时，您可以使用 **-package** 选项一步生成代码并将代码打包到 ZIP 文件中。

示例

从命令行窗口运行 packNGo

在编译过程完成后，您可以从命令行窗口运行 **packNGo**。使用 **packNGo** 对生成代码进行 zip 文件打包，生成 **portzingbit.zip** 文件。请维护相对文件层次结构。

- 1 将文件夹更改为代码生成文件夹。例如，对于 MATLAB Coder，使用 `codegen/dll/zingbit`；对于 Simulink 代码生成，使用 `zingbit_grt_rtw`。
- 2 加载说明该编译的 `buildInfo` 对象。
- 3 使用 `packType` 和 `fileName` 的属性设置运行 `packNGo`。

```
cd codegen/dll/zingbit;
load buildInfo.mat
packNGo(buildInfo,'packType','hierarchical', ...
    'fileName','portzingbit');
```

在 Simulink 编辑器中配置 packNGo

如果从代码生成窗格中配置 zip 文件打包，代码生成器将在编译过程中使用 `packNGo` 输出一个 zip 文件。

- 1 选择**代码生成 > 代码和工件打包**。或者，提供一个 **Zip 文件名**。要应用更改，请点击**确定**。
- 2 编译模型。在编译过程结束时，代码生成器输出 zip 文件。zip 文件中的文件夹结构是分层结构。

从命令行为 Simulink 配置 packNGo

您可以使用 `set_param` 函数配置 ZIP 文件打包。在编译过程中，代码生成器使用 `packNGo` 来创建 ZIP 文件。

要为模型 `zingbit` 配置 ZIP 文件打包，请运行以下命令：

```
set_param('zingbit','PackageGeneratedCodeAndArtifacts','on');
```

`zingbit.zip` 函数创建文件 `packNGo`。

要为 ZIP 文件指定另一个名称，例如 `portzingbit.zip`，请运行以下命令：

```
set_param('zingbit','PackageGeneratedCodeAndArtifacts','on');
set_param('zingbit','PackageName','portzingbit.zip')
```

输入参数

`buildInfo` — 提供编译信息的对象

`buildInfo` 对象 | `buildInfo.mat` 的路径

在编译过程中，代码生成器将 `buildInfo.mat` 放在代码生成文件夹中。此 MAT 文件包含 `buildInfo` 对象。该对象提供 `packNGo` 用于生成 zip 文件的信息。

您可以将参数指定为一个 `buildInfo` 对象：

```
load buildInfo.mat
packNGo(buildInfo,'packType','hierarchical', ...
    'fileName','portzingbit');
```

您也可以将参数指定为 `buildInfo.mat` 文件的路径：

```
buildInfoFile = fullfile(pathToBuildFolder, 'buildInfo.mat');
packNGo(buildInfoFile, 'packType', 'hierarchical', ...
    'fileName', 'portzingbit');
```

您也可以将参数指定为包含 `buildInfo.mat` 的文件夹的路径：

```
packNGo(pathToBuildFolder, 'packType', 'hierarchical', ...
    'fileName', 'portzingbit');
```

名称-值对组参数

将可选的参数对组指定为 `Name1=Value1,...,NameN=ValueN`，其中 `Name` 是参数名称，`Value` 是对应的值。名称-值参数必须出现在其他参数后，但参数对组的顺序无关紧要。

在 R2021a 之前，使用逗号分隔每个名称和值，并用引号将 `Name` 引起来。

示例：'packType','flat','nestedZipFiles',true

packType — 指定 ZIP 文件的结构类型

'flat'（默认） | 'hierarchical'

如果为 'flat'，则将生成的代码文件以单一扁平文件夹结构打包为一个 zip 文件。函数不打包以下文件：

- `buildInfo.mat` 子文件。
- `CMakeLists.txt` 文件。

如果为 'hierarchical'，则将生成的代码文件以分层结构打包为一个主 zip 文件。层次结构包含顶层模型、引用模型和共享实用工具文件夹。函数还打包以下文件：

- 文件夹对应的 `buildInfo.mat` 文件。
- 编译文件夹中的 `CMakeLists.txt` 文件。

示例：'packType','flat'

nestedZipFiles — 确定次级 zip 文件中的文件路径是否相对于主 zip 文件的根文件夹

true（默认） | false

如果为 true，则创建包含三个次级 zip 文件的主 zip 文件：

- `mlrFiles.zip` - 您的 `matlabroot` 文件夹树中的文件
- `sDirFiles.zip` - 代码生成文件夹中及其下的文件
- `otherFiles.zip` - 不在 `matlabroot` 或 `start` 文件夹树中的必需文件

如果为 false，则创建包含文件夹（例如，您的代码生成文件夹和 `matlabroot`）的主 zip 文件。

示例：'nestedZipFiles',true

fileName — 指定主 zip 文件的文件名

'modelOrFunctionName.zip'（默认） | 'myName'

如果未指定 'fileName'-值对组，该函数会将文件打包到名为 `modelOrFunctionName.zip` 的 zip 文件中，并将该 zip 文件放在代码生成文件夹中。

如果您使用值 'myName' 指定 'fileName'，该函数将在代码生成文件夹中创建 `myName.zip`。

要为主 zip 文件指定另一个位置，请提供该位置的绝对路径，`fullPath/myName.zip`

示例: 'fileName','/home/user/myModel.zip'

minimalHeaders — 选择是否只包含最少的头文件

true (默认) | false

如果为 true, 则在 zip 文件中只包含编译代码所需的最少头文件。

如果为 false, 则在 zip 文件中包含在包含路径上找到的头文件。

示例: 'minimalHeaders',true

includeReport — 选择是否在代码生成报告中包含 html 文件夹

false (默认) | true

如果为 false, 则不在 zip 文件中包含 html 文件夹。

如果为 true, 则在 zip 文件中包含 html 文件夹。

示例: 'includeReport',false

ignoreParseError — 指示 packNGo 在遇到解析错误时不终止

false (默认) | true

如果为 false, 则在遇到解析错误时终止。

如果为 true, 则在遇到解析错误时不终止。

示例: 'ignoreParseError',false

ignoreFileMissing — 指示 packNGo 在文件缺失时不终止

false (默认) | true

如果为 false, 则在遇到文件缺失错误时终止。

如果为 true, 则在遇到文件缺失错误时不终止。

示例: 'ignoreFileMissing',false

限制

- 该函数只能针对源文件运行, 例如 *.c、*.cpp、.cu 和 *.h 文件。该函数不支持编译标志、定义或联编文件。
- 该函数不能将源文件打包用于可重用的库子系统。
- 可能会包含不必要的文件。该函数可能会根据编译信息中记录的源路径和包含路径查找其他文件, 即使这些文件未使用。

版本历史记录

在 R2006b 中推出

另请参阅

codebuild

主题

“Customize Post-Code-Generation Build Processing”

“转移或共享生成的代码”

“Compile Code in Another Development Environment”

rtwbuild

(不推荐) 从模型中编译生成的代码。

注意 不推荐使用 `rtwbuild`。请改用 `slbuild`。

语法

`rtwbuild(model)`

`rtwbuild(model,Name,Value)`

`rtwbuild(subsystem)`

`rtwbuild(subsystem,'Mode','ExportFunctionCalls')`

`blockHandle = rtwbuild(subsystem,'Mode','ExportFunctionCalls')`

说明

`rtwbuild(model)` 根据当前模型配置参数设置从 `model` 生成代码。如果 `model` 尚未加载到 MATLAB 环境中，`rtwbuild` 会在生成代码之前加载它。

如果清除**仅生成代码**模型配置参数，该函数将生成代码并编译可执行映像文件。

为了减少代码生成时间，在重新编译模型时，`rtwbuild` 提供增量模型编译。代码生成器仅在模型或子模型自最近一次模型编译后有变动的情况下才重新编译它们。要强制实施顶层模型编译，请参阅 `'ForceTopModelBuild'` 参数。

`rtwbuild(model,Name,Value)` 使用由一个或多个 `Name,Value` 对组参数指定的其他选项。

`rtwbuild(subsystem)` 根据当前模型配置参数设置从 `subsystem` 生成代码。在开始编译之前，打开（或加载）父模型。

如果清除**仅生成代码**模型配置参数，该函数将生成代码并编译可执行映像文件。

`rtwbuild(subsystem,'Mode','ExportFunctionCalls')` 从包含可导出为外部应用程序代码的函数调用的 `subsystem` 生成代码（需要有 Embedded Coder®）。

如果**创建模块**配置参数设置为“SIL”并且您有 Embedded Coder，则 `blockHandle = rtwbuild(subsystem,'Mode','ExportFunctionCalls')` 返回为从指定子系统生成的代码创建的 SIL 模块的句柄。然后，您可以使用 SIL 模块进行数值等效性测试。

示例

为模型生成代码和构建可执行映像文件

为模型 `rtwdemo_rtwintro` 生成 C 代码。

```
rtwbuild('rtwdemo_rtwintro')
```

对于 GRT 系统目标文件，代码生成器生成以下代码文件，并将它们放在文件夹 `rtwdemo_rtwintro_grt_rtw` 和 `slprj/grt/_sharedutils` 中。

模型文件	共享文件	接口文件
<code>rtwdemo_rtwintro.c</code>	<code>rtGetInf.c</code>	<code>rtmodel.h</code>
<code>rtwdemo_rtwintro.h</code>	<code>rtGetInf.h</code>	
<code>rtwdemo_rtwintro_private.h</code>	<code>rtGetNaN.c</code>	
<code>rtwdemo_rtwintrotypes.h</code>	<code>rtGetNaN.h</code>	
	<code>rt_nonfinite.c</code>	
	<code>rt_nonfinite.h</code>	
	<code>rtwtypes.h</code>	
	<code>multiword_types.h</code>	
	<code>builtin_typeid_types.h</code>	

如果应用以下模型配置参数设置，代码生成器将生成其他结果。

参数设置	结果
代码生成 > 仅生成代码未选中	可执行映像文件 <code>rtwdemo_rtwintro.exe</code>
代码生成 > 报告 > 创建代码生成报告处于选中状态	报告提供有关下列各项的信息和链接：生成的代码文件，子系统和代码接口报告，入口函数、输入端口、输出端口、接口参数和数据存储

强制进行顶层模型构建

为 `TopModelCode` 生成代码和编译可执行映像文件，它引用模型 `ReferenceModelCode` 而不管模型校验和与参数设置如何。

```
openExample('simulinkcoder/FilePackagingModelsCodeAndDataExample','supportingFile','TopModelCode');
rtwbuild('TopModelCode',...
'ForceTopModelBuild',true)
```

为子系统生成代码和编译可执行映像文件

为模型 `rtwdemo_rtwintro` 中的子系统 `Amplifier` 生成 C 代码。

```
rtwbuild('rtwdemo_rtwintro/Amplifier')
```

对于 GRT 目标，代码生成器生成以下代码文件并将它们放在文件夹 `Amplifier_grt_rtw` 和 `slprj/grt/_sharedutils` 中。

模型文件	共享文件	接口文件
Amplifier.c	rtGetInf.c	rtmodel.h
Amplifier.h	rtGetInf.h	
Amplifier_private.h	rtGetNaN.c	
Amplifier_types.h	rtGetNaN.h	
	rt_nonfinite.c	
	rt_nonfinite.h	
	rtwtypes.h	
	multiword_types.h	
	builtin_typeid_types.h	

如果应用下表中列出的参数设置，代码生成器会生成列出的结果。

参数设置	结果
代码生成 > 仅生成代码未选中	可执行映像文件 Amplifier.exe
代码生成 > 报告 > 创建代码生成报告处于选中状态	报告提供有关下列各项的信息和链接：生成的代码文件，子系统和代码接口报告，入口函数、输入端口、输出端口、接口参数和数据存储

编译子系统以用于将代码导出为外部应用程序

要将映像文件导出到外部应用程序代码，请从函数调用子系统编译可执行映像文件。

```
rtwdemo_exporting_functions
rtwbuild('rtwdemo_exporting_functions/rtwdemo_subsystem','Mode','ExportFunctionCalls')
```

可执行映像文件 **rtwdemo_subsystem.exe** 出现在您的工作文件夹中。

创建用于验证的 SIL 模块

从一个函数调用子系统创建一个 SIL 模块，您可以用它来测试从模型中生成的代码。

打开模型 **rtwdemo_exporting_functions** 中的子系统 **rtwdemo_subsystem**，并将**创建模块**模型配置参数设置为 “SIL” 。

创建 SIL 模块。

```
mysilblockhandle = rtwbuild('rtwdemo_exporting_functions/rtwdemo_subsystem',...
'Mode','ExportFunctionCalls')
```

代码生成器为生成的子系统代码生成 SIL 模块。您可以将模块添加到提供测试向量或激励输入的环境或测试框架模型中。然后，您可以运行执行 SIL 测试的仿真，并验证 SIL 模块中的生成代码产生的结果与原始子系统相同。

命名导出的初始化函数

命名从函数调用子系统编译可执行映像文件时生成的初始化函数。

```
rtwdemo_exporting_functions
rtwbuild('rtwdemo_exporting_functions/rtwdemo_subsystem',...
'Mode','ExportFunctionCalls','ExportFunctionInitializeFunctionName','subsysinit')
```

初始化函数名称 `subsysinit` 出现在 `rtwdemo_subsystem_ert_rtw/ert_main.c` 中。

在“编译状态”窗口中显示状态信息

在生成代码并运行模型 `rtwdemo_mdltreftop_witherr` 的并行编译时，在“编译状态”窗口中显示编译信息。

```
rtwbuild('rtwdemo_mdltreftop_witherr', ...
'OpenBuildStatusAutomatically',true)
```

输入参数

model — 要为其生成代码或编译可执行映像文件的模型对象或名称
object | 'modelName'

要为其生成代码或编译可执行映像文件的模型，指定为表示模型名称的对象或字符向量。

示例: `'rtwdemo_exporting_functions'`

subsystem — 要为其生成代码或编译可执行映像文件的子系统名称
'subsystemName'

要为其生成代码或编译可执行映像文件的子系统，指定为表示子系统名称或完整模块路径的字符向量。

示例: `'rtwdemo_exporting_functions/rtwdemo_subsystem'`

名称-值对组参数

将可选的参数对组指定为 `Name1=Value1,...,NameN=ValueN`，其中 `Name` 是参数名称，`Value` 是对应的值。名称-值参数必须出现在其他参数后，但参数对组的顺序无关紧要。

在 R2021a 之前，使用逗号分隔每个名称和值，并用引号将 `Name` 引起来。

示例: `rtwbuild('TopModelCode','ForceTopModelBuild',true)`

ForceTopModelBuild — 强制重新生成顶层模型代码
`false`（默认） | `true`

强制重新生成顶层模型代码，指定为 `true` 或 `false`。

操作	指定
强制代码生成器为包含引用模型的系统的顶层模型重新生成代码	<code>true</code>

操作	指定
指定代码生成器根据模型和模型参数的更改确定是否重新生成顶层模型代码	false

如果您更改了与外部或自定义代码相关联的项目，例如自定义目标的代码，请考虑强制重新生成顶层模型的代码。例如，如果您更改了下列各项，请将 **ForceTopModelBuild** 设置为 **true**：

- TLC 代码
- S-Function 源代码，包括 **rtwmakecfg.m** 文件
- 集成的自定义代码

您也可以通过删除“Code generation folder”中的文件夹（如 **slprj**）或生成的模型代码文件夹，强制为顶层模型重新生成代码。

generateCodeOnly — 仅生成代码

false | true

如果未指定值，**代码生成**窗格中的**仅生成代码** (GenCodeOnly) 选项将控制编译过程行为。

如果指定值，该参数将覆盖**代码生成**窗格中的**进生成代码** (GenCodeOnly) 选项。

操作	指定
仅生成代码。	true
生成代码并编译可执行文件。	false

Mode — 导出函数调用（仅用于子系统编译）

'ExportFunctionCalls' | 'Normal'

- 'ExportFunctionCalls' - 如果您有 Embedded Coder，将从其中包含可导出为外部应用程序代码的函数调用的 **subsystem** 生成代码。
- 'Normal' - 不导出函数调用。

ExportFunctionInitializeFunctionName — 函数名称

字符向量

为指定的子系统命名导出的初始化函数。

示例：

```
rtwbuild(subsystem,'Mode','ExportFunctionCalls','ExportFunctionInitializeFunctionName',fcnname)
```

OpenBuildStatusAutomatically — 在“编译状态”窗口中显示编译信息

false（默认） | true

在“编译状态”窗口中显示编译信息，指定为 **true** 或 **false**。有关使用“编译状态”窗口的详细信息，请参阅“Monitor Parallel Building of Referenced Models”。

“编译状态”窗口支持引用模型层次结构的并行编译。不要使用“编译状态”窗口进行串行编译。

操作	指定
在“编译状态”窗口中显示编译信息	true
无操作	false

ObfuscateCode — 生成经过混淆处理的 C 代码

false (默认) | true

指定是否生成经过混淆处理的 C 代码，指定为 true 或 false。

操作	指定
生成经过混淆处理的 C 代码，您可以将其与第三方共享，从而降低涉及知识产权问题的可能性。	true
无操作。	false

IncludeModelReferenceSimulationTargets — 用于编译模型引用仿真目标的选项

false (默认) | true

用于编译模型引用仿真目标的选项，指定为由 'IncludeModelReferenceSimulationTargets' 和 true 或 false 组成的以逗号分隔的对组。

数据类型：logical

输出参数

blockHandle — 为生成的子系统代码创建的 SIL 模块的句柄

句柄

为生成的子系统代码创建的 SIL 模块的句柄。仅当以下两个条件都得到满足时才返回：

- 您得到使用 Embedded Coder 软件的许可。
- **创建模块**模型配置参数设置为 “SIL” 。

提示

您可以通过以下方式启动代码生成和编译过程：

- 按 **Ctrl+B**。
- 选择**代码 > C/C++ 代码 > 编译模型**。
- 从 MATLAB 命令行调用 **slbuild** 命令。

版本历史记录

在 R2009a 中推出

R2020b: rtwbuild 在默认情况下不生成模型引用仿真目标

R2020b 中的行为有变化

从 R2020b 开始，**rtwbuild** 函数在默认情况下不生成模型引用仿真目标。通过排除模型引用仿真目标，可以加快模型层次结构的代码生成。

通过使用 **IncludeModelReferenceSimulationTargets** 参数，您可以继续使用 **rtwbuild** 函数生成仿真和代码生成目标。

扩展功能

自动并行支持

通过使用 Parallel Computing Toolbox™ 自动运行并行计算来加快代码执行。

要并行编译引用模型，请在顶层模型中，选中配置参数复选框**启用并行模型引用编译**。有关详细信息，请参阅“Reduce Build Time for Referenced Models by Using Parallel Builds”。

在 Parallel Computing Toolbox™ 命令（例如 `parfor` 或 `spmd` 循环）中，不要调用 `rtwbuild`、`rtwrebuild` 或 `slbuild` 命令来编译为并行编译配置的模式。

另请参阅

`codebuild` | `rtwrebuild` | `slbuild` | `coder.buildstatus.open` | `coder.buildstatus.close`

主题

“Build and Run a Program”

“编译从 Simulink 模型生成的代码的方法”

“Control Regeneration of Top Model Code”

“生成要导出到外部代码库的组件源代码” (Embedded Coder)

“软件在环仿真” (Embedded Coder)

RTW.BuildInfo

提供用于编译和链接生成的代码的信息

说明

RTW.BuildInfo 对象包含用于编译和链接生成的代码的信息。

创建对象

语法

buildInformation = RTW.BuildInfo

描述

buildInformation = RTW.BuildInfo 返回一个编译信息对象。您可以使用该对象来指定用于编译和链接生成的代码的信息。例如：

- 编译器选项
- 预处理器标识符定义
- 链接器选项
- 源文件和路径
- 包含文件和路径
- 预编译的外部库

属性

ComponentName — 组件名称

字符向量 | 字符串

生成的代码组件的名称。

对象函数

addCompileFlags	Add compiler options to build information
addDefines	Add preprocessor macro definitions to build information
addIncludeFiles	Add include files to build information
addIncludePaths	Add include paths to build information
addLinkFlags	Add link options to build information
addLinkObjects	Add link objects to build information
addNonBuildFiles	Add nonbuild-related files to build information
addSourceFiles	Add source files to build information
addSourcePaths	Add source paths to build information
addTMFTokens	Add template makefile (TMF) tokens to build information
findBuildArg	Find a specific build argument in build information

findIncludeFiles	Find and add include (header) files to build information
getBuildArgs	Get build arguments from build information
getCompileFlags	Get compiler options from build information
getDefines	Get preprocessor macro definitions from build information
getFullFileList	Get list of files from build information
getIncludeFiles	Get include files from build information
getIncludePaths	Get include paths from build information
getLinkFlags	Get link options from build information
getNonBuildFiles	Get nonbuild-related files from build information
getSourceFiles	Get source files from build information
getSourcePaths	Get source paths from build information
removeSourceFiles	Remove source files from build information object
setTargetProvidesMain	Disable inclusion of code generator provided (generated or static) main.c source file during build
updateFilePathsAndExtensions	Update files in build information with missing paths and file extensions
updateFileSeparator	Update file separator character for file lists in build information

示例

检索编译信息对象

当您编译生成的代码时，编译过程会在 **buildInfo.mat** 文件中存储 **RTW.BuildInfo** 对象。要从包含 **buildInfo.mat** 文件的代码生成文件夹中检索该对象，请运行：

```
bi=load('buildInfo.mat');
bi.buildInfo
```

```
ans =
```

BuildInfo with properties:

```

    ComponentName: 'slexAircraftExample'
        Viewer: []
        Tokens: [27×1 RTW.BuildInfoKeyValuePair]
    BuildArgs: [13×1 RTW.BuildInfoKeyValuePair]
        MakeVars: []
        MakeArgs: ""
    TargetPreCompLibLoc: ""
    TargetLibSuffix: ""
    ModelRefs: []
        SysLib: [1×1 RTW.BuildInfoModules]
        Maps: [1×1 struct]
    LinkObj: []
    Options: [1×1 RTW.BuildInfoOptions]
        Inc: [1×1 RTW.BuildInfoModules]
        Src: [1×1 RTW.BuildInfoModules]
        Other: [1×1 RTW.BuildInfoModules]
        Path: []
    Settings: [1×1 RTW.BuildInfoSettings]
    DisplayLabel: 'Build Info'
        Group: ""

```

该对象包含编译信息。

配置 RTW.BuildInfo 以指定用于编译的代码

此示例说明如何创建 RTW.BuildInfo 对象并注册源文件。

创建一个 RTW.BuildInfo 对象。

```
buildInfo = RTW.BuildInfo;
```

注册源文件。

```
buildInfo.ComponentName = 'foo1';  
addSourceFiles(buildInfo, 'foo1.c');
```

指定编译方法并创建静态库。

```
tmf = fullfile(tmffolder, 'ert_vcx64.tmf');  
buildResult1 = codebuild(pwd, buildInfo, tmf)
```

版本历史记录

在 R2006a 中推出

另请参阅

主题

“Customize Post-Code-Generation Build Processing”

Simulink.fileGenControl

为图更新和模型编译生成的文件指定根文件夹

语法

```
cfg = Simulink.fileGenControl('getConfig')
Simulink.fileGenControl(Action,Name,Value)
```

说明

`cfg = Simulink.fileGenControl('getConfig')` 返回 `Simulink.FileGenConfig` 对象的实例的句柄，该对象包含以下文件生成控制参数的当前值：

- **CacheFolder** - 指定用于仿真的模型编译工件（包括 Simulink® 缓存文件）的根文件夹。
- **CodeGenFolder** - 指定代码生成文件的根文件夹。
- **CodeGenFolderStructure** - 控制代码生成文件夹中的文件夹结构。

要获取或设置参数值，请使用 `Simulink.FileGenConfig` 对象。

以下 Simulink 预设项决定 MATLAB 会话的初始参数值：

- **仿真缓存文件夹** - `CacheFolder`
- **代码生成文件夹** - `CodeGenFolder`
- **代码生成文件夹结构** - `CodeGenFolderStructure`

`Simulink.fileGenControl(Action,Name,Value)` 执行使用当前 MATLAB 会话的文件生成控制参数的操作。可使用一个或多个 `name,value` 对组参数指定其他选项。

示例

获取文件生成控制参数值

要获取当前 MATLAB 会话的文件生成控制参数值，请使用 `getConfig`。

```
cfg = Simulink.fileGenControl('getConfig');

myCacheFolder = cfg.CacheFolder;
myCodeGenFolder = cfg.CodeGenFolder;
myCodeGenFolderStructure = cfg.CodeGenFolderStructure;
```

使用 Simulink.FileGenConfig 对象设置文件生成控制参数

要为当前 MATLAB 会话设置文件生成控制参数值，请使用 `setConfig` 操作。首先，在 `Simulink.FileGenConfig` 对象的实例中设置值。然后，传递对象实例。此示例假设您的系统有 `aNonDefaultCacheFolder` 和 `aNonDefaultCodeGenFolder` 文件夹。

```
% Get the current configuration
cfg = Simulink.fileGenControl('getConfig');
```

```
% Change the parameters to non-default locations
% for the cache and code generation folders
cfg.CacheFolder = fullfile('C:', 'aNonDefaultCacheFolder');
cfg.CodeGenFolder = fullfile('C:', 'aNonDefaultCodeGenFolder');
cfg.CodeGenFolderStructure = 'TargetEnvironmentSubfolder';

Simulink.fileGenControl('setConfig', 'config', cfg);
```

直接设置文件生成控制参数

您可以为当前 MATLAB 会话设置文件生成控制参数值，而无需创建 `Simulink.FileGenConfig` 对象的实例。此示例假设您的系统有 `aNonDefaultCacheFolder` 和 `aNonDefaultCodeGenFolder` 文件夹。

```
myCacheFolder = fullfile('C:', 'aNonDefaultCacheFolder');
myCodeGenFolder = fullfile('C:', 'aNonDefaultCodeGenFolder');

Simulink.fileGenControl('set', 'CacheFolder', myCacheFolder, ...
    'CodeGenFolder', myCodeGenFolder, ...
    'CodeGenFolderStructure', ...
    Simulink.filegen.CodeGenFolderStructure.TargetEnvironmentSubfolder);
```

如果您不想在不同文件夹中为不同目标环境生成代码，请为 `'CodeGenFolderStructure'` 指定值 `Simulink.filegen.CodeGenFolderStructure.ModelSpecific`。

重置文件生成控制参数

您可以将文件生成控制参数重置为 Simulink 预设项中的值。

```
Simulink.fileGenControl('reset');
```

创建仿真缓存和代码生成文件夹

要创建文件生成文件夹，请结合使用 `set` 操作和 `'createDir'` 选项。通过 `'keepPreviousPath'` 选项，您可以保留 MATLAB 路径上以前的文件生成文件夹。

```
%
myCacheFolder = fullfile('C:', 'aNonDefaultCacheFolder');
myCodeGenFolder = fullfile('C:', 'aNonDefaultCodeGenFolder');

Simulink.fileGenControl('set', ...
    'CacheFolder', myCacheFolder, ...
    'CodeGenFolder', myCodeGenFolder, ...
    'keepPreviousPath', true, ...
    'createDir', true);
```

输入参数

Action — 指定操作

`'reset' | 'set' | 'setConfig'`

指定使用当前 MATLAB 会话的文件生成控制参数的操作：

- 'reset' - 将文件生成控制参数重置为 Simulink 预设项中的值。
- 'set' - 通过直接传递值，为当前 MATLAB 会话设置文件生成控制参数。
- 'setConfig' - 通过使用 Simulink.FileGenConfig 对象的实例，为当前 MATLAB 会话设置文件生成控制参数。

名称-值对组参数

将可选的参数对组指定为 Name1=Value1,...,NameN=ValueN，其中 Name 是参数名称，Value 是对应的值。名称-值参数必须出现在其他参数后，但参数对组的顺序无关紧要。

在 R2021a 之前，使用逗号分隔每个名称和值，并用引号将 Name 引起来。

示例：Simulink.fileGenControl(Action, Name, Value);

config — 指定 Simulink.FileGenConfig 的实例

对象句柄

指定包含要设置的文件生成控制参数的 Simulink.FileGenConfig 对象实例。

setConfig 的选项。

示例：Simulink.fileGenControl('setConfig', 'config', cfg);

CacheFolder — 指定仿真缓存文件夹

字符向量

为 CacheFolder 参数指定仿真缓存文件夹路径值。

set 的选项。

示例：Simulink.fileGenControl('set', 'CacheFolder', myCacheFolder);

CodeGenFolder — 指定代码生成文件夹

字符向量

为 CodeGenFolder 参数指定代码生成文件夹路径值。您可以指定绝对路径或相对于编译文件夹的路径。例如：

- 'C:\Work\mymodelsimcache' 和 '/mywork/mymodelgencode' 指定绝对路径。
- 'mymodelsimcache' 是相对于当前工作文件夹 (pwd) 的路径。该软件会在设置 CacheFolder 或 CodeGenFolder 参数时将相对路径转换为完全限定路径。例如，如果 pwd 为 '/mywork'，则结果是 '/mywork/mymodelsimcache'。
- './test/mymodelgencode' 是相对于 pwd 的路径。如果 pwd 为 '/mywork'，则结果是 '/test/mymodelgencode'。

set 的选项。

示例：Simulink.fileGenControl('set', 'CodeGenFolder', myCodeGenFolder);

CodeGenFolderStructure — 指定生成代码的文件夹结构

Simulink.filegen.CodeGenFolderStructure.ModelSpecific (默认) |
Simulink.filegen.CodeGenFolderStructure.TargetEnvironmentSubfolder

指定生成代码的文件夹内子文件夹的布局：

- **Simulink.filegen.CodeGenFolderStructure.ModelSpecific** (默认值) - 将生成的代码放在特定于模型的文件夹内的子文件夹中。
- **Simulink.filegen.CodeGenFolderStructure.TargetEnvironmentSubfolder** - 如果模型是为不同目标环境配置的，请将各个模型生成的代码分别放在不同的子文件夹中。子文件夹的名称对应于目标环境。

set 的选项。

示例: `Simulink.fileGenControl('set', 'CacheFolder', myCacheFolder, ... 'CodeGenFolder', myCodeGenFolder, ... 'CodeGenFolderStructure', ... Simulink.filegen.CodeGenFolderStructure.TargetEnvironmentSubfolder);`

keepPreviousPath — 保留 MATLAB 路径上以前的文件夹路径

false (默认) | true

指定是否保留 MATLAB 路径上以前的 **CacheFolder** 和 **CodeGenFolder** 值:

- **true** - 保留 MATLAB 路径上以前的文件夹路径值。
- **false** (默认值) - 从 MATLAB 路径中删除以前的旧路径值。

reset、set 或 setConfig 的选项。

示例: `Simulink.fileGenControl('reset', 'keepPreviousPath', true);`

createDir — 为文件生成创建文件夹

false (默认) | true

指定如果文件夹不存在，是否为文件生成创建文件夹:

- **true** - 为文件生成创建文件夹。
- **false** (默认值) - 不为文件生成创建文件夹。

set 或 setConfig 的选项。

示例: `Simulink.fileGenControl('set', 'CacheFolder', myCacheFolder, 'CodeGenFolder', myCodeGenFolder, 'keepPreviousPath', true, 'createDir', true);`

避免命名冲突

使用 **Simulink.fileGenControl** 设置 **CacheFolder** 和 **CodeGenFolder** 会将指定文件夹添加到 MATLAB 搜索路径中。此函数的作用与使用 **addpath** 将文件夹添加到搜索路径相同，可能导致名称冲突。例如，如果您为 **CacheFolder** 或 **CodeGenFolder** 指定的文件夹包含与某个打开模型同名的模型文件，则会发生命名冲突。有关详细信息，请参阅“什么是 MATLAB 搜索路径？”和“MATLAB 可访问的文件和文件夹”。

要使用非默认的仿真缓存文件夹或代码生成文件夹位置，请执行以下操作:

- 1 删除以下文件夹中可能导致冲突的任何工件:
 - 当前工作文件夹 **pwd**。
 - 您打算使用的非默认仿真缓存文件夹和代码生成文件夹。
- 2 通过使用 **Simulink.fileGenControl** 或 **Simulink** 预设项，指定仿真缓存文件夹和代码生成文件夹的非默认位置。

输出参数

cfg — 文件生成控制参数的当前值
对象句柄

Simulink.FileGenConfig 对象的实例，它包含文件生成控制参数的当前值。

版本历史记录

在 R2010b 中推出

另请参阅

“Simulation cache folder” | “Code generation folder” | “Code generation folder structure”

主题

“管理编译过程文件夹”

“共享 Simulink 缓存文件以加快仿真速度”

Simulink.ModelReference.modifyProtectedModel

修改现有受保护模型

语法

```
Simulink.ModelReference.modifyProtectedModel(model)
Simulink.ModelReference.modifyProtectedModel(model,Name,Value)
```

```
[harnessHandle] = Simulink.ModelReference.modifyProtectedModel(model,'
Harness',true)
[~,neededVars] = Simulink.ModelReference.modifyProtectedModel(model)
```

说明

Simulink.ModelReference.modifyProtectedModel(model) 修改从指定的 **model** 创建的现有受保护模型的选项。如果未指定 **Name,Value** 对组参数，则修改后的受保护模型将更新为默认值，并且仅支持仿真。

Simulink.ModelReference.modifyProtectedModel(model,Name,Value) 使用由一个或多个 **Name,Value** 对组参数指定的其他选项。这些选项与 **Simulink.ModelReference.protect** 函数提供的选项相同。不过，这些选项还带有附加选项，可用于更改只读视图、仿真和代码生成的加密密码。当您向受保护模型添加功能或更改加密密码时，非受保护模型必须可用。软件会在 MATLAB 路径中搜索模型。如果找不到模型，软件会报告错误。

[harnessHandle] = Simulink.ModelReference.modifyProtectedModel(model,'Harness',true) 为受保护模型创建框架模型。它在 **harnessHandle** 中返回框架模型的句柄。

[~,neededVars] = Simulink.ModelReference.modifyProtectedModel(model) 返回一个元胞数组，其中包含受保护模型所使用的基础工作区变量的名称。

示例

用默认值更新受保护模型

创建一个支持代码生成的可修改的受保护模型，然后将其重置为默认值。

添加修改受保护模型时的密码。如果跳过此步骤，则当创建可修改的受保护模型时，系统会提示您设置密码。

```
openExample('sldemo_mdldref_counter');
Simulink.ModelReference.ProtectedModel.setPasswordForModify(...
'sldemo_mdldref_counter','password');
```

创建一个支持代码生成和 Web 视图的可修改的受保护模型。

```
Simulink.ModelReference.protect('sldemo_mdldref_counter','Mode',...
'CodeGeneration','Modifiable',true,'Report',true);
```

提供修改受保护模型的密码。

```
Simulink.ModelReference.ProtectedModel.setPasswordForModify(...  
'sldemo_mdldref_counter','password');
```

修改模型以使用默认值。

```
Simulink.ModelReference.modifyProtectedModel(...  
'sldemo_mdldref_counter');
```

生成的受保护模型会更新为默认值，并且仅支持仿真。

从受保护模型中删除功能

创建一个支持代码生成和 Web 视图的可修改的受保护模型，然后修改它，删除 Web 视图支持。

添加修改受保护模型时的密码。如果跳过此步骤，则当创建可修改的受保护模型时，系统会提示您设置密码。

```
openExample('sldemo_mdldref_counter');  
Simulink.ModelReference.ProtectedModel.setPasswordForModify(...  
'sldemo_mdldref_counter','password');
```

创建一个支持代码生成和 Web 视图的可修改的受保护模型。

```
Simulink.ModelReference.protect('sldemo_mdldref_counter','Mode',...  
'CodeGeneration','Webview',true,'Modifiable',true,'Report',true);
```

提供修改受保护模型的密码。

```
Simulink.ModelReference.ProtectedModel.setPasswordForModify(...  
'sldemo_mdldref_counter','password');
```

从您创建的受保护模型中删除对 Web 视图的支持。

```
Simulink.ModelReference.modifyProtectedModel(...  
'sldemo_mdldref_counter','Mode','CodeGeneration','Report',true);
```

更改代码生成的加密密码

更改可修改的受保护模型的加密密码。

添加修改受保护模型时的密码。如果跳过此步骤，则当创建可修改的受保护模型时，系统会提示您设置密码。

```
openExample('sldemo_mdldref_counter');  
Simulink.ModelReference.ProtectedModel.setPasswordForModify(...  
'sldemo_mdldref_counter','password');
```

添加受保护模型用户在生成代码时必须提供的密码。

```
Simulink.ModelReference.ProtectedModel.setPasswordForSimulation(...  
'sldemo_mdldref_counter','cgpassword');
```

创建一个可修改的受保护模型，该模型包含报告并支持采用加密的代码生成。

```
Simulink.ModelReference.protect('sldemo_mdhref_counter','Mode',...  
'CodeGeneration','Encrypt',true,'Modifiable',true,'Report',true);
```

提供修改受保护模型的密码。

```
Simulink.ModelReference.ProtectedModel.setPasswordForModify(...  
'sldemo_mdhref_counter','password');
```

更改仿真的加密密码。

```
Simulink.ModelReference.modifyProtectedModel(...  
'sldemo_mdhref_counter','Mode','CodeGeneration','Encrypt',true,...  
'Report',true,'ChangeSimulationPassword',...  
{'cgpassword','new_password'});
```

为受保护模型添加框架模型

为现有受保护模型添加框架模型。

添加修改受保护模型时的密码。如果跳过此步骤，则当创建可修改的受保护模型时，系统会提示您设置密码。

```
openExample('sldemo_mdhref_counter');  
Simulink.ModelReference.ProtectedModel.setPasswordForModify(...  
'sldemo_mdhref_counter','password');
```

创建一个可修改的受保护模型，该模型包含报告并支持采用加密的代码生成。

```
Simulink.ModelReference.protect('sldemo_mdhref_counter','Mode',...  
'CodeGeneration','Modifiable',true,'Report',true);
```

提供修改受保护模型的密码。

```
Simulink.ModelReference.ProtectedModel.setPasswordForModify(...  
'sldemo_mdhref_counter','password');
```

为受保护模型添加框架模型。

```
[harnessHandle] = Simulink.ModelReference.modifyProtectedModel(...  
'sldemo_mdhref_counter','Mode','CodeGeneration','Report',true,...  
'Harness',true);
```

输入参数

model — 模型名称

字符串或字符向量（默认）

模型名称，指定为字符串或字符向量。它包含模型的名称或引用受保护模型的 Model 模块的路径名称。

名称-值对组参数

将可选的参数对组指定为 Name1=Value1,...,NameN=ValueN，其中 Name 是参数名称，Value 是对应的值。名称-值参数必须出现在其他参数后，但参数对组的顺序无关紧要。

在 R2021a 之前，使用逗号分隔每个名称和值，并用引号将 Name 引起来。

示例: 'Mode','CodeGeneration','OutputFormat','Binaries','ObfuscateCode',true 指定为受保护模型生成经过模糊处理的代码。它还指定只有生成代码中的二进制文件和头文件对受保护模型的用户可见。

常规

Path — 受保护模型的文件夹

当前工作文件夹（默认） | 字符串或字符向量

受保护模型的文件夹，指定为字符串或字符向量。

示例: 'Path','C:\Work'

Report — 选择是否生成报告

false（默认） | true

选择是否生成报告，指定为布尔值。

要查看报告，请右键点击受保护模型的徽章图标，然后选择 **Display Report**。或者，使用 **report** 选项调用 **Simulink.ProtectedModel.open** 函数。

报告以 HTML 格式生成。它包括有关受保护模型的环境、功能、许可证要求和接口的信息。

示例: 'Report',true

hdl — 选择是否生成 HDL 代码

false（默认） | true

选择是否生成 HDL 代码，指定为布尔值。

此选项需要 HDL Coder™ 许可证。启用此选项时，请确保指定 **Mode**。您可以将此选项设置为 **true**，同时将 **Mode** 设置为 **CodeGeneration**，以便为受保护模型同时启用 C 代码和 HDL 代码生成支持。

如果您只想启用仿真和 HDL 代码生成支持，而不想启用 C 代码生成，请将 **Mode** 设置为 **HDLCodeGeneration**。您不必将 **hdl** 选项设置为 **true**。

示例: 'hdl',true

Harness — 选择是否创建框架模型

false（默认） | true

选择是否创建框架模型，指定为布尔值。

示例: 'Harness',true

CustomPostProcessingHook — 为受保护模型文件添加后处理函数的选项

函数句柄

为受保护模型文件添加后处理函数的选项，指定为函数句柄。该函数接受 **Simulink.ModelReference.ProtectedModel.HookInfo** 对象作为输入变量。此对象提供关于受保护模型创建期间生成的源代码文件和其他文件的信息。该对象还提供有关您不能修改的导出符号的信息。在打包受保护模型之前，系统会调用后处理函数。

对于具有顶层模型接口的受保护模型，**Simulink.ModelReference.ProtectedModel.HookInfo** 对象无法提供有关导出符号的信息。

示例: 'CustomPostProcessingHook',@(protectedMdlInf)myHook(protectedMdlInf)

功能**Mode — 模型保护模式****'Normal'** (默认) | **'Accelerator'** | **'CodeGeneration'** | **'HDLCodeGeneration'** | **'ViewOnly'**

模型保护模式。指定以下值之一：

- **'Normal'**: 如果顶层模型在 **'Normal'** 模式下运行，受保护模型将作为顶层模型的子模型运行。
- **'Accelerator'**: 顶层模型可以在 **'Normal'**、**'Accelerator'** 或 **'Rapid Accelerator'** 模式下运行。
- **'CodeGeneration'**: 顶层模型可以在 **'Normal'**、**'Accelerator'** 或 **'Rapid Accelerator'** 模式下运行，并支持代码生成。
- **'HDLCodeGeneration'**: 顶层模型可以在 **'Normal'**、**'Accelerator'** 或 **'Rapid Accelerator'** 模式下运行，并支持 HDL 代码生成。
- **'ViewOnly'**: 关闭仿真和生成代码功能模式。打开只读视图模式。

示例: **'Mode','Accelerator'****OutputFormat — 受保护代码可见性****'CompiledBinaries'** (默认) | **'MinimalCode'** | **'AllReferencedHeaders'**

注意 仅当您将 **Mode** 指定为 **'Accelerator'** 或 **'CodeGeneration'** 时，此参数才会影响输出。当您将 **Mode** 指定为 **'Normal'** 时，只有 MEX 文件是输出包的一部分。

受保护代码可见性。此参数确定为受保护模型生成的代码的哪一部分对用户可见。指定以下值之一：

- **'CompiledBinaries'**: 只有二进制文件和头文件可见。
- **'MinimalCode'**: 只包含用所选编译设置编译代码所需的最少头文件。编译文件夹中的代码可见。用户可以检查受保护模型报告中的代码，并根据自己的目的重新编译它。
- **'AllReferencedHeaders'**: 包含 include 路径上的头文件。编译文件夹中的代码可见。代码引用的头文件也可见。

示例: **'OutputFormat','AllReferencedHeaders'****ObfuscateCode — 选择是否对生成的代码进行模糊处理****true** (默认) | **false**

选择是否对生成的代码进行模糊处理，指定为布尔值。仅当对受保护模型启用了代码生成时才适用。HDL 代码生成不支持模糊处理。

示例: **'ObfuscateCode',true****Webview — 选择是否包含 Web 视图****false** (默认) | **true**

选择是否包含受保护模型的只读视图，指定为布尔值。

要打开受保护模型的 Web 视图，请使用以下方法之一：

- 右键点击受保护模型徽章图标，然后选择 **Show Web view**。
- 使用 **Simulink.ProtectedModel.open** 函数。例如，要显示受保护模型 **sldemo_mdhref_counter** 的 Web 视图，您可以调用：

```
Simulink.ProtectedModel.open('sldemo_mdref_counter', 'webview');
```

- 在当前文件夹浏览器中双击 **.slxp** 受保护模型文件。
- 在受保护模型的 Block Parameter 对话框中，点击 **Open Model**。

示例: 'Webview',true

加密

ChangeSimulationPassword — 更改仿真的加密密码的选项

由两个字符向量组成的元胞数组

更改仿真的加密密码的选项，指定为由两个字符向量组成的元胞数组。第一个向量是旧密码，第二个向量是新密码。

示例: 'ChangeSimulationPassword',{'old_password','new_password'}

ChangeViewPassword — 更改只读视图的加密密码的选项

由两个字符向量组成的元胞数组

更改只读视图的加密密码的选项，指定为由两个字符向量组成的元胞数组。第一个向量是旧密码，第二个向量是新密码。

示例: 'ChangeViewPassword',{'old_password','new_password'}

ChangeCodeGenerationPassword — 更改代码生成的加密密码的选项

由两个字符向量组成的元胞数组

更改代码生成的加密密码的选项，指定为由两个字符向量组成的元胞数组。第一个向量是旧密码，第二个向量是新密码。

示例: 'ChangeCodeGenerationPassword',{'old_password','new_password'}

Encrypt — 选择是否对受保护模型进行加密

false (默认) | true

选择是否对受保护模型进行加密，指定为布尔值。当您在保护模型期间指定了密码或使用以下方法时适用：

- 用于模型的只读视图的密码：
`Simulink.ModelReference.ProtectedModel.setPasswordForView`
- 用于仿真的密码：`Simulink.ModelReference.ProtectedModel.setPasswordForSimulation`
- 用于代码生成的密码：
`Simulink.ModelReference.ProtectedModel.setPasswordForCodeGeneration`
- 用于 HDL 代码生成的密码：
`Simulink.ModelReference.ProtectedModel.setPasswordForHDLCodeGeneration`

示例: 'Encrypt',true

输出参数

harnessHandle — 框架模型的句柄

双精度

框架模型的句柄，返回为双精度值或 0，具体取决于 Harness 的值。

如果 **Harness** 为 **true**，则值为框架模型的句柄；否则，值为 **0**。

neededVars — 基础工作区变量的名称

元胞数组

受保护模型使用的基础工作区变量的名称，以元胞数组形式返回。

该元胞数组还可以包含受保护模型不使用的变量。

版本历史记录

在 R2014b 中推出

另请参阅

[Simulink.ModelReference.protect](#) |

[Simulink.ModelReference.ProtectedModel.setPasswordForModify](#)

switchTarget

选择模型配置集的目标

语法

```
switchTarget(myConfigObj,systemTargetFile,[])
switchTarget(myConfigObj,systemTargetFile,targetOptions)
```

说明

`switchTarget(myConfigObj,systemTargetFile,[])` 会更改活动配置集的所选系统目标文件。

`switchTarget(myConfigObj,systemTargetFile,targetOptions)` 设置由 `targetOptions` 指定的配置参数。

示例

获取配置集、默认选项和切换目标

此示例说明如何获取 `model` 的活动配置集，以及如何更改该配置集的系统目标文件。

```
% Get configuration set for model
myConfigObj = getActiveConfigSet(model);
% Switch system target file
switchTarget(myConfigObj,'ert.tlc',[]);
```

获取配置集、设置选项、切换目标

此示例说明如何获取当前模型的活动配置集 (`gcs`)，设置各种 `targetOptions`，然后更改系统目标文件选择。

```
% Get configuration set for current model
myConfigObj=getActiveConfigSet(gcs);

% Specify target options
targetOptions.TLCOptions = '-aVarName=1';
targetOptions.MakeCommand = 'make_rtw';
targetOptions.Description = 'my target';
targetOptions.TemplateMakefile = 'grt_default_tmf';

% Define a system target file
targetSystemFile='grt.tlc';

% Switch system target file
switchTarget(myConfigObj,targetSystemFile,targetOptions);
```

使用 `targetOptions` 来验证值（可选）。

```
% Verify values (optional)
targetOptions

    TLCOptions: '-aVarName=1'
    MakeCommand: 'make_rtw'
    Description: 'my target'
    TemplateMakefile: 'grt_default_tmf'
```

获取配置集、设置工具链编译选项和切换目标

使用选项来选择默认 ERT 目标文件，而不是 `set_param(model,'SystemTargetFile','ert.tlc')`。

```
% use switchTarget to select toolchain build of default ERT target
model='rtwdemo_rtwintro';
open_system(model);

% Get configuration set for model
myConfigObj = getActiveConfigSet(model);

% Specify target options for toolchain build approach
targetOptions.MakeCommand = '';
targetOptions.Description = 'Embedded Coder';
targetOptions.TemplateMakefile = '';

% Switch system target file
switchTarget(myConfigObj,'ert.tlc',targetOptions);
```

输入参数

myConfigObj — 配置集对象
object

`ConfigSet` 的配置集对象或 `Simulink.ConfigSetRef` 的配置引用对象。调用 `getActiveConfigSet` 以获取配置集对象。

示例： `myConfigObj = getActiveConfigSet(model);`

systemTargetFile — 系统目标文件的名称
字符向量

指定出现在**系统目标文件浏览器**中的系统目标文件的名称（例如，对于 Embedded Coder 为 `ert.tlc`，对于 Simulink Coder 为 `grt.tlc`）。

示例： `systemTargetFile = 'ert.tlc';`

targetOptions — 具有提供配置参数选项的字段值的结构体
结构体

具有定义代码生成目标选项的字段的结构体。您可以选择通过在结构体字段中填充值来修改某些配置参数。如果不想使用选项，请指定空结构体 (`[]`)。

targetOptions 中的字段值

指定 `targetOptions` 的结构体字段值。如果您选择不指定选项，请使用空结构体 (`[]`)。

示例： `targetOptions = [];`

TemplateMakefile — 指定模板联编文件的文件名的字符向量
字符向量

示例: `targetOptions.TemplateMakefile = 'myTMF';`

TLCOptions — 指定 TLC 参数的字符向量
字符向量

示例: `targetOptions.TLCOptions = '-aVarName=1';`

MakeCommand — 指定 make 命令 MATLAB 语言文件的字符向量
字符向量

示例: `targetOptions.MakeCommand = 'make_rtw';`

Description — 指定系统目标文件的描述的字符向量
字符向量

示例: `targetOptions.Description = 'Create Visual C/C++ Solution File for Embedded Coder';`

版本历史记录

在 R2009b 中推出

另请参阅

`getActiveConfigSet` | `ConfigSet` | `Simulink.ConfigSetRef`

主题

“以编程方式选择系统目标文件”

“配置系统目标文件”

“Set Target Language Compiler Options”

target 包

管理目标硬件和构建工具信息

说明

使用这些类来管理目标硬件和构建工具信息。例如，为代码生成注册新目标硬件，为外部模式和处理器在环 (PIL) 仿真设置目标连接，或为在开发计算机上编译生成的代码创建自定义 CMake 工具链定义。

类

target.AddOn	Describe add-on properties for target type
target.Alias	Create alternative identifier for target object
target.API	Describe API details
target.APIImplementation	Describe API implementation details
target.ApplicationExecutionTool	Capture system command information to run application from MATLAB computer
target.ApplicationStatus	Describe status of application on target hardware
target.Board	Provide hardware board details
target.Breakpoint	Provide breakpoint details for debugger
target.BuildTool	Describe build tool
target.BuildToolType	Describe build tool type
target.CMake	Specify CMake installation for building generated code
target.CMakeBuildType	Describe CMake build type or build configuration
target.CMakeBuilder	Configure how CMake builds generated code
target.CMakeCacheEntry	Configure a CMake cache entry
target.Command	Capture system command for execution on MATLAB computer
target.BuildDependencies	Describe C and C++ build dependencies to associate with target hardware
target.CommunicationChannel	Describe communication channel properties
target.CommunicationInterface	Describe data I/O details for target hardware
target.CommunicationProtocolStack	Describe communication protocol parameters
target.Connection	Base class for target connection properties
target.ConnectionProperties	Describe target-specific connection properties
target.DebugExecutionTool	Provide MATLAB service interface for debugger to manage processes on target hardware
target.DebugIOTool	Debug byte stream I/O tool service interface
target.Directive	Describe command-line flag for tool
target.EnvironmentConfiguration	Configure system environment for toolchain
target.ExecutionService	Describe implementation of execution service for target application
target.ExecutionTool	MATLAB service interface for tool that manages application execution on target hardware
target.ExternalMode	Represent external mode protocol stack
target.ExternalModeConnectivity	Base class for external mode connectivity options
target.FileType	Define identifier for file type
target.FloatingPointDataType	Describe floating point data type implemented by compiler for target hardware

target.Function	Provide function signature information
target.HardwareComponentSupport	Describe support for a hardware component
target.HostProcessExecutionTool	Capture system command information to run target application from MATLAB computer
target.LanguageImplementation	Provide C and C++ compiler implementation details
target.MainFunction	Provide C and C++ dependencies for main function of target hardware application
target.MakefileBuilder	Specify that toolchain is makefile-based
target.MakeToolType	Describe syntax for makefile type
target.MATLABDependencies	Describe MATLAB class and function dependencies
target.PairedDirective	Describe pair of command-line flags
target.Object	用于目标类型的基类
target.PILProtocol	Describe PIL protocol implementation for target hardware
target.Port	Describe connection via target hardware port
target.PortConnection	Describe target connection port
target.Processor	Provide target processor information
target.ProfilingFreezingOverhead	Capture freezing and unfreezing instrumentation overhead
target.ProfilingFunctionOverhead	Capture function instrumentation overhead
target.ProfilingTaskOverhead	Capture task instrumentation overhead
target.RepeatingDirective	Describe repeated command-line flag for tools
target.RS232Channel	Describe serial communication channel
target.SystemCommandExecutionTool	Capture system command information to run target application from MATLAB computer
target.TargetConnection	Provide details about connecting MATLAB computer to target hardware
target.TCPChannel	Describe TCP communication properties
target.Timer	Provide timer details for processor
target.Tools	Describe properties of tools for target hardware
target.Toolchain	Capture high-level information about toolchain
target.UDPChannel	Describe UDP communication
target.XCP	Describe XCP protocol stack for target hardware
target.XCPExternalModeConnectivity	Represent connectivity options in external mode protocol stack
target.XCPPlatformAbstraction	Specify XCP platform abstraction layer for target hardware
target.XCPTCIPTransport	Represent XCP TCP/IP transport protocol layer
target.XCPTransport	Base class for XCP transport protocol layer
target.XCPSerialTransport	Represent XCP serial transport protocol layer

函数

target.add	Add target object to internal database
target.clear	Clear all target objects from internal database
target.create	Create target object
target.export	Export target object data
target.get	Retrieve target objects from internal database
target.remove	Remove target object from internal database
target.update	Update target objects in internal database
target.upgrade	Upgrade existing definitions of hardware devices

版本历史记录

在 R2019a 中推出

另请参阅

主题

“注册新硬件设备”

“Customise Connectivity for XCP External Mode Simulations”

“Create Custom CMake Toolchain Definition”

“Define Custom Makefile-Based Toolchains Using Target Framework”

target.Object 类

包： target

用于目标类型的基类

描述

target.Object 是一个抽象基类，它使目标类型能够继承常用功能。

类属性

摘要

HandleCompatible

true

true

有关类属性的信息，请参阅“类属性”。

属性

IsValid — 数据有效性

true | false

- 如果对象通过验证过程，则为 true。
- 否则为 false。

属性：

GetAccess

SetAccess

public

private

数据类型： logical

方法

公共方法

validate Validate data integrity of target feature object

示例

验证硬件设备数据

有关使用此类的示例，请参阅“验证硬件设备数据”。

版本历史记录

在 R2019b 中推出

另请参阅

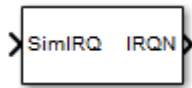
主题

“注册新硬件设备”

Simulink Coder 模块

Async Interrupt

生成执行下游子系统或 Task Sync 模块的 Versa Module Eurocard (VME) 中断服务例程 (ISR)



库:
Simulink Coder / Asynchronous / Interrupt Templates

描述

对于示例 RTOS (VxWorks®) 中的每个指定 VME 中断级别，Async Interrupt 模块都会生成一个中断服务例程 (ISR)，该例程调用以下项之一：

- 函数调用子系统
- Task Sync 模块
- 为函数调用输入事件配置的 Stateflow® 图

注意 使用 Interrupt Templates 模块库中的模块 (Async Interrupt 和 Task Sync) 进行仿真和代码生成。您可用使用这些模块提供的示例作为起点，帮助您为目标环境开发自定义模块。

假设和限制

- 该模块支持 VME 中断 1 到 7。
- 该模块使用这些 RTOS (VxWorks) 系统调用：
 - `sysIntEnable`
 - `sysIntDisable`
 - `intConnect`
 - `intLock`
 - `intUnlock`
 - `tickGet`

性能方面的考虑

在中断级别执行大型子系统会对系统中同等优先级和更低优先级中断的中断响应时间产生重大影响。通常，最好使 ISR 尽可能短。仅将包含几个模块的函数调用子系统连接到 Async Interrupt 模块。

对于大型子系统，更好的解决方案是使用 Task Sync 模块将函数调用子系统的执行与 RTOS 任务同步。将 Task Sync 模块放在 Async Interrupt 模块和函数调用子系统之间。然后，Async Interrupt 模块使用 Task Sync 模块作为 ISR。ISR 向任务释放同步信号量（执行 `semGive`），并立即从中断级别返回。然后，示例 RTOS (VxWorks) 调度并运行该任务。请参阅 Task Sync 模块的说明。

端口

输入

输入 — 仿真的中断源

标量

仿真的中断源。

输出参数

输出 — 控制信号

标量

以下项的控制信号：

- 函数调用子系统
- Task Sync 模块
- 为函数调用输入事件配置的 Stateflow 图

参数

VME interrupt number(s) — 要安装的中断的 VME 中断编号

[1 2] (默认) | 整数数组

要安装的中断的 VME 中断编号数组。有效范围是 1..7。

Async Interrupt 模块输出信号的宽度对应于指定的 VME 中断编号的数目。

注意 一个模型可以包含多个 Async Interrupt 模块。但是，如果您使用多个 Async Interrupt 模块，请不要重复使用在任一模块中指定的 VME 中断编号。

VME interrupt vector offset(s) — 对应于 VME 中断编号的中断向量偏移编号

[192 193] (默认) | 整数数组

对应于为参数 **VME interrupt number(s)** 输入的 VME 中断编号的唯一中断向量偏移编号的数组。Stateflow 软件将偏移传递给 RTOS (VxWorks) 调用 `intConnect(INUM_TO_IVEC(offset),...)`。

Simulink task priority(s) — 下游模块的优先级

[10 11] (默认) | 整数数组

下游模块的 Simulink 优先级。Async Interrupt 模块的每个输出驱动一个下游模块（例如，一个函数调用子系统）。指定与您为参数 **VME interrupt number(s)** 指定的 VME 中断编号对应的优先级的数组。

生成速率转换代码需要参数 **Simulink task priority** 的值（请参阅“Rate Transitions and Asynchronous Blocks”）。当异步任务必须从其基本速率或调用方获得真实时间时，还需要 Simulink 任务优先级值来保持绝对时间完整性。分配的优先级通常高于分配给周期性任务的优先级。

注意 Simulink 软件不仿真异步任务行为。异步任务的任务优先级仅用于代码生成目的，在仿真期间不适用。

Preemption flag(s); preemptable-1; non-preemptable-0 — 选择抢占

[0 1] (默认) | 整数数组

如果 Async Interrupt 模块的输出信号驱动一个 Task Sync 模块，则将此选项设置为 1。

在示例 RTOS (VxWorks) 中，优先级较高的中断可以抢占优先级较低的中断。要在执行 ISR 期间锁定中断，请将抢占标志设置为 0。此设置会导致在 ISR 代码的开头和结尾生成 `intLock()` 和 `intUnlock()` 调用。请务必谨慎使用中断锁定，因为它会增大系统对 `intLockLevelSet()` 级别及以下级别的中断的中断响应时间。指定与为参数 **VME interrupt number(s)** 输入的 VME 中断编号对应的标志的数组。

注意 指定参数 **VME interrupt vector offset(s)** 和 **Simulink task priority** 的数组中的元素数必须与为参数 **VME interrupt number(s)** 指定的数组中的元素数匹配。

Manage own timer — 选择计时器管理器

on (默认) | off

如果选中，由 Async Interrupt 模块生成的 ISR 通过从硬件计时器读取绝对时间来管理它自己的计时器。使用参数 **Timer size** 指定硬件计时器的大小。

Timer resolution (seconds) — ISR 计时器的分辨率

1/60 (默认)

ISR 计时器的分辨率。Async Interrupt 模块生成的 ISR 维护自己的绝对时间计数器。默认情况下，这些计时器通过使用 **tickGet** 调用从 RTOS (VxWorks) 核获取其值。参数 **Timer resolution** 确定这些计数器的分辨率。默认分辨率为 1/60 秒。您的板支持包 (BSP) 的 **tickGet** 分辨率可能与之不同。请确定您的 BSP 的 **tickGet** 分辨率，并在参数 **Timer resolution** 中输入该分辨率。

如果您的目标是 RTOS (VxWorks) 示例以外的 RTOS，请将 **tickGet** 调用替换为对目标 RTOS 的等效调用。或者，生成代码来读取目标硬件上的计时器寄存器。有关详细信息，请参阅“Timers in Asynchronous Tasks”和“Async Interrupt Block Implementation”。

Timer size — 存储时钟计时单元的位数

“32bits” (默认) | “16bits” | “8bits” | “自动”

存储硬件计时器的时钟计时单元的位数。当您选择参数 **Manage own timer** 时，由 Async Interrupt 模块生成的 ISR 使用该计时器大小。大小可以是“32bits”（默认值）、“16bits”、“8bits”或“auto”。如果选择“auto”，代码生成器将根据参数 **Application lifespan (days)** 和 **Timer resolution** 的设置来确定计时器大小。

默认情况下，计时器值存储为 32 位整数。当参数 **Timer size** 设置为“auto”时，您可以通过设置参数 **Application lifespan (days)** 来间接控制计数器的字长。如果您为 **Application lifespan (days)** 设置的值对于代码生成器来说太大而无法作为指定分辨率的 32 位整数来处理，则代码生成器使用另一个 32 位整数来处理溢出。

有关详细信息，请参阅“Control Memory Allocation for Time Counters”。另请参阅“Timers in Asynchronous Tasks”。

Enable simulation input — 选择添加仿真输入端口

on (默认) | off

如果选中，则 Simulink 软件会向 Async Interrupt 模块添加输入端口。此端口仅用于仿真。将一个或多个仿真的中断源连接到仿真输入。

注意 在生成代码之前，请考虑删除驱动仿真输入的模块，以防止这些模块对生成的代码产生影响。您也可以使用 **Variant control mode** 参数设置为“sim codegen switching”的 Variant Source 模块，如“Dual-Model Approach: Code Generation”中所述。如果使用 Variant Source 模块，驱动模块的采样时间会影响生成的代码中支持的采样时间。

版本历史记录

在 R2006a 中推出

另请参阅

Task Sync

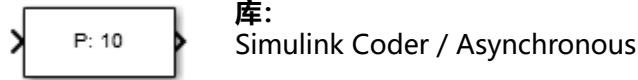
主题

"异步事件"

"异步事件"

Asynchronous Task Specification

指定由异步中断触发的引用模型表示的异步任务的优先级



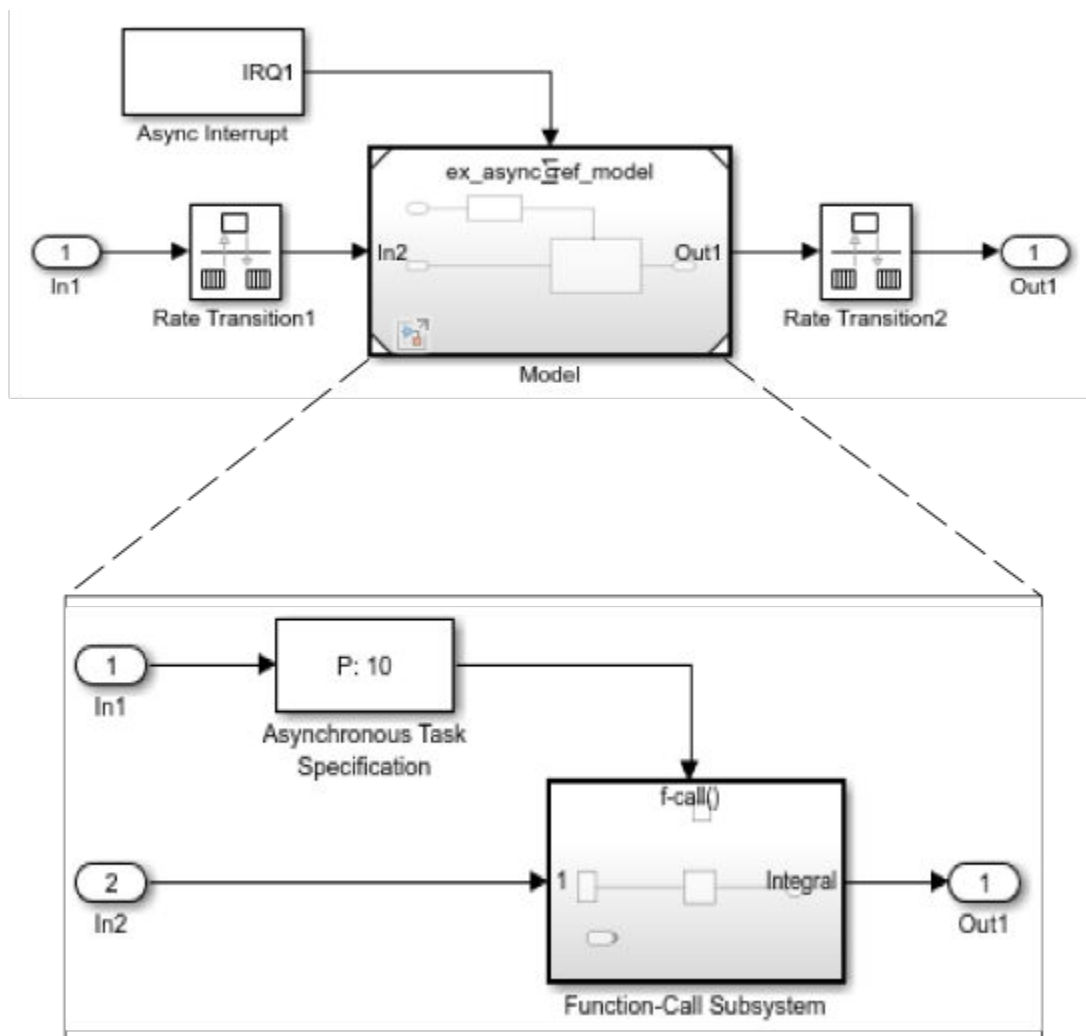
说明

Asynchronous Task Specification 模块指定异步任务的参数，如任务优先级。异步任务由带异步中断触发器的函数调用子系统表示。此模块用于控制带异步事件触发器的函数调用子系统的调度。您可以通过为引用模型中的每个函数调用子系统分配优先级来控制调度。

要使用此模块，请按照“Convert an Asynchronous Subsystem into a Model Reference”中的过程进行操作。

通过观察下图可知：

- 该模块必须位于根级 Inport 模块和函数调用子系统之间的引用模型中。Asynchronous Task Specification 模块必须紧跟在 Inport 模块之后并直接连接到该模块。这两个模块的组合构成名为 JMAAB-B 的建模风格。
- Inport 模块必须从父模型中的 Async Interrupt 模块接收中断信号。
- Inport 模块必须配置为接收和发送函数调用触发信号。



端口

输入

Port_1 — 中断输入信号

标量

从根级 Inport 模块接收的中断输入信号。

输出

Port_1 — 具有优先级的中断信号

标量

触发函数调用子系统的具有指定任务优先级的中断信号。

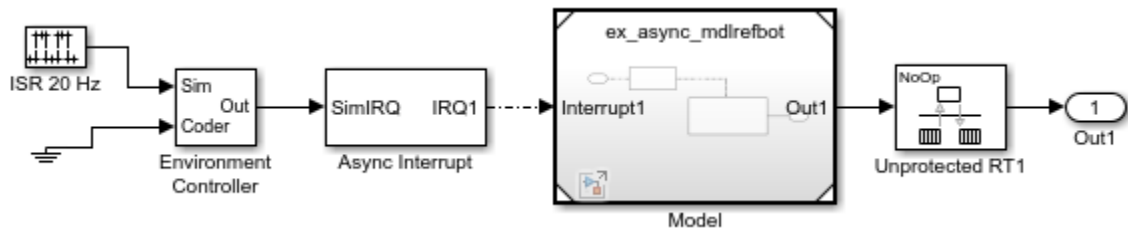
参数

任务优先级 — 调用函数调用子系统的异步任务的优先级
10 (默认)

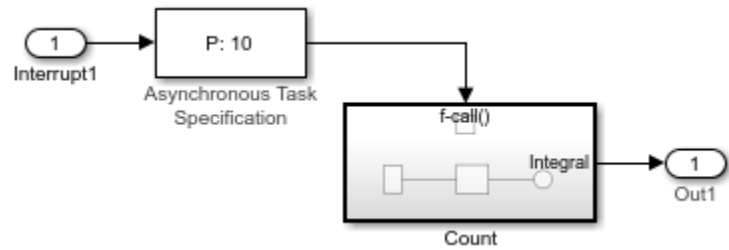
指定一个整数或 [] 作为调用所连接的函数调用子系统的异步任务的优先级。优先级必须为可产生相关速率转换行为的值。

- 如果指定一个整数，它必须与父模型中的中断信号发起方的优先级值匹配。
- 如果您指定 []，则优先级不必与顶层模型中的中断信号发起方的优先级匹配。速率转换算法是保守算法（未优化）。优先级未知，但为静态。

假定有以下模型。



引用模型具有以下内容。



如果 **Task priority** 参数设置为 10，则父模型中的 Async Interrupt 模块的优先级也必须为 10。如果参数设置为 []，Async Interrupt 模块的优先级可以是 10 以外的值。

版本历史记录
在 R2011a 中推出

另请参阅

模块
Inport | Function-Call Subsystem

- 主题
- “异步事件”
 - “生成并同步 RTOS 任务的执行”
 - “Pass Asynchronous Events in RTOS as Input to a Referenced Model”
 - “Convert an Asynchronous Subsystem into a Model Reference”

“Rate Transitions and Asynchronous Blocks”

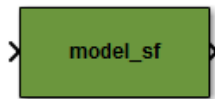
“异步支持”

“异步事件”

“模型引用”

Generated S-Function

将模型或子系统表示为生成的 S-Function 代码



库：
Simulink Coder / S-Function Target

描述

Generated S-Function 模块的实例表示代码生成器从其模型或子系统的 S-Function 系统目标文件生成的代码。

要求

- S-Function 模块必须与生成它的对应模型或子系统执行相同的操作。
- 在创建模块之前，显式指定 Inport 模块信号属性，如信号宽度或采样时间。此规则的唯一例外是采样时间，如“生成的 S-Function 中的采样时间传播”所述。
- 将 Generated S-Function 模块的求解器参数设置为与原始模型或子系统的参数相同。生成的 S-Function 代码与原始子系统具有完全相同的操作。有关此规则的例外情况，请参阅“具有生成的 S-Function 的顶层模型的求解器类型”。

端口

输入

Input — S-Function 输入

视情况而定

请查看需求。

输出参数

Output — S-Function 输出

视情况而定

请查看需求。

参数

生成的 S-Function 名称(model_sf) — S-Function 的名称

model_sf (默认) | 字符向量

生成的 S-Function 的名称。代码生成器通过将 _sf 追加到从其生成模块的模型或子系统的名称来派生该名称。

显示模块列表 — 选择显示模块列表

off (默认) | on

如果选中，则显示系统为 S-Function 生成的模块。

版本历史记录

在 R2011b 中推出

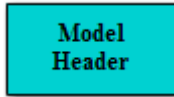
另请参阅

主题

“将 S-Function 目标用于模型或子系统”

Model Header

指定外部头代码



库:
Simulink Coder / Custom Code

描述

对于包含 Model Header 模块的模型，代码生成器会将您指定的外部代码添加到它生成的头文件 (`model.h`) 中。您可以指定代码生成器要添加到头文件顶部和底部附近的代码。

如果在引用模型中包含此模块，则代码生成器在编译仿真目标时将忽略该模块，但会在其他系统目标文件中处理该模块。

注意 此模块旨在满足特殊情况下需将自定义代码插入到所生成代码中的要求。请参阅 C Caller 和 C Function 模块，它们通常用于将自定义算法代码集成到模型中。

参数

模型头文件的顶部 — 要添加到生成的头文件顶部附近的代码

无默认值

指定需要代码生成器添加到模型头文件顶部附近的代码。代码生成器将代码放在标注为 **user code (top of header file)** 的节中。

模型头文件的底部 — 要添加到生成的头文件底部的代码

无默认值

指定需要代码生成器添加到模型头文件底部的代码。代码生成器将代码放在标注为 **user code (bottom of header file)** 的节中。

版本历史记录

在 R2006a 中推出

另请参阅

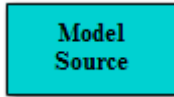
Model Source | System Derivatives | System Disable | System Enable | System Initialize | System Outputs | System Start | System Terminate | System Update

主题

"Place External C/C++ Code in Generated Code"

Model Source

指定外部源代码



库:
Simulink Coder / Custom Code

描述

对于包含 Model Source 模块的模型，代码生成器会将您指定的外部代码添加到它生成的源文件（`model.c` 或 `model.cpp`）中。您可以指定代码生成器要添加到源文件顶部和底部附近的代码。

如果在引用模型中包含此模块，则代码生成器在编译仿真目标时将忽略该模块，但会在其他系统目标文件中处理该模块。

注意 此模块旨在满足特殊情况下需将自定义代码插入到所生成代码中的要求。请参阅 C Caller 和 C Function 模块，它们通常用于将自定义算法代码集成到模型中。

参数

模型源顶部 — 要添加到生成的源文件顶部附近的代码

无默认值

指定需要代码生成器添加到模型源文件顶部附近的代码。代码生成器将代码放在标注为 **user code (top of source file)** 的节中。

模型源底部 — 要添加到生成的源文件底部的代码

无默认值

指定需要代码生成器添加到模型源文件底部的代码。代码生成器将代码放在标注为 **user code (bottom of source file)** 的节中。

版本历史记录

在 R2006a 中推出

另请参阅

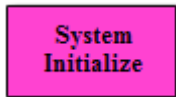
Model Header | System Derivatives | System Disable | System Enable | System Initialize | System Outputs | System Start | System Terminate | System Update

主题

“Place External C/C++ Code in Generated Code”

System Initialize

指定外部系统初始化代码



库:
Simulink Coder / Custom Code

描述

对于包含 System Initialize 模块和使用 `SystemInitialize` 函数的模块的模型或非虚拟子系统，代码生成器会将您指定的外部代码添加到它生成的 `SystemInitialize` 函数中。您可以为代码生成器指定要添加到函数代码的声明、执行和退出节的代码。

如果在引用模型中包含此模块，则代码生成器在编译仿真目标时将忽略该模块，但会在其他系统目标文件中处理该模块。

注意 此模块旨在满足特殊情况下需将自定义代码插入到所生成代码中的要求。请参阅 C Caller 和 C Function 模块，它们通常用于将自定义算法代码集成到模型中。

参数

系统初始化函数声明代码 — 要添加到生成函数的声明节的代码

无默认值

指定希望代码生成器添加到模型或子系统的 `SystemInitialize` 函数的声明节的代码。

系统初始化函数执行代码 — 要添加到生成函数的执行节的代码

无默认值

指定希望代码生成器添加到模型或子系统的 `SystemInitialize` 函数的执行节的代码。

系统初始化函数退出代码 — 添加到生成函数的退出节的代码

无默认值

指定希望代码生成器添加到模型或子系统的 `SystemInitialize` 函数的退出节的代码。

版本历史记录

在 R2006a 中推出

另请参阅

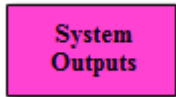
Model Header | Model Source | System Derivatives | System Disable | System Enable | System Outputs | System Start | System Terminate | System Update

主题

"Place External C/C++ Code in Generated Code"

System Outputs

指定外部系统输出代码



库:
Simulink Coder / Custom Code

描述

对于包含 System Outputs 模块和使用 **SystemOutputs** 函数的模块的模型或非虚拟子系统，代码生成器会将您指定的外部代码添加到它生成的 **SystemOutputs** 函数中。您可以为代码生成器指定要添加到函数代码的声明、执行和退出节的代码。

如果在引用模型中包含此模块，则代码生成器在编译仿真目标时将忽略该模块，但会在其他系统目标文件中处理该模块。

注意 此模块旨在满足特殊情况下需将自定义代码插入到所生成代码中的要求。请参阅 C Caller 和 C Function 模块，它们通常用于将自定义算法代码集成到模型中。

参数

系统输出函数声明代码 — 要添加到生成函数的声明节的代码

无默认值

指定希望代码生成器添加到模型或子系统的 **SystemOutputs** 函数的声明节的代码。

系统输出函数执行代码 — 要添加到生成函数的执行节的代码

无默认值

指定希望代码生成器添加到模型或子系统的 **SystemOutputs** 函数的执行节的代码。

系统输出函数退出代码 — 添加到生成函数的退出节的代码

无默认值

指定希望代码生成器添加到模型或子系统的 **SystemOutputs** 函数的退出节的代码。

版本历史记录

在 R2006a 中推出

另请参阅

Model Header | Model Source | System Derivatives | System Disable | System Enable | System Initialize | System Start | System Terminate | System Update

主题

"Place External C/C++ Code in Generated Code"

仿真目标参数

自动调节

描述

对 cuDNN 库使用自动调整。

类别：仿真目标

设置

默认值：On



On

使用此选项启用自动调节功能。启用自动调整后，cuDNN 库可找到最快的卷积算法。这会提高 SegNet 和 ResNet 等大型网络的性能。



Off

禁用 cuDNN 库的自动调节。

依存关系

- 此参数要求具有 GPU Coder™ 许可证。
- 要启用此参数，请在**语言**下选择 “C++”，在**目标库**下选择 “cuDNN”。

命令行信息

参数：SimDLAutoTuning

值：'on' | 'off'

默认值：'on'

另请参阅

相关示例

- “模型配置参数：仿真目标”
- “GPU Code Generation for Deep Learning Networks Using MATLAB Function Block” (GPU Coder)

GPU 加速参数

代码生成参数：代码生成

模型配置参数：代码生成

代码生成类别包括用于定义代码生成过程的参数，包括目标选择。它还包括用于在数据和函数的生成代码中插入注释和 pragma 指令的参数。这些参数需要具备 Simulink Coder 许可证。适用于基于 ERT 的目标的其他参数需要具备 Embedded Coder 许可证。为 NVIDIA® GPU 生成 CUDA® C++ 代码需要 GPU Coder 许可证。

这些配置参数出现在**配置参数 > 代码生成**常规类别中。

参数	描述
System target file	指定将使用的目标文件配置。
“浏览” (第 7-5 页)	浏览文件配置选项。
Shared coder dictionary (Embedded Coder)	包含代码接口配置的共享代码生成器字典。
Language	指定 C 或 C++ 代码生成。
Language standard	为您的执行环境指定语言标准。
“Generate GPU code”	使用 GPU Coder 进行 CUDA 代码生成。 此参数要求具有 GPU Coder 许可证。
“Description”	目标文件描述。
Generate code only	指定是只生成代码还是生成可执行文件。
Package code and artifacts	指定是否自动打包生成的代码和工件以便进行转移。
Zip file name	指定 .zip 文件的名称，生成的代码和工件会打包到该文件中以便进行转移。
Compiler optimization level	控制用于编译生成的代码的编译器优化。
Custom compiler optimization flags	指定自定义编译器优化标志。
Toolchain	指定在编译可执行文件或库时要使用的工具链。
Build configuration	为工具链指定编译器优化或调试设置。
Toolchain details	显示或自定义编译配置设置。
Generate makefile	启用基于模板联编文件生成联编文件。
Make command	指定 make 命令和（可选地）附加联编文件选项。
Template makefile	指定生成联编文件所依据的模板联编文件。
Select objective	选择一个代码生成目标以与代码生成顾问结合使用。
Prioritized objectives (Embedded Coder)	按优先级排列的代码生成目标列表。
“Set Objectives” (Embedded Coder)	打开“配置集目标”对话框。
“Set Objectives — Code Generation Advisor Dialog Box” (Embedded Coder)	选择代码生成目标并确定其优先级。
Check model before generating code	选择在生成代码之前是否运行代码生成顾问检查。
“Check Model”	检查模型是否满足代码生成目标。

这些配置参数位于**高级参数**下。

参数	描述
Custom FFT library callback	在为 MATLAB 代码中的 FFT 函数生成的代码中，为 FFTW 库调用指定回调类。
Custom BLAS library callback	在从 MATLAB 代码生成的代码中为 BLAS 调用指定 BLAS 库回调类。
Custom LAPACK library callback	在从 MATLAB 代码生成的代码中为 LAPACK 调用指定 LAPACK 库回调类。
Verbose build	显示代码生成进度。
Retain .rtw file	指定 model.rtw 文件保留。
Profile TLC	探查 TLC 文件的执行时间。
Enable TLC assertion	生成 TLC 堆栈跟踪。
Start TLC coverage when generating code	生成 TLC 执行报告。
Start TLC debugger when generating code	指定 TLC 调试器的使用
Show Custom Hardware App in Simulink Toolstrip	Simulink 工具条的只读内部参数。
Show Embedded Hardware App in Simulink Toolstrip	Simulink 工具条的只读内部参数。
"Package" (Embedded Coder)	指定一个包，其中包含要应用于模型级函数和内部数据的内存段。
"Refresh package list" (Embedded Coder)	将搜索路径上的用户定义的包添加到包列表中。
"初始化/终止" (Embedded Coder)	指定是否将内存段应用于 Initialize/Start 和 Terminate 函数。
"Execution" (Embedded Coder)	指定是否将内存段应用于执行函数。
"Shared utility" (Embedded Coder)	指定是否将内存段应用于共享工具函数。
"Constants" (Embedded Coder)	指定是否将内存段应用于常量。
"Inputs/Outputs" (Embedded Coder)	指定是否将内存段应用于根输入和输出。
"Internal data" (Embedded Coder)	指定是否将内存段应用于内部数据。
"Parameters" (Embedded Coder)	指定是否将内存段应用于参数。
"Validation results" (Embedded Coder)	显示内存段验证的结果。

高级参数下的以下参数不常使用，没有其他文档。

参数	描述
PostCodeGenCommand character vector - "	将指定的后期代码生成命令添加到模型编译过程中。

参数	描述
TLCOptions character vector - "	指定其他 TLC 命令行选项。 当您对配置了服务接口的组件模型使用 Embedded Coder 时，您无法指定 - aInlineSetEventsForThisBaseRateFcn=TLC_FALSE , - aSuppressMultiTaskScheduler=TLC_FALSE , - -aRateBasedStepFcn=TLCFALSE , - (wrapper function)

以下参数仅供 MathWorks 使用。

参数	描述
Comment	仅供 MathWorks 使用。
PreserveName	仅供 MathWorks 使用。
PreserveNameWithParent	仅供 MathWorks 使用。
SignalNamingFcn	仅供 MathWorks 使用。
TargetTypeEmulationWarnSuppressLevel int - 0	仅供 MathWorks 使用。 当大于或等于 2 时，隐藏在快速原型环境中模拟整数大小时代码生成器显示的警告消息。

“配置参数”对话框还包括其他代码生成参数：

- “模型配置参数：代码生成优化”（第 19-2 页）
- “Model Configuration Parameters: Code Generation Report”
- “Model Configuration Parameters: Comments”
- “模型配置参数：代码生成标识符”（第 10-2 页）
- “模型配置参数：代码生成自定义代码”（第 11-2 页）
- “模型配置参数：代码生成接口”（第 12-2 页）

另请参阅

详细信息

- “模型配置”
- “Control Data and Function Placement in Memory by Inserting Pragmas” (Embedded Coder)

浏览

描述

打开系统目标文件浏览器，您可以在其中选择预设目标配置，其中包含系统目标文件、模板联编文件和 make 命令。您选择的值将填入 **System target file**。

类别：代码生成

提示

- System Target File Browser 列出了在 MATLAB 路径上找到的系统目标文件。一些系统目标文件需要额外的许可产品，例如 Embedded Coder 产品。
- 要为快速仿真配置模型，请选择 **rsim.tlc**。
- 要为 Simulink Real-Time™ 配置您的模型，请选择 **slrealtime.tlc**。

另请参阅

相关示例

- “模型配置参数：代码生成”（第 7-2 页）
- “配置系统目标文件”
- “比较各产品的系统目标文件支持”

代码生成参数：报告

代码生成参数：注释

Simulink 模块注释

包括 Simulink 模块注释

模型配置窗格： 代码生成 / 注释

描述

指定是否插入 Simulink 模块注释。

依存关系

- **包括注释**启用此参数。
- 此参数启用**使用以下方式追溯至模型** (Embedded Coder)。

设置

on (默认) | off

On

插入自动生成的描述模块代码的注释。在生成的文件中，注释位于描述的生成代码之前。

Off

隐藏注释。

推荐的设置

应用场景	设置
调试	On
可追溯性	On
效率	无影响
安全预警	无建议

编程用法

参数: SimulinkBlockComments

类型: 字符向量

值: 'on' | 'off'

默认值: 'on'

版本历史记录

在 R2006a 之前推出

另请参阅

主题

"Model Configuration Parameters: Comments"

“Trace Simulink Model Elements in Generated Code” (Embedded Coder)

代码生成参数：标识符

模型配置参数：代码生成标识符

代码生成 > 标识符类别包括用于在生成的代码中配置注释的参数。这些参数需要具备 Simulink Coder 许可证。适用于基于 ERT 的目标的其他参数需要具备 Embedded Coder 许可证。

在“配置参数”对话框中，以下配置参数位于**代码生成 > 标识符**窗格上。

参数	描述
Global variables (Embedded Coder)	自定义生成的全局变量标识符。
Global types (Embedded Coder)	自定义生成的全局类型标识符。
Field name of global types (Embedded Coder)	自定义生成的全局类型字段名称。
Subsystem methods (Embedded Coder)	为可重用子系统自定义生成的函数名称。
Subsystem method arguments (Embedded Coder)	为可重用子系统自定义生成的函数参数名称。
Local temporary variables (Embedded Coder)	自定义生成的局部临时变量标识符。
Local block output variables (Embedded Coder)	自定义生成的局部模块输出变量标识符。
Constant macros (Embedded Coder)	自定义生成的常量宏标识符。
Shared utilities identifier format (Embedded Coder)	自定义共享实用工具标识符。
Minimum mangle length (Embedded Coder)	指定用于生成名称修饰文本的最小字符数，以帮助避免名称冲突。
Maximum identifier length	指定生成的函数、类型定义、变量名称中的最大字符数。
System-generated identifiers (Embedded Coder)	指定代码生成器是否在系统生成的标识符中为 \$N 标记使用更短、更一致的名称。
Generate scalar inlined parameters as (Embedded Coder)	控制生成的代码中内联参数值标量的表示。
Use the same reserved names as Simulation Target	指定是否使用与 仿真目标 窗格中指定的名称相同的保留名称。
Reserved names	输入生成的代码中与自定义代码中指定的变量或函数名称匹配的变量或函数名称。

以下配置参数位于**高级参数**下。

参数	描述
Shared checksum length (Embedded Coder)	指定 \$C 标记的字符长度。
EMX array utility functions identifier format (Embedded Coder)	为 emxArray （可嵌入的 mxArray ）工具函数自定义生成的标识符。
EMX array types identifier format (Embedded Coder)	为 emxArray （可嵌入的 mxArray ）类型自定义生成的标识符。
Custom token text (Embedded Coder)	指定要为 \$U 标记插入的文本。

参数	描述
Duplicate enumeration member names	选择代码生成器检测到两个成员名称相同的枚举类型时要执行的诊断操作。此参数仅适用于具有导入的数据作用域以及相同存储类型和值的枚举。
Signal naming (Embedded Coder)	指定在生成的代码中命名信号的规则。
M-function (Embedded Coder)	
Parameter naming (Embedded Coder)	指定在生成的代码中命名参数的规则。
M-function (Embedded Coder)	
#define naming (Embedded Coder)	指定在生成的代码中命名 #define 参数（用存储类 "Define (Custom)" 定义）的规则。
M-function (Embedded Coder)	

另请参阅

详细信息

- “Model Configuration Set Customization”

代码生成参数：自定义代码

模型配置参数：代码生成自定义代码

代码生成 > 自定义代码类别包括用于将自定义 C 代码插入生成代码的参数。这些参数需要具备 Simulink Coder 许可证。

在“配置参数”对话框中，以下配置参数位于**代码生成 > 自定义代码**窗格中。

参数	描述
Use the same custom code settings as Simulation Target	指定是否使用与 仿真目标 > 自定义代码 窗格中相同的自定义代码设置。
Additional code	指定要包含在生成的模型源文件顶部附近的自定义代码。
Include headers	指定要包含在生成的模型头文件顶部附近的自定义代码。
Initialize code	指定要包含在生成的模型初始化函数中的自定义代码。
Terminate code	指定要包含在生成的模型终止函数中的自定义代码。
Include directories	指定要添加到包含路径的包含文件夹的列表。
Source files	指定要编译并与生成的代码链接的附加源文件列表。
库	指定要与生成的代码链接的附加库列表。
Defines	指定要添加到编译器命令行的预处理器宏定义。

另请参阅

详细信息

- “Model Configuration Set Customization”

库

用生成的代码链接附加库

模型配置窗格： 代码生成 / 自定义代码

描述

指定要与生成的代码链接的附加库列表。

设置

" (默认) | 字符串

输入要与生成代码链接的静态库文件的空格分隔列表。

提示

如果文件位于当前 MATLAB 文件夹或其中一个包含文件夹中，可以只指定文件名。

推荐的设置

应用场景	设置
调试	无影响
可追溯性	无影响
效率	无影响
安全预警	无影响

编程用法

参数: CustomLibrary

类型: 字符向量

值: 库文件名

默认值: "

限制

此参数不支持包含空格的 Windows® 文件名。

版本历史记录

在 R2006a 之前推出

另请参阅

主题

“模型配置参数：代码生成自定义代码” (第 11-2 页)

“Integrate External Code by Using Model Configuration Parameters”

代码生成参数：接口

模型配置参数：代码生成接口

代码生成 > 接口类别包括用于配置生成代码接口的参数。这些参数需要具备 Simulink Coder 许可证。适用于基于 ERT 的目标的其他参数需要具备 Embedded Coder 许可证。使用 NVIDIA CUDA 深度神经网络库 (cuDNN) 或适用于 NVIDIA GPU 的 TensorRT 高性能推理库为深度学习模型生成代码需要 GPU Coder 许可证。

在“配置参数”对话框中，以下配置参数位于**代码生成 > 接口**窗格中。

参数	描述
Code replacement library	指定代码生成器在为模型生成代码时使用的代码替换库。
Code replacement libraries (Embedded Coder)	指定代码生成器在为模型生成代码时使用的多个代码替换库。
“共享代码位置” (第 12-7 页)	指定生成工具函数、导出的数据类型定义和具有自定义存储类的导出数据声明的位置。
Support: floating-point numbers (Embedded Coder)	指定是否生成浮点数据和运算。
“支持: 非有限数” (第 12-9 页)	指定是否生成非有限数据以及对非有限数据的运算。
Support: complex numbers (Embedded Coder)	指定是否生成复数数据和运算。
Support: absolute time (Embedded Coder)	指定是否为绝对时间值和历时值生成和保留整数计数器。
Support: continuous time (Embedded Coder)	指定是否为使用连续时间的模块生成代码。 当您对配置了服务接口的组件模型使用 Embedded Coder 时，此参数不适用。
“支持: 可变大小信号” (Embedded Coder)	指定是否为使用可变大小信号的模型生成代码。
Code interface packaging	选择生成的代码 C 或 C++ 代码接口的打包。
Multi-instance code error diagnostic	选择在模型违反生成多实例代码要求时显示的诊断严重性级别。
Pass root-level I/O as (Embedded Coder)	控制如何将根级模型输入和输出传递给可重用的 model_step 函数。
Remove error status field in real-time model data structure (Embedded Coder)	指定是记录还是监控错误状态。
Include model types in model class (Embedded Coder)	指定是否在模型类中生成模型类型定义。
“数组布局” (第 12-19 页)	将代码生成的数组数据的布局指定为列优先或行优先
External functions compatibility for row-major code generation	选择在 Simulink 遇到未指定数组布局的函数时的诊断操作
“Generate C API for: signals” (第 12-11 页)	生成具有信号结构体的 C API 数据接口代码。

参数	描述
“Generate C API for: parameters” (第 12-12 页)	生成具有参数调整结构体的 C API 数据接口代码。
Generate C API for: states	生成具有状态结构体的 C API 数据接口代码。
“生成用于根级 I/O 的 C API” (第 12-13 页)	生成具有根级 I/O 结构体的 C API 数据接口代码。
“ASAP2 接口” (第 12-14 页)	为 ASAP2 数据接口生成代码。
“外部模式” (第 12-15 页)	为外部模式数据接口生成代码。当对配置了服务接口的组件模型使用 Embedded Coder 时，此参数不相关，因此不受支持。
Transport layer	指定通信的传输协议。
MEX-file arguments	指定要传递给外部模式接口 MEX 文件的参数，以便与正在执行的目标通信。
Static memory allocation	控制用于外部模式通信的内存缓冲器。
Static memory buffer size	指定用于外部模式通信的内存缓冲区大小。
Automatically allocate static memory	自动为外部模式通信中使用的缓冲区分配静态内存。
Maximum duration	在确定外部模式通信所需的静态内存大小时，以基本速率指定软件必须考虑的最大持续时间。
Target library	指定在代码生成过程中使用的目标深度学习库。 “cuDNN” 或 “TensorRT” 需要 GPU Coder 许可证。
ARM Compute Library version	指定 ARM® Compute Library 的版本。
ARM Compute Library architecture	指定目标硬件支持的 ARM 架构。
“自动调节” (第 12-16 页)	对 cuDNN 库使用自动调整。启用自动调整后，cuDNN 库可找到最快的卷积算法。 此参数要求具有 GPU Coder 许可证。

这些配置参数位于**高级参数**下。

参数	描述
Support non-inlined S-functions (Embedded Coder)	指定是否为非内联的 S-Function 生成代码。 当您对配置了服务接口的组件模型使用 Embedded Coder 时，此参数不适用。
Maximum word length	指定代码生成过程在生成系统定义的多字类型定义时使用的最大字长（以位为单位）。
“动态大小字符串的缓冲区大小（以字节为单位）” (第 12-18 页)	为没有最大长度的动态字符串信号生成的字符缓冲区的字节数。
Multiword type definitions (Embedded Coder)	指定在生成的代码中对多字数据类型使用系统定义的还是用户定义的类型定义。

参数	描述
Classic call interface	指定是否生成与 R2012a 之前创建的模型中 GRT 目标的主程序模块兼容的模型函数调用。 当您对配置了服务接口的组件模型使用 Embedded Coder 时，此参数不适用。
“使用动态内存分配进行模型初始化” (Embedded Coder)	控制生成的代码如何为模型数据分配内存。
Single output/update function	指定是否生成 <code>model_step</code> 函数。 当您对配置了服务接口的组件模型使用 Embedded Coder 时，此参数不适用。
“需要终止函数” (Embedded Coder)	指定是否生成 <code>model_terminate</code> 函数。
Combine signal/state structures (Embedded Coder)	指定是否在生成的代码中将全局模块信号和全局状态数据合并到一个数据结构体中 当您对配置了服务接口的组件模型使用 Embedded Coder 时，将无法清除此参数。
Generate separate internal data per entry-point function (Embedded Coder)	将模型中以相同速率工作的模块信号（模块 I/O）和离散状态 (DWork) 生成为相同的数据结构。
“MAT 文件记录” (第 12-21 页)	指定 MAT 文件日志记录。 当您对配置了服务接口的组件模型使用 Embedded Coder 时，此参数不适用。
“MAT 文件变量名称修饰符” (Embedded Coder)	选择要添加到 MAT 文件变量名称的文本。
Existing shared code (Embedded Coder)	指定包含现有共享代码的文件夹
Remove disable function (Embedded Coder)	从包含模型引用层次结构的基于 ERT 的系统的生成代码中删除 <code>disable</code> 函数的不可达（死代码）实例。
Remove reset function (Embedded Coder)	从包含模型引用层次结构的基于 ERT 的系统的生成代码中删除 <code>reset</code> 函数的不可达（死代码）实例。
“等间距设定的 LUT 对象结构体顺序” (第 12-17 页)	更改为查找表对象（其设定参数设置为显式值）生成的结构体中字段的顺序。
LUT object struct order for explicit value specification	更改为查找表对象（其设定参数设置为显式值）生成的结构体中字段的顺序。
Generate destructor (Embedded Coder)	指定是否为 C++ 模型类生成析构函数。
Use dynamic memory allocation for model block instantiation (Embedded Coder)	指定在模型对象注册期间，生成的代码是否使用运算符 <code>new</code> 为配置有 C++ 类接口的引用模型实例化对象。
Ignore custom storage classes (Embedded Coder)	指定是应用还是忽略自定义存储类。
Ignore test point signals (Embedded Coder)	指定测试点的内存缓冲区分配。
Implement each data store block as a unique access point (Embedded Coder)	为 Data Store Memory 模块的每次读取/写入操作创建唯一变量。
Generate full file banner	生成完整的头文件前注，包括时间戳。

高级参数下的以下参数不常使用，没有其他文档。

参数	描述
GenerateSharedConstants	控制代码生成器是否生成具有共享常量和共享函数的代码。默认值是 on 。当设置为 off 时，代码生成器不生成共享常量。
InferredTypesCompatibility	为了与包括 tmwtypes.h 在内的原有代码兼容，请指定代码生成器在 model.h 内创建预处理器指令 #define __TMWTYPES__ 。
TargetLibSuffix character vector - "	控制用于命名目标的依存库的后缀（例如， _target.lib 或 _target.a ）。如果指定，字符向量必须包含句点（.）。（对于生成的模型引用库，库的后缀在 Windows 系统上默认为 _rtwlib.lib ，在 UNIX® 系统上默认为 _rtwlib.a 。） 此参数不适用于使用工具链方法的模型编译，请参阅“Library Control Parameters”
TargetPreCompLibLocation character vector - "	控制预编译库的位置。如果不设置此参数，代码生成器将使用 rtwmakecfg.m 中指定的位置。
IsERTTarget	指示当前选择的目标是否派生自 ERT 目标。
CPPClassGenCompliant	指示目标是否支持生成和配置与模型代码交互的 C++ 类接口。
ConcurrentExecutionCompliant	指示目标是否支持并发执行
UseToolchainInfoCompliant	指示自定义目标符合工具链。
ModelStepFunctionPrototypeControlCompliant	指示目标是否支持控制为 Simulink 模型生成的初始化和单步函数的函数原型。 当您对配置了服务接口的组件模型使用 Embedded Coder 时，此参数不适用。
ParMdlRefBuildCompliant	指示在编译包含引用模型的模型时，模型是否配置为并行编译。
CompOptLevelCompliant off, on	在 SelectCallback 中为目标设置，以指示目标是否支持使用 编译器优化级别 参数控制用于编译所生成代码的编译器优化级别。 对于自定义目标，默认值为 off ，对于 Simulink Coder 和 Embedded Coder 产品提供的目标，默认值为 on 。
ModelReferenceCompliant 字符向量 - off、on	在 SelectCallback 中为目标设置，以指示目标是否支持模型引用。

以下参数仅供 MathWorks 使用。

参数	描述
ExtModeTesting	仅供 MathWorks 使用。
ExtModeIntrflLevel	仅供 MathWorks 使用。

参数	描述
ExtModeMexFile	仅供 MathWorks 使用。

另请参阅

详细信息

- “Model Configuration Set Customization”

共享代码位置

描述

指定生成工具函数、导出的数据类型定义和具有自定义存储类的导出数据声明的位置。

类别：代码生成 > 接口

设置

默认值：“自动”

“自动”

代码生成器将实用工具代码放在 `codeGenFolder/slprj/target/_sharedutils`（或 `codeGenFolder/targetSpecific/_shared`）文件夹中，用于包含现有共享代码 (Embedded Coder)或至少以下模块之一的模型：

- Model 模块
- Simulink Function 模块
- Function Caller 模块
- 从 Stateflow 或 MATLAB Function 模块对 Simulink 函数的调用
- 选择**导出图级别函数**参数时的 Stateflow 图形函数

如果模型不包含上述模块之一或现有共享代码 (Embedded Coder)，代码生成器会将实用工具代码放在编译文件夹（通常是包含 `model.c` 或 `model.cpp` 的文件夹）中。

“共享位置”

将实用工具的代码放在 `codeGenFolder/slprj/target/_sharedutils`（或 `codeGenFolder/targetSpecific/_shared`）文件夹中。

命令行信息

参数：UtilityFuncGeneration

类型：字符向量

值：'Auto' | 'Shared location'

默认值：'Auto'

推荐的设置

应用场景	设置
调试	“共享位置” (GRT) 无影响 (ERT)
可追溯性	“共享位置” (GRT) 无影响 (ERT)
效率	无影响 (执行, RAM) “共享位置” (ROM)
安全预警	无影响

另请参阅

相关示例

- “模型配置参数：代码生成接口”（第 12-2 页）
- “运行时环境配置”
- “管理编译过程文件夹”
- “共享实用工具代码”

支持: 非有限数

描述

指定是否生成非有限数据以及对非有限数据的运算。

类别: 代码生成 > 接口

设置

默认值: on

☒ On
生成非有限数据 (例如, NaN 和 Inf) 和相关运算。

☐ Off
不生成非有限数据和运算。如果清除此选项, 则当代码生成器遇到非有限数据或表达式时会出错。错误消息报告有问题的模块和参数。

注意 代码生成是在假设不存在非有限数据的情况下优化的。然而, 如果您的应用程序通过信号数据或 MATLAB 代码生成非有限数, 则在处理非有限数据时, 生成代码的行为可能与仿真结果不一致。

依存关系

- 对于基于 ERT 的目标, 参数**支持: 浮点数**启用**支持: 非有限数**。
- 如果对于顶层模型为 “off”, 则对于引用模型也必须为 “off” 。
- 选择参数 **MAT 文件记录**时, 还必须选择**支持: 非有限数**, 如果使用基于 ERT 的系统目标文件, 还要选择**支持: 浮点数**。

命令行信息

参数: SupportNonFinite
类型: 字符向量
值: 'on' | 'off'
默认值: 'on'

推荐的设置

应用场景	设置
调试	无影响
可追溯性	无影响
效率	Off (执行, ROM) , 无影响 (RAM)
安全预警	无建议

另请参阅

相关示例

- “模型配置参数：代码生成接口”（第 12-2 页）

Generate C API for: signals

描述

生成具有信号结构体的 C API 数据接口代码。

类别: Code Generation > Interface

设置

默认值: off

☒ On
生成用于全局模块输出的 C API 接口。

☐ Off
不生成 C API 信号。

命令行信息

参数: RTWCAPISignals

类型: 字符向量

值: 'on' | 'off'

默认值: 'off'

推荐的设置

应用场景	设置
调试	无影响
可追溯性	无影响
效率	无影响
安全预警	在开发期间无影响 对于生产代码生成, 为 Off

另请参阅

相关示例

- “模型配置参数: 代码生成接口” (第 12-2 页)
- “使用 C API 在生成的代码和外部代码之间交换数据”

Generate C API for: parameters

描述

生成具有参数调整结构体的 C API 数据接口代码。

类别：Code Generation > Interface

设置

默认值：off

☒ On
生成用于全局模块参数的 C API 接口。

☐ Off
不生成 C API 参数。

命令行信息

参数：RTWCAPIParams

类型：字符向量

值：'on' | 'off'

默认值：'off'

推荐的设置

应用场景	设置
调试	无影响
可追溯性	无影响
效率	无影响
安全预警	在开发期间无影响 对于生产代码生成，为 Off

另请参阅

相关示例

- “模型配置参数：代码生成接口”（第 12-2 页）
- “使用 C API 在生成的代码和外部代码之间交换数据”

生成用于根级 I/O 的 C API

描述

生成具有根级 I/O 结构体的 C API 数据接口代码。

类别：代码生成 > 接口

设置

默认值：off

- ☒ On
生成根级输入和输出的 C API 接口。
- ☐ Off
不生成根级输入和输出的 C API 接口。

命令行信息

参数：RTWCAPIRootIO
类型：字符向量
值：'on' | 'off'
默认值：'off'

推荐的设置

应用场景	设置
调试	无影响
可追溯性	无影响
效率	无影响
安全预警	在开发期间无影响 对于生产代码生成，为 Off

另请参阅

相关示例

- “模型配置参数：代码生成接口”（第 12-2 页）
- “使用 C API 在生成的代码和外部代码之间交换数据”

ASAP2 接口

兼容性注意事项

ASAP2 接口在配置窗格中不再可用，只能通过使用命令行参数来启用。要生成 a2l 文件，请参阅 “生成 ASAP2 和 CDF 标定文件”。

描述

为 ASAP2 数据接口生成代码。

类别：Code Generation > Interface

设置

默认值：off

☒ On
为 ASAP2 数据接口生成代码。

☐ Off
不为 ASAP2 数据接口生成代码。

命令行信息

参数：GenerateASAP2

类型：字符向量

值：'on' | 'off'

默认值：'off'

推荐的设置

应用场景	设置
调试	无影响
可追溯性	无影响
效率	无影响
安全预警	在开发期间无影响 对于生产代码生成，为 Off

另请参阅

相关示例

- “模型配置参数：代码生成接口”（第 12-2 页）
- “导出 ASAP2 文件用于数据测量和标定”

外部模式

描述

为外部模式数据接口生成代码。

类别：Code Generation > Interface

设置

默认值： off



On

为外部模式数据接口生成代码。



Off

不为外部模式数据接口生成代码。

命令行信息

参数： ExtMode

类型： 字符向量

值： 'on' | 'off'

默认值： 'off'

推荐的设置

应用场景	设置
调试	无影响
可追溯性	无影响
效率	无影响
安全预警	在开发期间无影响 对于生产代码生成，为 Off

另请参阅

相关示例

- “模型配置参数：代码生成接口” （第 12-2 页）
- “用于参数调节、信号监控和代码执行探查的外部模式仿真”
- “External Mode Simulation by Using XCP Communication”
- “使用 TCP/IP 或串行通信的外部模式仿真”

自动调节

描述

对 cuDNN 库使用自动调整。

类别：代码生成 > 接口

设置

默认值：On



On

使用此选项启用自动调节功能。启用自动调整后，cuDNN 库可找到最快的卷积算法。这会提高 SegNet 和 ResNet 等大型网络的性能。



Off

禁用 cuDNN 库的自动调节。

依存关系

- 此参数要求具有 GPU Coder 许可证。
- 要启用此参数，请在**语言**下选择 “C++”，在**目标库**下选择 “cuDNN”。

命令行信息

参数：DLAutoTuning

值：'on' | 'off'

默认值：'on'

另请参阅

相关示例

- “模型配置参数：代码生成接口”（第 12-2 页）
- “GPU Code Generation for Deep Learning Networks Using MATLAB Function Block”（GPU Coder）

等间距设定的 LUT 对象结构体顺序

描述

更改为查找表对象（其设定参数设置为显式值）生成的结构体中字段的顺序。

类别：Code Generation > Interface > Advanced Parameters

设置

默认值： "Size,Breakpoints,Table"

"Size,Breakpoints,Table"

按大小、断点、表的顺序显示结构体。

"Size,Table,Breakpoints"

按大小、表、断点的顺序显示结构体。

命令行信息

参数： LUTObjectStructOrderEvenSpacing

类型： 字符向量

值： 'Size,Breakpoints,Table' | 'Size,Table,Breakpoints'

默认值： 'Size,Breakpoints,Table'

推荐的设置

应用场景	设置
调试	无影响
可追溯性	无影响
效率	无影响
安全预警	无影响

动态大小字符串的缓冲区大小（以字节为单位）

描述

为没有最大长度的动态字符串信号生成的字符缓冲区的字节数。

类别：Code Generation > Interface

设置

默认值：256

命令行信息

参数：DynamicStringBufferSize

类型：字符向量

值：标量

默认值：256

推荐的设置

应用场景	设置
调试	无影响
可追溯性	无影响
效率	无影响
安全预警	无影响

数组布局

描述

将代码生成的数组数据的布局指定为列优先或行优先。

类别：代码生成 > 接口

设置

默认值：“列优先”

“列优先”

在列优先数组布局中生成代码。以如下 4×3 矩阵 A 为例：

```
A =
  1  2  3
  4  5  6
  7  8  9
 10 11 12
```

在列优先数组布局中，列元素在内存中是连续的。A 在生成的代码中表示为：

```
1  4  7 10  2  5  8 11  3  6  9 12
```

“行优先”

在行优先数组布局中生成代码。例如，对于矩阵 A，在行优先数组布局中，行元素是连续的。A 在生成的代码中表示为：

```
1  2  3  4  5  6  7  8  9 10 11 12
```

选择参数使用针对行优先数组布局优化的算法以启用高效的行优先算法。

命令行信息

参数：ArrayLayout

类型：字符向量

值：'Column-major' | 'Row-major'

默认值：'Column-major'

推荐的设置

应用场景	设置
调试	无影响
可追溯性	无影响
效率	无影响
安全预警	无影响

另请参阅

相关示例

- “模型配置参数：代码生成接口”（第 12-2 页）
- “Code Generation of Matrices and Arrays”

MAT 文件记录

描述

指定 MAT 文件日志记录。当对配置了服务接口的组件模型使用 Embedded Coder 时，此参数不相关，因此不受支持。

类别：代码生成 > 接口

设置

默认值：对于 GRT 目标为 on，对于基于 ERT 的目标为 off



启用 MAT 文件记录。选择此选项时，生成的代码将以下列方式之一将指定的仿真数据保存到 MAT 文件中：

- **配置参数 > 数据导入/导出**（请参阅“模型配置参数：数据导入/导出”）
- To Workspace 模块
- To File 模块
- 启用了**将数据记录到工作区**参数的 Scope 模块

在仿真中，这些数据将写入 MATLAB 工作区，如“导出仿真数据”和“Configure Signal Data for Logging”中所述。设置 MAT 文件记录会将数据重定向到 MAT 文件。该文件名为 **model.mat**，其中 **model** 是您的模型的名称。



禁用 MAT 文件记录。清除此选项有以下好处：

- 消除与支持文件系统相关联的开销，嵌入式应用程序通常不要求支持文件系统
- 消除初始化、更新和清理日志变量所需的额外代码和内存使用量
- 在某些情况下，可消除与根输出端口相关联的代码和存储
- 无需在 **model_step** 中的当前时间和停止时间之间进行比较，从而允许生成的程序无限期运行，而不管停止时间如何设置

依存关系

- 选择 **MAT 文件记录**时，还必须选择配置参数**支持：非有限数**，如果使用基于 ERT 的系统目标文件，还要选择 **支持：浮点数**。
- 选择此选项将启用 **MAT 文件变量名称修饰符**。
- 对于基于 ERT 的系统目标文件，如果使用导出的函数调用，请清除此参数。

限制

- 代码生成器不支持自定义数据类型（非 Simulink 内置的数据类型）的 MAT 文件记录。
- MAT 文件记录不支持作用域为文件的数据，例如，应用内置存储类 **FileScope** 的数据项。

- 在引用模型中，仅支持以下数据日志记录功能：
 - To File 模块
 - 状态日志记录 - 软件将数据存储顶层模型的 MAT 文件中。
- 在 Embedded Coder 产品的上下文中，MAT 文件记录不支持以下 IDE：Analog Devices® VisualDSP++®、Texas Instruments™ Code Composer Studio™、Wind River® DIAB/GCC。
- MAT 文件记录不支持应用了会在生成代码中产生不可寻址数据的 **ImportedExternPointer** 存储类或存储类的 Outport 模块。例如，存储类 GetSet 会导致 Outport 作为函数调用出现在生成的代码中，而函数调用是不可寻址的。无论您是通过使用模型数据编辑器等直接应用存储类，还是通过将 Outport 解析为使用存储类的 Simulink.Signal 对象来应用存储类，此限制都适用。作为解决办法，请将存储类应用于进入 Outport 模块的信号。

命令行信息

参数: MatFileLogging
类型: 字符向量
值: 'on' | 'off'
默认值: 对于 GRT 目标为 'on', 对于基于 ERT 的目标为 'off'

推荐的设置

应用场景	设置
调试	On
可追溯性	无影响
效率	Off
安全预警	Off

另请参阅

相关示例

- “模型配置参数：代码生成接口” （第 12-2 页）
- “Log Program Execution Results”
- “Log Data for Analysis”
- “Virtualized Output Ports Optimization” (Embedded Coder)

代码生成参数: GPU 代码

计算能力

描述

指定为其生成 CUDA 代码的 GPU 设备的最小计算能力。

类别: 代码生成 > GPU 代码

设置

默认值: "3.5"

选择代码生成的最小计算能力。计算能力确定 GPU 硬件所支持的功能。应用程序在运行时使用它来确定 GPU 设备上可用的硬件功能和指令。如果您指定自定义计算能力, GPU Coder 将忽略此设置。

要查看代码生成的 CUDA 计算能力要求, 请参考下表。

目标	计算能力
CUDA MEX	请参阅 "GPU Computing Requirements" (Parallel Computing Toolbox)。
源代码、静态或动态库以及可执行文件	3.2 或更高版本。
8 位整数精度的深度学习应用程序	6.1、6.3 或更高版本。
半精度 (16 位浮点) 的深度学习应用程序	5.3、6.0、6.2 或更高版本。

依存关系

- 此参数要求具有 GPU Coder 许可证。
- 要启用此参数, 请在**代码生成**窗格下选择**生成 GPU 代码**。

命令行信息

参数: GPUComputeCapability

类型: 字符向量

值: '3.2' | '3.5' | '3.7' | '5.0' | '5.2' | '5.3' | '6.0' | '6.1' | '6.2' | '7.0' | '7.1' | '7.2' | '7.5' |

默认值: '3.5'

另请参阅

相关示例

- "Model Configuration Parameters: GPU Code"
- "Code Generation from Simulink Models with GPU Coder" (GPU Coder)

Simulink Coder 参数：高级参数

Simulink 模型的配置参数

模型顾问检查

用于创建受保护模型的参数

Simulink Coder 工具

代码映射编辑器 - C

将模型元素与代码接口定义相关联


说明

代码映射编辑器采用图形界面，您可以在其中为代码生成配置模型中的数据元素。模型引用层次结构中的每个模型都有自己的代码映射。将模型数据元素的每个类别与整个模型中的特定存储类相关联。然后，根据需要为特定数据元素覆盖这些设置。

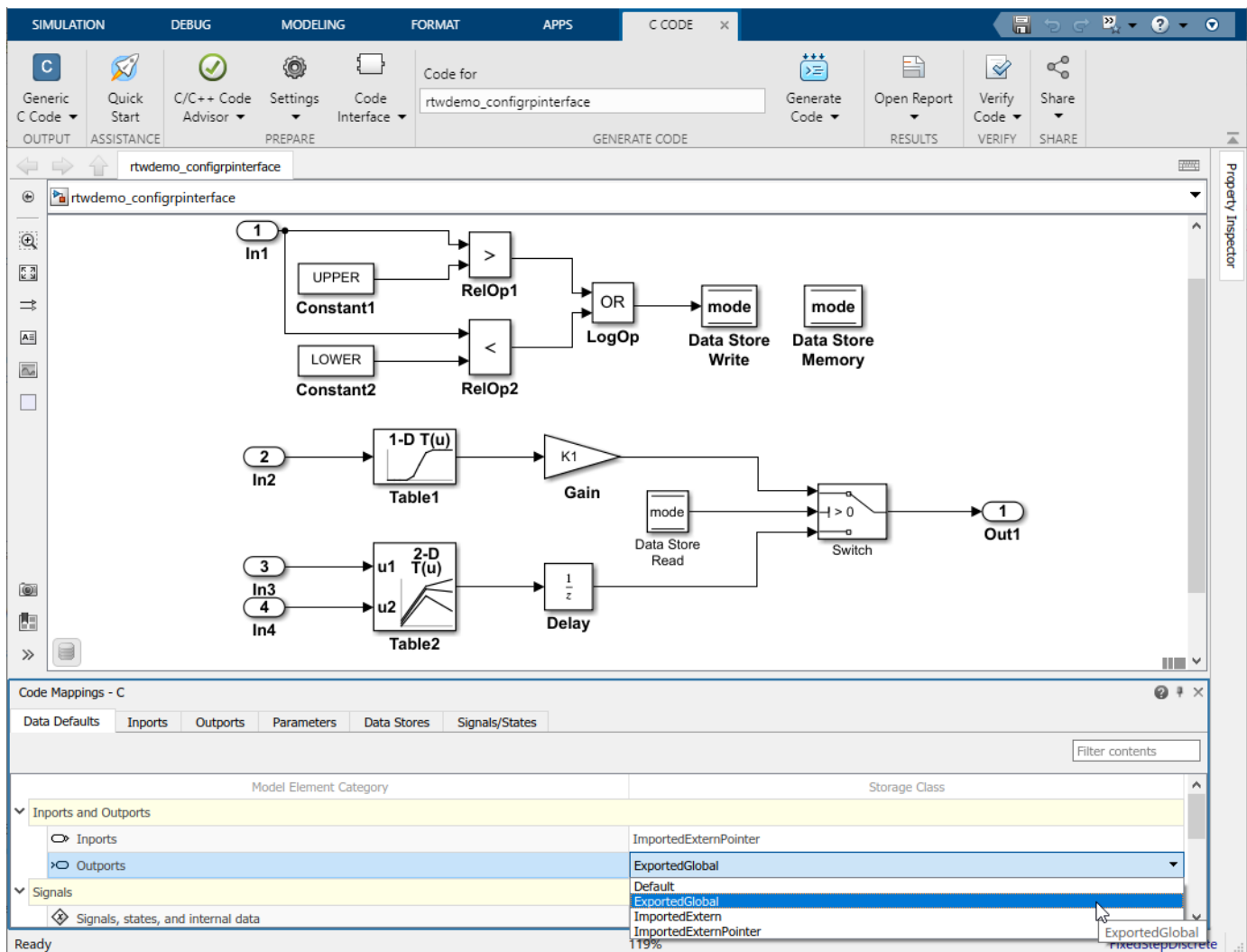
存储类定义代码生成器在为关联的数据生成代码时使用的属性，如形式和位置。

要为代码生成配置数据元素和函数，请使用代码映射编辑器显示中的选项卡：

- **数据默认值**
- **输入端口**
- **输出端口**
- **参数**
- **数据存储**
- **信号/状态**

当存在可以为模型元素配置的其他代码映射属性时，您可以通过在活动表中选择一行并点击  图标来配置这些属性。

您需要将信号添加到模型代码映射中，才能配置该信号以进行代码生成。通过将鼠标悬停在信号线上方或下方出现的省略号上以打开操作栏，在代码映射中添加和删除信号。点击**添加信号**或**删除信号**按钮。在代码映射编辑器中的**信号/状态**选项卡上也可以找到这些按钮。



打开 代码映射编辑器 - C

执行以下操作之一：

- 打开 Simulink Coder。在 **C 代码** 选项卡上，选择 **代码接口** > **默认代码映射** 或 **代码接口** > **个体元素代码映射**。
- 打开 Simulink Coder。在 Simulink 编辑器窗口左下角的 **C 代码** 选项卡上，点击 **代码映射** 选项卡。
- 在 Simulink 编辑器窗口的模型画布中，点击右下角的透视控件并选择 **代码**。然后，点击 **代码映射** 选项卡。

示例

为根级 Inport 和 Outport 模块配置代码生成

为整个模型中的根级 Inport 和 Outport 模块配置代码生成。应用默认配置可以节省时间，尤其是对于使用大量数据的大型模型。应用默认映射后，您可以调整单个数据元素的映射。

设置示例环境

- 1 将外部代码文件 `exDblFloat.h` 复制到一个可写文件夹中。
- 2 打开模型 `rtwdemo_configrpinterface`。将模型副本保存到与文件 `exDblFloat.h` 相同的位置。
- 3 打开 Simulink Coder。C 代码选项卡包括代码映射编辑器。

配置默认映射

配置代码生成器，以在生成的文件 `rtwdemo_configrpinterface.h` 和 `rtwdemo_configrpinterface.c` 中声明和定义输入端口与输出端口的全局变量。

- 1 在 C 代码选项卡中，选择代码接口 > 默认代码映射。
- 2 在数据默认值选项卡的输入端口和输出端口下，选择输入端口行。然后，将存储类设置为 “ImportedExternPointer”。将输出端口的存储类设置为 “ExportedGlobal”。编辑器会更新两个所选数据元素类别的默认存储类设置。

The screenshot displays the Simulink Coder environment. The top toolbar includes tabs for SIMULATION, DEBUG, MODELING, FORMAT, APPS, and C CODE. The C CODE tab is active, showing a 'Code for' field with the text 'rtwdemo_configrpinterface'. Below the toolbar, the 'rtwdemo_configrpinterface' model is visible, featuring inputs In1, In2, In3, and In4, and an output Out1. The model contains various blocks such as 'UPPER Constant1', 'LOWER Constant2', 'RelOp1', 'RelOp2', 'OR LogOp', 'mode Data Store Write', 'mode Data Store Memory', '1-D T(u) Table1', 'K1 Gain', '2-D T(u) Table2', 'u1', 'u2', 'Delay', and 'Switch'.

At the bottom, the 'Code Mappings - C' window is open, showing the 'Data Defaults' tab. This tab contains a table with two columns: 'Model Element Category' and 'Storage Class'. The 'Inports and Outputs' category is expanded, showing 'Inports' and 'Outputs'. The 'Outputs' row is selected, and the 'Storage Class' dropdown menu is open, displaying options: 'Default', 'ExportedGlobal', 'ImportedExtern', and 'ImportedExternPointer'. The 'ExportedGlobal' option is selected.

Model Element Category	Storage Class
Inports and Outputs	
Inports	ImportedExternPointer
Outputs	ExportedGlobal
Signals	
Signals, states, and internal data	



为默认配置配置单个输入端口和输出端口

- 1 在代码映射编辑器中，点击**输入端口**选项卡。每个输入端口的存储类设置为 **Auto**，这意味着代码生成器可能出于优化目的而消除或更改相关代码的表示。如果无法进行优化，代码生成器将应用模型默认配置。
- 2 强制代码生成器使用输入端口的默认配置，即存储类 **ImportedExternPointer**。按 **Ctrl** 键并选择输入端口。对于其中一个所选输入端口，将存储类设置为“模型默认:ImportedExternPointer”。编辑器会更新所选输入端口的存储类设置。
- 3 强制代码生成器对模型根输出端口使用存储类 **ExportedGlobal**。点击**输出端口**选项卡。选择 **Out2** 的行。然后，将存储类设置为“模型默认值:ExportedGlobal”。

配置单个数据元素

要配置单个数据元素的属性（例如，当您需覆盖默认配置设置时），请使用不同数据元素类型的选项卡。对于此示例，我们将覆盖 Inport 模块 In1 的默认存储类设置。

默认情况下，代码生成器根据模型中的 Inport 或 Outport 模块名称来命名输入端口和输出端口变量。当使用 **Auto** 以外的存储类设置配置数据元素时，可以通过设置存储类属性**标识符**来覆盖单个元素的默认设置。此属性使您能够在不修改模型设计的情况下为代码指定标识符。在此示例中，为 Inport 和 Outport 模块设置**标识符**。

- 1 在代码映射编辑器中，点击**输入端口**选项卡。
- 2 对于 In1，将存储类设置为 “ImportedExtern”。
- 3 对于每个输入端口，选择对应的行。然后，点击  图标。按如下方式设置**标识符**属性：
 - 将 In1 设置为 input1。
 - 将 In2 设置为 input2。
 - 将 In3 设置为 input3。
 - 将 In4 设置为 input4。
- 4 点击**输出端口**。
- 5 选择输出端口 Out1。点击  图标，并将**标识符**属性设置为 output。

生成和验证代码

生成代码，并验证为 Inport 和 Output 模块生成的代码是否如预期所示。例如：

- **rtwdemo_configrpinterface_private.h** 包含以下声明：

```
/* Exported data declaration */

/* Data with Imported storage */
extern real_T input1;      /* '<Root>/In1' */
extern real_T input2;      /* '<Root>/In2' */
extern real_T input3;      /* '<Root>/In3' */
extern real_T input4;      /* '<Root>/In4' */
```

- **rtwdemo_configrpinterface.h** 将 **output** 列为输出结构体 **ExtY_rtwdemo_configrpinterfac_T** 中的字段。

```
/* External outputs (root outputs fed by signals with default storage) */
typedef struct {
    real_T output;          /* '<Root>/Out1' */
} ExtY_rtwdemo_configrpinterfac_T;
```

- 以下代码片段显示表示 `In2`、`input2` 的变量，该变量用于采样率为 1 秒的生成单步入口函数。

```
/* Model step function for TID1 */
void rtwdemo_configrpininterface_step1(void) /* Sample time: [1.0s, 0.0s] */
{
    /* Lookup_n-D: '<Root>/Table1D' incorporates:
     * Inport: '<Root>/In2'
     */
    rtwdemo_configrpininterface_B.Table1D = look1_binlepw(input2,
rtwdemo_configrpininterfac_ConstP.Table1D_bp01Data,
rtwdemo_configrpininterfac_ConstP.Table1D_tableData, 10U);
    .
    .
    .
}
```

参数

数据默认值

模型元素类别 — 模型数据元素的类别

字符向量

为某一类别的 Simulink 模型数据元素命名。为一个类别设置的存储类适用于整个模型中属于该类别的元素。

模型元素类别	描述
输入端口	模型的根级输入端口，如 Inport 和 In Bus Element 模块。
输出端口	模型的根级输出端口，如 Outport 和 Out Bus Element 模块。
信号、状态和内部数据	模型内部的数据元素，如模块输出信号、离散模块状态、数据存储和过零信号。
共享局部数据存储	设置了跨模型实例共享模块参数的 Data Store Memory 模块。这些数据存储只在定义它们的模型中可访问。数据存储值在模型的实例之间共享。
全局数据存储	由基础工作区或数据字典中的信号对象定义的数据存储。一个应用程序中的多个模型可以使用这些数据存储。要在代码映射编辑器中查看和配置这些数据存储，请点击类别名称右侧的 Refresh 链接。点击此链接可更新模型图。
模型参数	在模型中定义的参数，例如模型工作区中的参数。不包括模型实参。
外部参数	在基础工作区或数据字典中定义为对象的参数。一个应用程序中的多个模型可以使用这些参数。要在代码映射编辑器中查看和配置这些参数，请点击类别名称右侧的 Refresh 链接。点击此链接可更新模型图。

代码映射编辑器显示给定类别的有效存储类选项。这些选项可以包括：

- 未指定的存储类（“默认”）。代码生成器将数据元素类别的代码放置在标准结构体中，例如 `B_`、`ExtY_`、`ExtU_`、`DW_` 和 `P_`。请参阅 “Data Structures in the Generated Code”。
- 相关的预定义存储类，如 `ExportedGlobal`。
- 可用包中的相关存储类，如 `ImportFromFile`（需要 Embedded Coder）。
- 在 Embedded Coder 字典中定义的存储类（需要 Embedded Coder）。

存储类 — 模型数据元素的代码定义

字符向量

代码生成器用于确定它为模型数据元素生成的代码的属性（如外观和位置）的定义（设定）。有效设置为“Default”、“ExportedGlobal”、“ImportedExtern”和“ImportedExternPointer”。请参阅“Choose Storage Class for Controlling Data Representation in Generated Code”。

输入端口

源 — 根级 Inport 模块或总线元素的名称

字符向量

标识模型中的根 Inport 模块或 In Bus Element 模块的元素（例如 `InBus1.signal1`）。如果元素解析为数据对象，则代码映射编辑器会在源名称的右侧显示“解析为信号对象”图标，并根据元素的存储类设置是否为 **Auto** 来解析配置。如果存储类是 **Auto**，则数据元素采用数据对象指定的代码配置。编辑器将**存储类**列中的显示文本更改为 **From signal object:**，后跟数据对象的存储类名称。如果存储类不是 **Auto**，则数据元素采用您在代码映射编辑器中指定的配置。

存储类 — 根输入端口的代码定义

字符向量

代码生成器用于确定它为根输入端口生成的代码的属性（例如外观和位置）的定义。有效设置为“Auto”、“Model default”、“ExportedGlobal”、“ImportedExtern”和“ImportedExternPointer”。请参阅“Choose Storage Class for Controlling Data Representation in Generated Code”。

标识符 — 变量的名称

字符向量

在生成的代码中表示输入端口的变量的名称。

CalibrationAccess — 启用或禁用输入端口的标定

NoCalibration（默认） | Calibration

为输入端口选择 **Calibration** 以启用标定。选择 **NoCalibration** 以查看输入端口的值并禁用标定。

CompuMethod — 转换方法的名称

字符向量

为便于阅读而将 ECU 内部值转换为物理值的方法的名称。

DisplayIdentifier — 输入端口的显示名称

字符向量

标定工具中用于测量目的的输入端口的可选显示名称，它不同于 Simulink 模型中的输入端口名称。

格式 — 输入端口值的显示格式

%[length].[layout]

为标定工具中的测量指定的特殊显示格式。此格式设定优先于在输入端口的 **CompuMethod** 中指定的显示格式。

输出端口

源 — 根 Outport 模块或总线元素的名称

字符向量

标识模型中的根级 Outport 模块或 Out Bus Element 模块的元素（例如 `OutBus1.signal1`）。如果元素解析为数据对象，则代码映射编辑器会在源名称的右侧显示“解析为信号对象”图标，并根据元素的存储

类设置是否为 **Auto** 来解析配置。如果存储类是 **Auto**，则数据元素采用数据对象指定的代码配置。编辑器将**存储类**列中的显示文本更改为 **From signal object:**，后跟数据对象的存储类名称。如果存储类不是 **Auto**，则数据元素采用您在代码映射编辑器中指定的配置。

存储类 — 根输出端口的代码定义
字符向量

代码生成器用于确定它为根输出端口生成的代码的属性（例如外观和位置）的定义。有效设置为 **"Auto"**、**"Model default"**、**"ExportedGlobal"**、**"ImportedExtern"** 和 **"ImportedExternPointer"**。请参阅 **"Choose Storage Class for Controlling Data Representation in Generated Code"**。

标识符 — 变量的名称
字符向量

在生成的代码中表示输出端口的变量的名称。

CalibrationAccess — 启用或禁用输出端口的标定
NoCalibration（默认） | **Calibration**

为输出端口选择 **Calibration** 以启用标定。选择 **NoCalibration** 以查看输出端口的值并禁用标定。

CompuMethod — 转换方法的名称
字符向量

为便于阅读而将 ECU 内部值转换为物理值的方法的名称。

DisplayIdentifier — 输出端口的显示名称
字符向量

标定工具中用于测量目的的输出端口的可选显示名称，它不同于 Simulink 模型中的输出端口名称。

格式 — 输出端口值的显示格式
%[length].[layout]

为标定工具中的测量指定的特殊显示格式。此格式设定优先于在输出端口的 **CompuMethod** 中指定的显示格式。

参数

源 — 模型参数参量、模型参数或外部参数的名称
字符向量

标识模型中的参数。如果元素解析为数据对象，则代码映射编辑器会在源名称的右侧显示“解析为参数对象”图标，并根据元素的存储类设置是否为 **Auto** 来解析配置。如果存储类是 **Auto**，则数据元素采用数据对象指定的代码配置。编辑器将**存储类**列中的显示文本更改为 **From parameter object:**，后跟数据对象的存储类名称。如果存储类不是 **Auto**，则数据元素采用您在代码映射中指定的配置。

下表列出了参数元素的类型。

参数元素的类型	描述
模型参数	在模型中定义的参数，例如模型工作区中的参数。不包括模型实参。

参数元素的类型	描述
外部参数	在基础工作区或数据字典中定义为对象的参数。一个应用程序中的多个模型可以使用这些参数。仅当模型使用这样的元素时，这种参数分组才会出现在编辑器中。要在代码映射编辑器中查看和配置这些参数，请点击类别名称右侧的 Refresh 链接。点击此链接可更新模型图。

存储类 — 参数的代码定义
字符向量

代码生成器用于确定它为参数生成的代码的属性（例如外观和位置）的定义。对于外部参数，在您点击类别名称右侧的 **Refresh** 链接后，编译的存储类（例如，为外部参数配置的存储类）将出现在**存储类**列的右侧。有效设置为 “Auto”、“Model default”、“ExportedGlobal”、“ImportedExtern” 和 “ImportedExternPointer”。请参阅 “Choose Storage Class for Controlling Data Representation in Generated Code”。

标识符 — 变量的名称
字符向量

在生成的代码中表示模型参数或模型参数参量的变量的名称。

CalibrationAccess — 启用或禁用模型参数的标定
NoCalibration（默认） | Calibration

为模型参数选择 **Calibration** 以启用标定。选择 **NoCalibration** 以查看模型参数的值并禁用标定。

CompuMethod — 转换方法的名称
字符向量

为便于阅读而将 ECU 内部值转换为物理值的方法的名称。

DisplayIdentifier — 模型参数的显示名称
字符向量

标定工具中用于测量目的的模型参数的可选显示名称，它不同于 Simulink 模型中的输出端口名称。

格式 — 模型参数值的显示格式
%[length].[layout]

为标定工具中的测量指定的特殊显示格式。此格式设定优先于模型参数的 **CompuMethod** 中指定的显示格式。

数据存储

源 — 局部数据存储、共享局部数据存储或全局数据存储的名称
字符向量

标识模型中的数据存储。如果元素解析为数据对象，则代码映射编辑器会在源名称的右侧显示“解析为信号对象”图标，并根据元素的存储类设置是否为 **Auto** 来解析配置。如果存储类是 **Auto**，则数据元素采用数据对象指定的代码配置。编辑器将**存储类**列中的显示文本更改为 **From signal object:**，后跟数据对象的存储类名称。如果存储类不是 **Auto**，则数据元素采用您在代码映射中指定的配置。

下表列出了数据存储元素的类型。

数据存储元素的类型	描述
局部数据存储	可从模型层次结构（该模型层次结构是你定义该数据存储所在的级别或其下面的级别）的任意位置访问的数据存储。通过在模型工作区中包含一个 Data Store Memory 模块或创建一个信号对象（合成的数据存储），可以在模型中以图形方式定义局部数据存储。
共享局部数据存储	设置了跨模型实例共享模块参数的 Data Store Memory 模块。这些数据存储只在定义它们的模型中可访问。数据存储值在模型的实例之间共享。仅当模型中存在这样的元素时，这种数据存储分组才会出现在编辑器中。
全局数据存储	由基础工作区或数据字典中的信号对象定义的数据存储。一个应用程序中的多个模型可以使用这些数据存储。这些数据存储的代码映射中是不可配置的。点击“刷新”按钮后，它们会以只读状态出现在代码映射编辑器中，以用于查看或记账目的。仅当模型使用这样的元素时，这种数据存储分组才会出现在编辑器中。要在代码映射编辑器中查看和配置这些数据存储，请点击类别名称右侧的 Refresh 链接。点击此链接可更新模型图。

局部和共享局部数据存储的名称以 `block-name: data-store-name` 格式显示。

根据数据存储元素在模型中的表示和配置方式，局部和共享局部数据存储可以解析为模型工作区、基础工作区或数据字典中的信号对象。全局数据存储解析为基础工作区或数据字典中的信号对象。

存储类 — 数据存储的代码定义

字符向量

代码生成器用于确定它为数据存储生成的代码的属性（例如外观和位置）的定义。对于全局数据存储，在您点击类别名称右侧的 Refresh 链接后，编译的存储类（例如，为全局数据存储配置的存储类）将出现在存储类别的右侧。有效设置为 “Auto”、“Model default”、“ExportedGlobal”、“ImportedExtern” 和 “ImportedExternPointer”。请参阅 “Choose Storage Class for Controlling Data Representation in Generated Code”。

路径 — 模型中数据存储的路径

字符向量

链接，点击该链接可以在模型图中高亮显示数据存储。

标识符 — 变量的名称

字符向量

在生成的代码中表示数据存储的变量的名称。

CalibrationAccess — 启用或禁用数据存储的标定

NoCalibration（默认） | Calibration

为数据存储选择 Calibration 以启用标定。选择 NoCalibration 以查看数据存储的值并禁用标定。

CompuMethod — 转换方法的名称

字符向量

为便于阅读而将 ECU 内部值转换为物理值的方法的名称。

DisplayIdentifier — 数据存储的显示名称

字符向量

标定工具中用于测量目的的数据存储的可选显示名称，它不同于 Simulink 模型中的输出端口名称。

格式 — 数据存储值的显示格式

`%[length].[layout]`

为标定工具中的测量指定的特殊显示格式。此格式设定优先于在数据存储的 **CompuMethod** 中指定的显示格式。

信号/状态

源 — 信号或状态的名称

字符向量

标识模型中的信号线或状态。如果元素解析为数据对象，则代码映射编辑器会在源名称的右侧显示“解析为信号对象”图标，并根据元素的存储类设置是否为 **Auto** 来解析配置。如果存储类是 **Auto**，则数据元素采用数据对象指定的代码配置。编辑器将**存储类**列中的显示文本更改为 **From signal object:**，后跟数据对象的存储类名称。如果存储类不是 **Auto**，则数据元素采用您在代码映射编辑器中指定的配置。

代码映射编辑器会：

- 使用数据元素名称列出命名信号和状态
- 使用 **source-block: port-number** 格式列出未命名信号
- 使用 **block-name: state-name** 格式列出在多个模块中使用的状态

要在模型的代码映射编辑器中配置单个信号线，首先必须将该信号添加到映射中。请参阅“Configure Signal Data for C Code Generation”。

存储类 — 信号的代码定义

字符向量

代码生成器用于确定它为信号线或状态生成的代码的属性（例如外观和位置）的定义。有效设置为“**Auto**”、“**Model default**”、“**ExportedGlobal**”、“**ImportedExtern**”和“**ImportedExternPointer**”。请参阅“Choose Storage Class for Controlling Data Representation in Generated Code”。

路径 — 模型中信号线或状态的路径

字符向量

链接，点击该链接可在模型图中高亮显示使用该状态的信号线或模块。

标识符 — 变量的名称

字符向量

在生成的代码中表示该信号或状态的变量的名称。

CalibrationAccess — 启用或禁用信号或状态的标定

NoCalibration（默认） | **Calibration**

为信号或状态选择 **Calibration** 以启用标定。选择 **NoCalibration** 以查看信号或状态的值并禁用标定。

CompuMethod — 转换方法的名称

字符向量

为便于阅读而将 ECU 内部值转换为物理值的方法的名称。

DisplayIdentifier — 信号或状态的显示名称

字符向量

标定工具中用于测量目的的信号或状态的可选显示名称，它不同于 Simulink 模型中的输出端口名称。

格式 — 信号或状态值的显示格式`%[length].[layout]`

为标定工具中的测量指定的特殊显示格式。此格式设定优先于在信号或状态的 **CompuMethod** 中指定的显示格式。

版本历史记录

在 R2020b 中推出

主题

“模型接口元素的 C 代码生成配置”

“Choose Data Configuration Approach”

“Choose Storage Class for Controlling Data Representation in Generated Code”

“为模型入口函数配置生成的 C 函数接口”

优化参数

模型配置参数：代码生成优化

代码生成 > **优化**类别包括用于提高模型仿真速度和生成代码性能的参数。用于改进所生成代码的模型配置参数需要具备 Simulink Coder 或 Embedded Coder。

参数	描述
Default parameter behavior	在生成的代码中，将数值模块参数转换为常量内联值。
Leverage target hardware instruction set extensions	选择指令集，为目标硬件生成 SIMD (Single Instruction, Multiple Data, 又称为单指令、多数据) 代码。
Optimize reductions	为归约运算循环生成单指令多数据 (SIMD) 代码。
Pass reusable subsystem outputs as (Embedded Coder)	指定可重用子系统如何传递输出。
Remove root level I/O zero initialization (Embedded Coder)	指定是否为设置为零的根级输入端口和输出端口生成初始化代码。
Remove internal data zero initialization (Embedded Coder)	指定是否为设置为零的内部工作结构体（如模块状态和模块输出）生成初始化代码。
Level (Embedded Coder)	选择要应用于生成代码的优化级别。
Priority (Embedded Coder)	优化生成的代码，以提高执行效率、降低内存消耗量或在两者之间实现某种平衡。
Specify custom optimizations (Embedded Coder)	选择此参数可在 详细信息 部分选择优化参数，而不是应用优化级别。
Use memcpy for vector assignment	通过用 memcpy 替换 for 循环，优化为向量赋值生成的代码。
Memcpy threshold (bytes)	指定用 memcpy 和 memset 函数调用替换生成代码中向量赋值的 for 循环时应满足的最小数组大小（以字节为单位）。
Enable local block outputs	指定模块信号是局部声明还是全局声明。
Reuse local block outputs	指定 Simulink Coder 软件是否重用信号内存。
Eliminate superfluous local variables (Expression folding)	将模块计算折叠成单个表达式。
Reuse global block outputs (Embedded Coder)	重用模块输出的全局内存。
Reuse output buffers of Model blocks (Embedded Coder)	重用引用模型缓冲区（如果可能）。
Perform in-place updates for Assignment and Bus Assignment blocks (Embedded Coder)	重用 Bus Assignment 和 Assignment 模块的输入和输出变量（如果可能）。
Reuse buffers for Data Store Read and Data Store Write blocks (Embedded Coder)	删除 Data Store Read 和 Data Store Write 模块的临时缓冲区。直接使用 Data Store Memory 模块（如果可能）。
Simplify array indexing (Embedded Coder)	在以循环方式访问数组时替换数组索引中的乘法运算。

参数	描述
Pack Boolean data into bitfields (Embedded Coder)	指定布尔信号是存储为一位位域还是存储为布尔数据类型。
Bitfield declarator type specifier (Embedded Coder)	指定选择配置参数 Pack Boolean data into bitfields (Embedded Coder) 时使用的位域类型。
Reuse buffers of different sizes and dimensions (Embedded Coder)	通过重用缓冲区来存储不同大小和维度的数据，降低内存消耗。
Optimize global data access (Embedded Coder)	选择全局变量优化。
Optimize block operation order in generated code (Embedded Coder)	在生成的代码中对模块运算重新排序，以提高代码执行速度。
Use bitsets for storing state configuration	使用位集来减少存储状态配置变量所需的内存量。
Use bitsets for storing Boolean data	使用位集来减少存储布尔数据所需的内存量。
Maximum stack size (bytes)	为模型指定最大堆栈大小（以字节为单位）。
Loop unrolling threshold	指定为其生成 for 循环的最小信号或参数宽度。
Optimize using the specified minimum and maximum values (Embedded Coder)	使用模型中信号和参数的指定最小值和最大值优化生成的代码。
子系统输出的最大参数个数	设置要分别传递的子系统输出的最大数量。
Inline invariant signals	将不变信号的符号名转换为常量值。
Remove code from floating-point to integer conversions with saturation that maps NaN to zero	删除用于处理 NaN 值的浮点到整数转换结果的代码。
Use memset to initialize floats and doubles to 0.0	指定是否生成将浮点数据显式初始化为 0.0 的代码。
Remove code from floating-point to integer conversions that wraps out-of-range values	删除用于处理超出范围的浮点数到整数转换结果的绕回代码。
Remove Code from Tunable Parameter Expressions That Saturate Out-of-Range Values (Embedded Coder)	删除可调参数的绕回代码。
Remove code that protects against division arithmetic exceptions (Embedded Coder)	指定是否为整数和定点数据生成防止除以零和 INT_MIN/-1 运算的代码。
Buffer for reusable subsystems	通过在可重用子系统边界插入缓冲区来提高可重用性。
禁用不兼容的优化	指定是否禁用与 Simulink 代码检查器不兼容的优化。
Base storage type for automatically created enumerations	为使用激活状态输出创建的枚举设置存储类型和大小。
Use signal labels to guide buffer reuse (Embedded Coder)	对于具有相同标签的信号，代码生成器尝试使用相同的信号内存。
Generate parallel for-loops (Embedded Coder)	指定是否应在生成代码中针对 Matlab Function、Matlab System 和 For Each 模块实现并行 for 循环。
Signal storage reuse	指定重用为存储模块输入和输出信号所分配的内存缓冲区，从而降低实时程序的内存需求

参数	描述
Operator to represent Bitwise and Logical Operator blocks (Embedded Coder)	指定生成的代码是包含按位运算符还是逻辑运算符， 或两者都包含。

另请参阅

相关示例

- “性能”