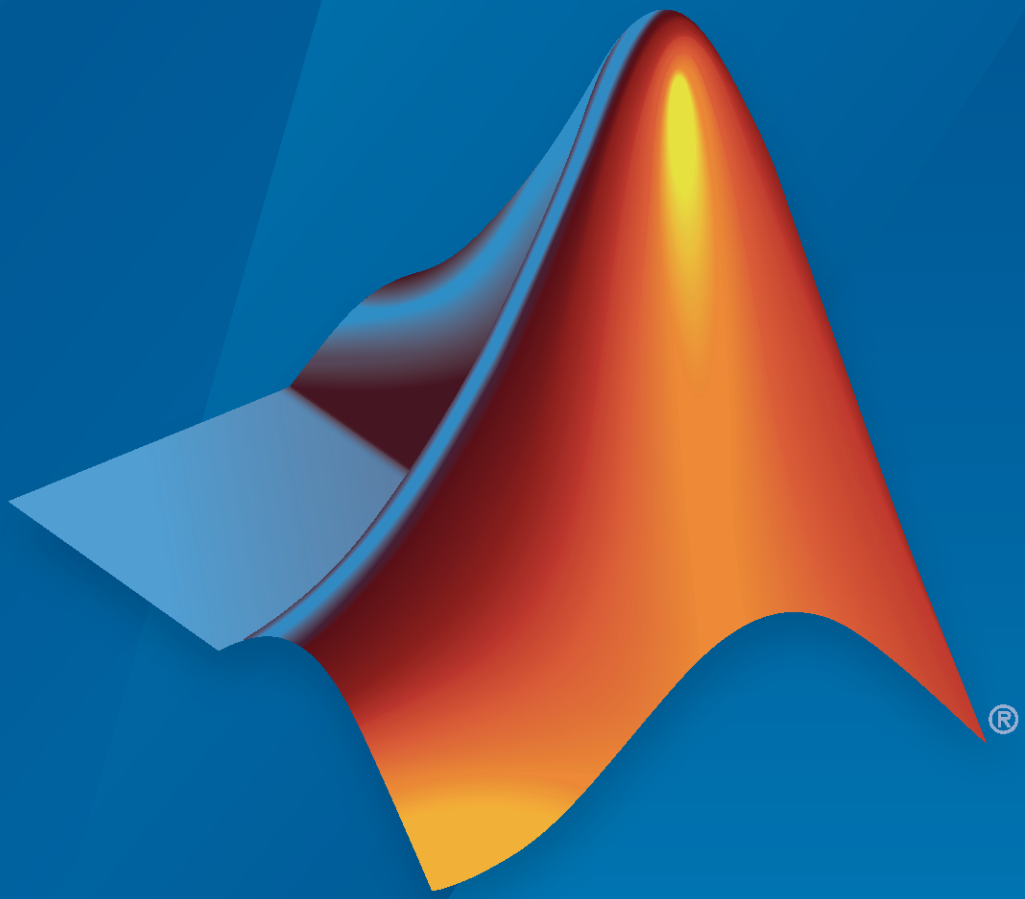
Simulink® Compiler™

Getting Started Guide



MATLAB® & SIMULINK®

R2021b

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Simulink® Compiler™ Getting Started

Trademarks

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.

Patents

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

Revision History

March 2020	Online only	New for Version 1.0 (Release 2020a)
September 2020	Online only	Revised for Version 1.1 (Release 2020b)
March 2021	Online only	Revised for Version 1.2 (Release 2021a)
September 2021	Online only	Revised for Version 1.3 (Release 2021b)

Contents

1 Simulink Compiler Getting Started

Simulink Compiler Getting Started

Simulink Compiler Product Description

Share simulations as standalone executables, web apps, and Functional Mockup Units (FMUs)

Simulink® Compiler™ enables you to share Simulink simulations as standalone executables. You can build the executables by packaging the compiled Simulink model and the MATLAB® code used to set up, run, and analyze a simulation. Standalone executables can be complete simulation apps that use MATLAB graphics and UIs designed with MATLAB App Designer. To cosimulate with an external simulation environment, you can generate standalone Functional Mockup Unit (FMU) binaries that adhere to the Functional Mockup Interface (FMI) standard.

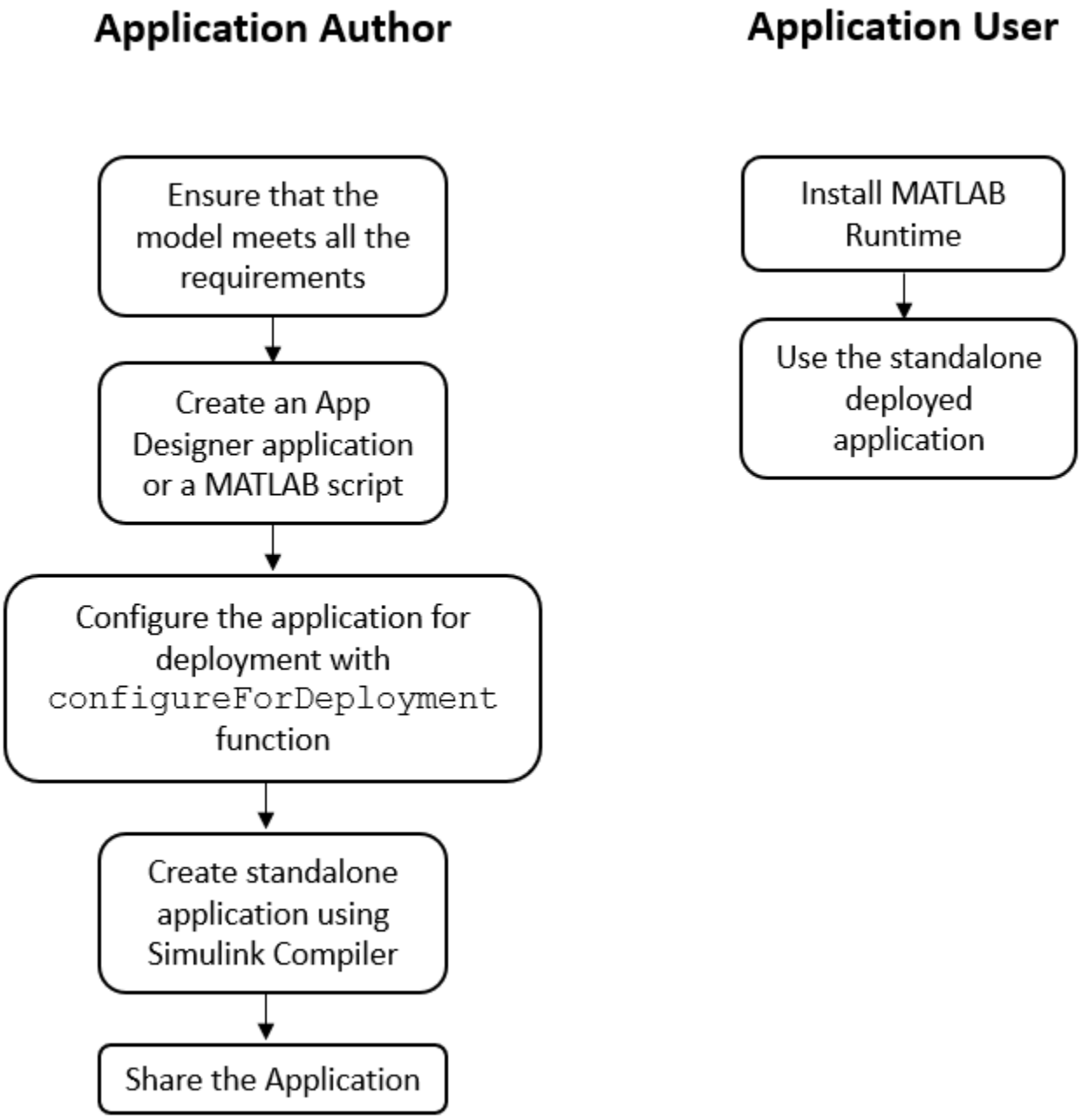
To provide browser-based access to your deployed simulation, you can create a web app and host it with MATLAB Web App Server™. Simulink simulations can be packaged into software components for integration with other programming languages (with MATLAB Compiler SDK™). Large-scale deployment to enterprise systems is supported through MATLAB Production Server™.

To generate C and C++ source code from Simulink, use Simulink Coder™.

Simulink Compiler Workflow Overview

Simulink Compiler lets you share simulations as standalone applications. Simulink Compiler extends the capabilities of MATLAB Compiler to allow Simulink `sim` command and associated Simulink functions in the deployed script or application. For more information about MATLAB Compiler, see MATLAB Compiler documentation.

The application users who use these deployed applications are not expected to interact with the Simulink model directly. The Simulink user provides the application user with a tool that allows them to explore task-specific scenarios without looking at the underlying model that represents the dynamic system. The application users can change model parameters and simulation inputs, and record and analyze simulation outputs.



To develop an application, the Simulink user:

1 Prepares the Simulink model to be compatible with Simulink Compiler, such as checking that the model simulates correctly in rapid accelerator mode. For limitations of rapid accelerator mode and Simulink Compiler, see “Simulink Compiler Limitations”.

Note For information on toolboxes supported by Simulink Compiler, see “Toolboxes Supported by Simulink Compiler” on page 1-6.

2. Creates an application that simulate the model using the `sim` command, in a script or an App designer app.
3. Configures the script or the app for deployment by using `simulink.compiler.configureForDeployment` function. The `simulink.compiler.configureForDeployment` function adapts the model to run in Rapid Accelerator mode.
4. Creates a standalone application using the `mcc` command or the `deploytool` app.
5. Shares the standalone application.

To use the application, the application user:

1. Installs MATLAB Runtime environment for the deployed application.
2. Uses the deployed application.

The following Simulink functions and classes are deployable:

Functions:

- `sim`
- `start_simulink`

Classes:

- `Simulink.SimulationInput` and its method `setVariable`
- `Simulink.SimulationOutput`
- `Simulink.SimulationData.Dataset`

See Also

`simulink.compiler.configureForDeployment` | `simulink.compiler.genapp` | `Simulink.SimulationInput` | `mcc` | `deploytool` | `sim`

Related Examples

- “Create and Deploy a Script with Simulink Compiler” on page 1-8
- “Export Simulink Models to Functional Mock-up Units” on page 1-10
- “Toolboxes Supported by Simulink Compiler” on page 1-6

Toolboxes Supported by Simulink Compiler

Simulink Compiler supports the following toolboxes:

- Aerospace Blockset™.
- Audio Toolbox™.
- Automated Driving Toolbox™.
- AUTOSAR Blockset.
- Communications Toolbox™.
- Computer Vision Toolbox™.
- Control System Toolbox™.
- Deep Learning Toolbox™: All blocks created from *gensim* that support code generation.
- DSP System Toolbox™.
- Fixed-Point Designer™: Fixed-point data point is supported.
- Fuzzy Logic Toolbox™.
- Model Predictive Control Toolbox™.
- Navigation Toolbox™.
- Phased Array System Toolbox™.
- Powertrain Blockset™.
- Robotics System Toolbox™.
- Simscape™.
- Simscape Driveline™.
- Simscape Electrical™.
- Simscape Fluids™.
- Simscape Multibody™.
- Simulink.
- Simulink Control Design™.
- Simulink Design Optimization™.
- Stateflow®.
- Symbolic Math Toolbox™.
- System Identification Toolbox™.
- Vehicle Dynamics Blockset™.
- Vehicle Network Toolbox™.
- Vision HDL Toolbox™.

Note Simulink Compiler supports all the blocks that support code generation in these toolboxes. Among these toolboxes, the prebuilt apps, UIs, and functions, and blocks that don’t support code generation are not supported by Simulink Compiler.

See Also

Related Examples

- “Create and Deploy a Script with Simulink Compiler” on page 1-8
- “Export Simulink Models to Functional Mock-up Units” on page 1-10

Create and Deploy a Script with Simulink Compiler

In this section...

In this example, you prepare a model to work with Simulink Compiler, develop and compile the script, and then deploy it as a standalone application.

Prepare the Model

Simulink Compiler uses rapid accelerator simulation targets to generate an executable to submit a Simulink model. Simulink Compiler only supports models which can run in rapid accelerator mode. To set the simulation mode of the model to rapid accelerator, use the model parameter `'SimulationMode'` with `SimulationInput` object. To enable simulation deployment of the model, your model must be supported by the Rapid Accelerator mode correctly.

Simulink Compiler only supports `sim` function syntax that takes `Simulink.SimulationInput` object and returns `Simulink.SimulationOutput` object.

If callbacks are present in the model, they are called during the build time of the application. However, once the application or the script is deployed, these callbacks are not invoked.

Write the Script to Deploy

After preparing the model, write the script that you would like to deploy. In this example, we use a model and change one of the tunable parameters in the script.

In the MATLAB Editor, create a function `deployedScript`. In this function, create a `Simulink.SimulationInput` object for the model, `sldemo_suspn_3dof` and change the value of `Mb` with the `setVariable` method of the `Simulink.SimulationInput` object. To ensure that the model runs in rapid accelerator mode, set the `SimulationMode` to `Rapid` through the `setModelParameter` method of the `Simulink.SimulationInput` object or use the `simulink.compiler.configureForDeployment` function as shown below.

The variables modified in the simulations can be in the base workspace or in the top model workspace. If your model uses external input variables, those variables must be in the MATLAB workspace before packaging for deployment.

```
function deployedScript()
    in = Simulink.SimulationInput('sldemo_suspn_3dof');
    in = in.setVariable('Mb', 1000);
    in = simulink.compiler.configureForDeployment(in);
    out = sim(in);
end
```

Save the function as a `deployedScript.m`.

Compile Script for Deployment

Before compiling the script that you want to deploy, ensure that the files for the model and script, in this case `sldemo_suspn_3dof` and the `deployedScript.m`, are included on the MATLAB search path. To compile the script, use the `mcc` command with the script name. To learn more about the `mcc` command, see `mcc`.

```
mcc -m deployedScript.m
```

Troubleshooting Tips

Simulink Compiler automatically packages the dependencies in the model and the deployed scripts. If the command `mcc` cannot find a dependency, you might see errors.

- If you see the error "Unable to resolve the name `Simulink.SimulationInput`", check that the model is on the path.
- If the dependent files are located in another directory, attach them by using the flag `-a`. For example, `mcc -m scriptName.m -a myDataFile.dat`.

Run the Deployed Script

Install MATLAB Runtime

To run the deployed executable, you need an appropriate runtime environment. To install the MATLAB Runtime, see https://www.mathworks.com/products/compiler/matlab-runtime.html.

Run the Deployed Application

You can run the deployed application only on the platform that the deployed application was developed on.

Run the deployed application from the Windows command prompt. Running the deployed application from the command prompt enables the application to print diagnostic messages in the command prompt when it encounters errors. These messages can be a helpful tool in troubleshooting the problem.

See Also

`configureForDeployment` | `Simulink.SimulationInput` | `mcc` | `deploytool` | `sim`

Related Examples

- “Deploy an App Designer Simulation with Simulink Compiler”
- “Deploy Simulations with Tunable Parameters”

Export Simulink Models to Functional Mock-up Units

Export Models

Export Simulink models to Functional Mock-up Unit (FMU) that supports Co-Simulation in FMI version 2.0. To check that the exported block is still a valid Simulink model, you can also direct the software to import the FMU back to a Simulink model as part of the export process.

Requirements include:

- Simulink Compiler
- A writable folder into which to place the exported FMU.

Exported models can have:

- Input and output data types: double, int32, boolean, string
- Matrices
- Bus Signals
- Tunable Parameter. Tunable parameter can be model arguments, base workspace, or data dictionary variables.
- Unit and description.

Standalone FMU

Simulink models can be exported to Standalone Co-Simulation FMU in version 2.0. The generated FMU package contains following files:

- `modelDescription.xml`
- `model.png (optional)`
- `binaries\win64\modelname.dll`, or `binaries\linux64\modelname.so`, or `binaries\darwin64\modelname.dylib`

You might experience an expected time delay in the exported FMU for Co-Simulation mode.

FMU Variables

FMU modelDescription.xml file contains interfacing variables converted from Simulink model:

- Variables with causality='input': converted from root Inport block
- Variables with causality='output': converted from root Outport block
- Variables with causality='parameter': converted from referenced Runtime Tunable Parameters
- Independent variable 'time'

To generate FMU input and output, define root Inport and Outport block in Simulink model. Name of the generated variable is converted from root Inport/Outport block name, by removing special and blank characters and avoiding duplicates. If input/output signal carries unit information, it is exported as `Unit` attribute of the FMU variable. If the input/output block has a non-empty description information under **Block Properties** > **General** , it is exported as `Description` attribute of the FMU variable.

The following input/output data types are supported:

- double (Real in FMI)
- int32 (Integer in FMI)
- boolean (Boolean in FMI)
- string (String in FMI)

If model root Inport or Outport block is a non-virtual bus, individual bus elements will be expanded to variables using structured naming convention (‘.’). If model root Inport or Outport block, individual scalar elements will be expanded to variables using array naming convention (‘[]’).

To export referenced variables as FMU parameter, you can

- define a variable.
- define a Simulink Parameter object.

Ensure that the variable and the parameter object is directly references by tunable parameters of Simulink blocks. In FMU Export dialog, expand **Parameter Details...** to configure each parameter. You can:

- Unselect **Exported** option to hide a parameter
- Modify **Exported Name** so the parameter is displayed with a different name on FMU interface. Do not use special characters and duplicate names.
- Set `Unit` and `Description` of FMU parameter variable by clicking on parameter name, and directly modifying the parameter object. You cannot configure FMU `Unit` and `Description` using MATLAB Window.

The following parameter data types are supported:

- double (Real in FMI)
- int32 (Integer in FMI)
- boolean or logical (Boolean in FMI)

If referenced parameter is a struct, individual struct members will be expanded to variables using structured naming convention (‘.’). If referenced parameter is array or matrix, individual scalar elements will be expanded to variables using array naming convention (‘[]’).

FMU Solver

Fixed-step solvers are supported for standalone FMU export. It is recommended to set a fixed fundamental sample time (**Solver > Solver details > Fixed-step size**) before exporting the model. When simulating the Standalone FMU in other environment, communication step-size must be an integral multiple of the fundamental sample time.

FMU Dynamic Library

Generated FMU contains dynamic library build for the current platform. Default fmi2TypesPlatform value is used.

All required and optional fmi2 functions defined by FMI standard can be invoked. However, the following functions have no operation and return fmi2OK immediately:

- Model-Exchange functions

- Functions accessing or serializing FMUstate
- Functions setting or getting input or output derivatives
- Functions querying fmi2DoStep status, or cancelling fmi2DoStep
- Function computing directional derivative of variables

Save Source Code with FMU Export

You can export a Simulink model to FMU along with C source code. You can check **Save Source Code** in the **Export Model to FMU Co-Simulation** window or use the command line API, `exportToFMU2CS('mdlName','SaveSourceCodeToFMU','on')` to export the model to FMU with C source code.

Note To export a Simulink model to FMU with C source code, install Simulink Coder

If the Simulink model contains model references with custom datatypes or fixed-point functions, exporting FMU with source code may error out due to duplicate header files in the `_sharedutils` folder. Follow instructions on Generate Shared Utility Code to set **Code Generation** > **Interface** > **Shared Code Placement** parameter to '`Shared Location`' and regenerate FMU.

Limitations

You cannot generate FMU from Simulink model, due to these limitations:

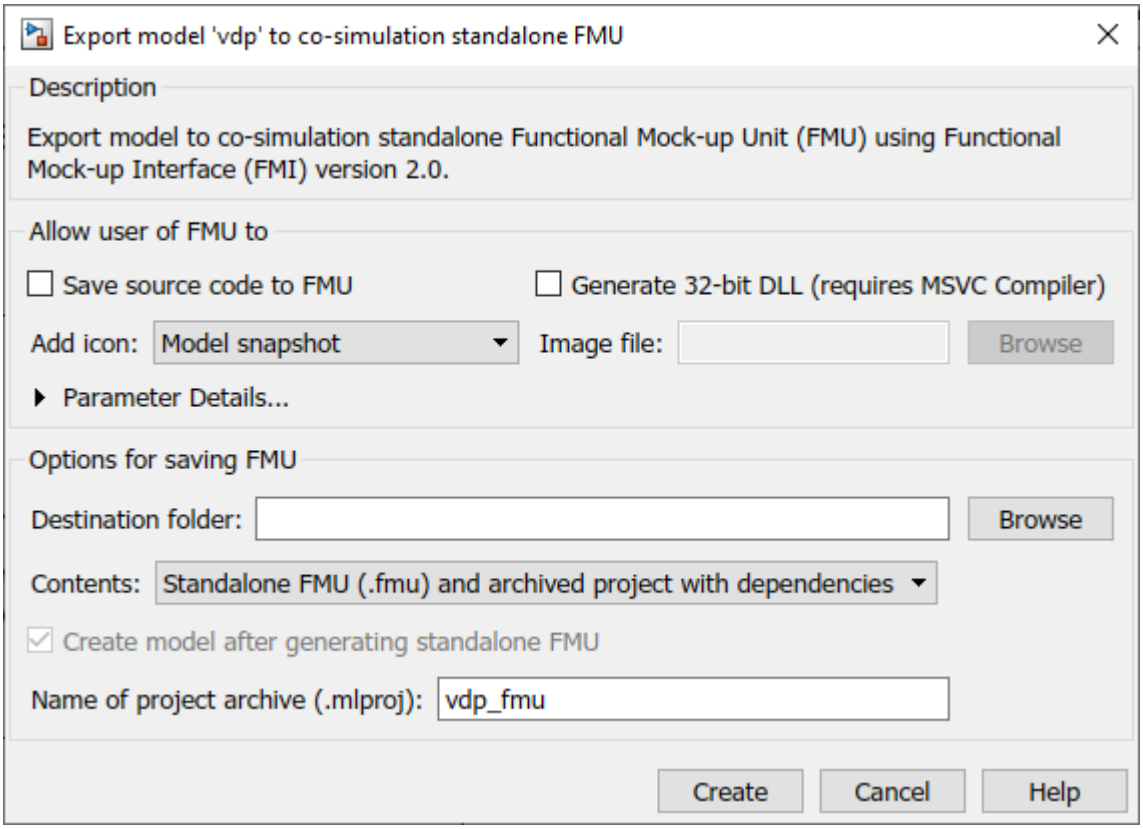
- Variable-step solvers are not supported
- Non-zero simulation start time are not supported
- Simulink model that references external resources (data files, mex or m files) are not supported

Export a Simulink Model

Use the Export Dialog Box

Export the `vdp` example using Simulink toolstrip: **Simulation** > **Save** > **Export Model ToStandalone FMU**

1. Open the model, `vdp`
2. In Simulink editor, Simulink toolstrip: **Simulation** > **Save** > **Standalone FMU**.
3. In Simulink editor, select **Save> Export Model to> FMU Co-Simulation**.
4. In the export dialog box, specify the path to export the FMU.



5 Click Create

By default, Simulink creates the FMU and a harness model with its dependencies stored in a `MAT` file. It then packs them into archived project (`.mlproj`). You can change the behavior by setting **Contents** option to **Standalone FMU (.fmu0 only**.

Use the Programmatic Interface

- Export the vdp example to an FMU using the default `exportToFMU2CS` function. This command creates the FMU file, *modelName.fmu*. By default, it also creates a Simulink model, *modelName_fmu.slx*, that contains an FMU Co-Simulation block with the original model. Create this model if you want to check the integrity of the exported FMU.

  ```
  load_system('vdp')
  set_param('vdp', 'SolverType', 'Fixed-step')
  exportToFMU2CS('vdp')
  ```

- Export the vdp example to an FMU using the exportToFMU2CS function, but do not create a Simulink model. This command creates the FMU file, *modelName.fmu*.

  ```
  load_system('vdp')
  set_param('vdp', 'SolverType', 'Fixed-step')
  exportToFMU2CS('vdp','CreateModelAfterGeneratingFMU','off')
  ```

- Export the vdp example to an FMU using the `exportToFMU2CS` function. Create a model for the FMU and use an image of the original model as the block icon. This command creates the FMU file, *modelName.fmu*, and a Simulink model with an FMU Co-Simulation block whose block icon is the original model.

  ```
  exportToFMU2CS('vdp','AddIcon','snapshot')
  ```

See Also

`configureForDeployment` | `Simulink.SimulationInput` | `mcc` | `deploytool` | `sim`

Related Examples

- “Deploy Simulations with Tunable Parameters”
- “Deploy an App Designer Simulation with Simulink Compiler”
- “Simulation Callbacks for Deployable Applications”