

Simulink® Check™

Support Package for CI/CD Automation for Simulink®
Check™ Reference



MATLAB® & SIMULINK®

R2024b



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

CI/CD Automation for Simulink® Check™ Reference

© COPYRIGHT 2022-2024 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.

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

September 2024 Online only New for Version 24.2 (R2024b)

1	Run Tasks with Process Advisor
2	Customize Your Process Model
3	Build System
4	Pipeline Generator
5	Report Generator API
6	Utilities
7	Built-In Task Library
8	Built-In Query Library

Run Tasks with Process Advisor

Process Advisor


Automate your development workflow and prequalify changes before submitting to source control

Description


With the CI/CD Automation for Simulink Check support package, you can use the Process Advisor app to create, deploy, and automate a consistent prequalification process for Model-Based Design (MBD). The app includes built-in tasks for performing common MBD tasks like checking modeling standards with the Model Advisor app, running tests with Simulink Test™, generating code with Embedded Coder®, and inspecting code with Simulink Code Inspector™. You can use the customizable process modeling system to define the steps in your process and use the app to run each of the steps. As you edit and save the artifacts in your project, the app tracks changes and automatically identifies tasks and task iterations that have outdated results. The Process Advisor app runs your tasks locally for prequalification. The tasks run on the machine that is running MATLAB® and does not use an external CI system.

If your process model defines multiple processes, you can select which process you want to use from the **Processes** gallery in the toolstrip. For more information, see “Manage Multiple Build and Verification Workflows Using Processes”.

In Process Advisor, you can:

- Point to a task in the **Tasks** column and click the run button  to run that task and outdated dependent tasks.
- Click **Run All** to run each of the tasks shown in the **Tasks** column.
- Click **Run All > Force Run All** to force the build system to run each task, even if the tasks already have up-to-date results.
- Click **Run All > Dry Run All** to dry run each task in the process. A dry run can help you test your process model by validating task inputs and generating representative task outputs without actually running the tasks. For more information, see “Dry Run Tasks to Test Process Model”.
- Click **Run All > Clean All** to clear the task results and delete task outputs for each of the tasks.
- Click **Run All > Refresh All** to manually refresh the list of tasks that appears in the **Tasks** column.
- Click **Run All > Set Dry Run as Default** to make dry runs the default task execution mode for the current MATLAB session. For more information, see “Dry Run Tasks to Test Process Model”.

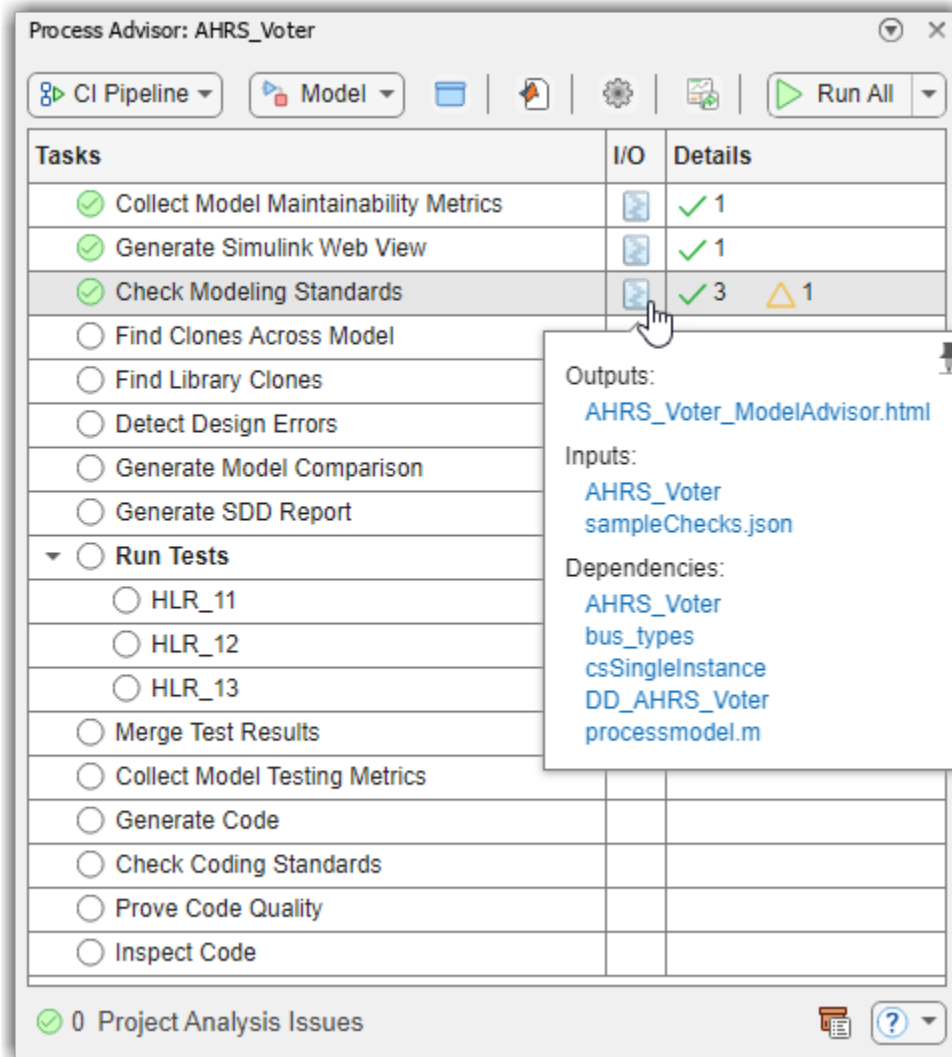
When the Process Advisor app runs tasks, a **Stop** button appears in the top-right corner. You can click the **Stop** button to stop the queued tasks from running next.

To edit the process model, click the **Edit process model** icon . If you have a P-coded process model file, you must delete the `processmodel.p` file before you can edit the process model using Process Advisor.

After Process Advisor analyzes the project, the **Project Analysis Issues** pane shows the errors or warnings that the artifact analysis generated. For more information, see “Troubleshoot Missing Tasks, Artifacts, and Dependencies”.

1 12 Project Analysis Issues

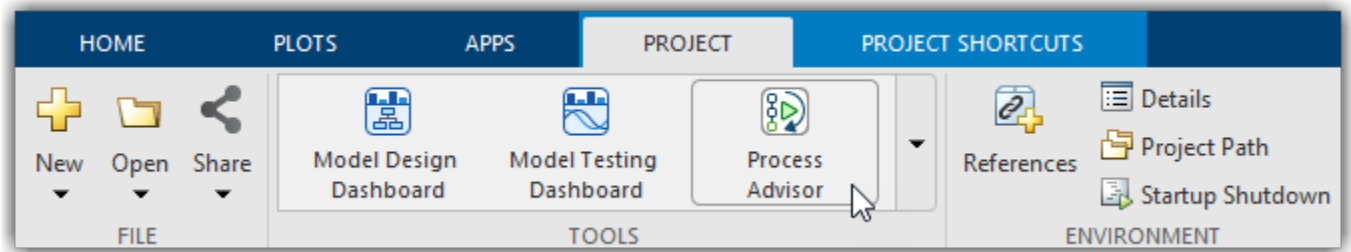
For information on how to get started, see “Automate and Run Tasks with Process Advisor”.



Open the Process Advisor App

- For a Simulink model:
 - On the **Apps** tab, click **Process Advisor**.
 - Or, in the Command Window, enter:


```
processadvisor(modelName)
```
- For a project:
 - On the **Project** tab, in the **Tools** section, click **Process Advisor**.



- Or, in the Command Window, enter:
`processAdvisorWindow`

Examples

Open Process Advisor For Model

Open the Process Advisor app for a Simulink model in a project.

Create and open a working copy of the Process Advisor example project. MATLAB copies the files to an example folder so that you can edit them.

```
processAdvisorExampleStart
```

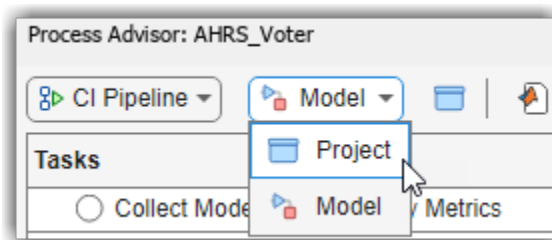
The project contains the model `AHRS_Voter.slx`.

Open the Process Advisor app for the model `AHRS_Voter.slx`.

```
processadvisor("AHRS_Voter")
```

By default, the Process Advisor pane shows the tasks for the current model.

To view the tasks associated with the project, in the Process Advisor pane, you can switch the filter from **Model** to **Project**.



Open Process Advisor For Project

Open the Process Advisor for a project and view the pipeline of tasks.


Create and open a working copy of an example project. MATLAB copies the files to an example folder so that you can edit them.


```
proj = Simulink.createFromTemplate("code_generation_example.sltx", ...  
Name="New Project");
```

Open the Process Advisor for the project.

```
processAdvisorWindow
```

The **Tasks** column shows the pipeline of tasks generated from the process model.

Click **Edit**  to open the process model file that defines the process.

- “Automate and Run Tasks with Process Advisor”
- “Modify Default Process Model to Fit Your Process”
- “Overview of Process Model”
- “Approaches to Running Processes in CI”

Programmatic Use

Note that you need to load a project before you open the Process Advisor.

`processadvisor(modelName)` opens the Simulink model, `modelName`, in the current project and opens a Process Advisor pane to the left of the Simulink canvas.

`processAdvisorWindow()` opens the Process Advisor app for the current project. The app opens in a standalone window.

See Also

[generateReport](#) | [processadvisor](#) | [processAdvisorWindow](#) | [runprocess](#)

Topics

“Automate and Run Tasks with Process Advisor”

“Modify Default Process Model to Fit Your Process”

“Overview of Process Model”

“Approaches to Running Processes in CI”

processadvisor

Open Process Advisor app for Simulink model

Syntax

```
processadvisor(modelName)  
processadvisor( ____, processName)
```

Description

`processadvisor(modelName)` opens the Simulink model, `modelName`, in the current project and opens a Process Advisor pane to the left of the Simulink canvas. You need to load a project to use the function `processadvisor`.

This function requires CI/CD Automation for Simulink Check.

`processadvisor(____, processName)` opens Process Advisor for the process specified by `processName`. By default, the function opens Process Advisor for the default process in the process model.

Examples

Open Process Advisor for Model in Project

Open the Process Advisor app for a specific model in a project.

Open the Process Advisor example project, which contains an example model `AHRS_Voter.slx`.

```
processAdvisorExampleStart
```

Open the Process Advisor app for the model `AHRS_Voter.slx`.

```
processadvisor("AHRS_Voter")
```

The `AHRS_Voter` model opens in Simulink and the Process Advisor app opens in a pane to the left of the Simulink canvas. You can use the Process Advisor app to run the tasks in your process.

Input Arguments

modelName — Model name

character vector | string

Model name, specified as a character vector or string.

Do not include the model extension (`.slx` or `.mdl`) in the model name.

Example: "AHRS_Voter"

Data Types: char | string

processName — Process name

string

Process name, specified as a string.

Example: "CIPipeline"

Data Types: string

Alternative Functionality**App**

You can also open the Process Advisor app for a model by using the Apps Gallery.

- 1 Open a Simulink model in your project.
- 2 Click the **Apps** tab.
- 3 In the **Model Verification, Validation, and Test** section, click **Process Advisor**.

See Also

Process Advisor | generateReport | processAdvisorWindow | runprocess

Topics

"Automate and Run Tasks with Process Advisor"

"Modify Default Process Model to Fit Your Process"

"Overview of Process Model"

"Approaches to Running Processes in CI"

processAdvisorWindow

Open standalone Process Advisor window for project

Syntax

```
processAdvisorWindow()
```

Description

`processAdvisorWindow()` opens the Process Advisor app for the current project. The app opens in a standalone window.

This function requires CI/CD Automation for Simulink Check.

Examples

Open Standalone Process Advisor Window

Open the Process Advisor app for a project.

Open the **Process Advisor** example project, which contains an example process model.

```
processAdvisorExampleStart
```

Open the Process Advisor app for the project.

```
processAdvisorWindow()
```

The standalone **Process Advisor** window shows each of the task iterations in the project, organized by task. In the **Task** column, the table shows each task and the artifacts that the task iterates over. You can double-click on an artifact name to open the artifact. For example, if you double-click on the name of a test case, the test case opens in Test Manager.

Alternative Functionality

App

You can also open the Process Advisor app for a project directly from the **Project** tab in MATLAB.

On the **Project** tab, in the **Tools** gallery, click **Process Advisor**.

See Also

Process Advisor | generateReport | processadvisor | runprocess

Topics

“Automate and Run Tasks with Process Advisor”

“Modify Default Process Model to Fit Your Process”

“Overview of Process Model”

"Approaches to Running Processes in CI"

Customize Your Process Model

createprocess

Create process model

Syntax

```
processModelPath = createprocess()  
processModelPath = createprocess(Name=Value)
```

Description

`processModelPath = createprocess()` creates a process model at the project root and returns the path to the created process model. The process model is saved as `processmodel.m`.

By default, the function creates a process model using the default process model template. The default template includes several built-in tasks for common model-based design activities. You can only call `createprocess` if you have a project open.

This function requires CI/CD Automation for Simulink Check.

`processModelPath = createprocess(Name=Value)` specifies the output process model using one or more `Name=Value` arguments.

Examples

Create Process Model

Open a project that does not have a process model and use `createprocess` to create a copy of the default process into the project.

Open an example project, for example `matlab.project.example.timesTable`, that does not have a process model.

Create a process model for the project.

```
processModelPath = createprocess
```

`createprocess` copies the default process model into the project root and saves the path to the process model to `processModelPath`.

Create a project object for the currently loaded project.

```
myProject = currentProject;
```

Add the process model file to the current project.

```
addFile(myProject, processModelPath)
```

Open the Process Advisor app in a standalone window to view the tasks associated with the project and project artifacts.


```
processAdvisorWindow
```

Overwrite Process Model with Empty Process

Open a project and overwrite the process model with an empty process model.

Open the **Process Advisor** example project, which contains an example process model.

```
processAdvisorExampleStart
```

Use `createprocess` to overwrite the existing process model with an empty process model.

```
processModelPath = createprocess(Template="empty",Overwrite=true)
```

Open the created process model to view the commented-out example code.

```
open(processModelPath)
```

Input Arguments

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, . . . ,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `processModelPath = createprocess(Overwrite=true)`

Template — Name of predefined process model template

```
"default" (default) | "empty" | "parallel"
```

Name of predefined process model template, specified as either:

- `"default"` — Process model file that includes several built-in tasks
- `"empty"` — Process model file that contains commented-out example code for adding built-in and custom tasks
- `"parallel"` — Process model file designed for parallel CI jobs.

Example: `"empty"`

Data Types: `char` | `string`

Overwrite — Setting to overwrite existing process model

```
false or 0 (default) | true or 1
```

Setting to overwrite existing process model, specified as a numeric or logical `0` (`false`) or `1` (`true`).

Example: `true`

Data Types: `logical`

Subprocess — Set up default process model to use subprocesses to group tasks

```
false or 0 (default) | true or 1
```

Set up default process model to group model verification and code verification tasks using subprocesses, specified as a numeric or logical `0` (`false`) or `1` (`true`).

If you specify `Subprocess` as `true`, the default process model template contains a subprocess for model verification tasks, "Model Verification", and a subprocess for code verification tasks, "Code Verification".

Example: `true`

Data Types: `logical`

Output Arguments

processModelPath — Path to created process model

character vector

Path to created process model, returned as a character vector.

By default, `createprocess` creates a process model at the project root.

Alternative Functionality

App

If a project does not have a process model, you can use the Process Advisor app to create the default process model. To open the Process Advisor app for a project, in the MATLAB Command Window, enter:

```
processAdvisorWindow
```

When you open the Process Advisor app on a project that does not have a process model, the app automatically creates a copy of the default process model at the root of the project.

See Also

`getprocess` | Process Advisor | `runprocess`

Topics

"Automate and Run Tasks with Process Advisor"

"Modify Default Process Model to Fit Your Process"

"Overview of Process Model"

"Approaches to Running Processes in CI"

getprocess

Get process model object for process model in project

Syntax

```
processModelObject = getprocess()
```

Description

`processModelObject = getprocess()` returns a process model object, `processModelObject`, for the process model in the project. You can use the process model object to view the properties of the process model in the project. For more information, see `padv.ProcessModel`.

If the current project does not have a process model, the function `getprocess` automatically creates a new process model at the root of the project.

This function requires CI/CD Automation for Simulink Check.

Examples

Find the Default Query for the Current Process

Use `getprocess` to find the default query that the current process model uses. If you have a task that does not specify an iteration query, the default query defines which artifacts the process iterates over. By default, custom tasks run once per project because the default query is `"padv.builtin.query.FindProjectFile"`.

Open the **Process Advisor** example project, which contains an example process model.

```
processAdvisorExampleStart
```

Get the properties of the current process model.

```
currentProcessModel = getprocess()
```

Get the default query for the current process model.

```
defaultQuery = currentProcessModel.DefaultQueryName
```

```
defaultQuery =
```

```
    "padv.builtin.query.FindProjectFile"
```

You can use the `findTask` and `findQuery` functions on the loaded process model to find specific tasks and queries in the process.

```
findTask(currentProcessModel, "padv.builtin.task.RunModelStandards")
```

Output Arguments

processModelObject — Properties of process model

`padv.ProcessModel` object

Properties of process model, returned as a `padv.ProcessModel` object.

The `padv.ProcessModel` object returns the names of the tasks, queries, default query, and root process model file for the process.

See Also

`createprocess` | `Process Advisor` | `runprocess`

Topics

“Automate and Run Tasks with Process Advisor”

“Modify Default Process Model to Fit Your Process”

“Overview of Process Model”

“Approaches to Running Processes in CI”

padv.Artifact

Store artifact information

Description

This object requires CI/CD Automation for Simulink Check. A `padv.Artifact` object represents an artifact that you can run a task on in the process defined in your process model. For example, you can use a `padv.Artifact` object as the input to functions like `runprocess` and `generateProcessTasks` when you only want to run or generate tasks associated with a specific artifact.

Creation

Syntax

```
artifactObject = padv.Artifact(artifactType,artifactAddress)
artifactObject = padv.Artifact( ___,Name=Value)
```

Description

`artifactObject = padv.Artifact(artifactType,artifactAddress)` stores artifact information in a `padv.Artifact` object, `artifactObject`. You can use the artifact information when you want to get the ID for a specific task iteration.

`artifactObject = padv.Artifact(___,Name=Value)` specifies the artifact using one or more `Name=Value` arguments.

Input Arguments

artifactType — Type of artifact

"sl_model_file" | "m_file" | "ma_config_file" | "sl_test_case" | ...

Type of artifact, specified as one or more of the values listed in this table. To specify multiple values, use an array. The build system uses the artifact type to identify and categorize the different file types and modeling constructs in your project.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file

Category	Artifact Type	Description
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow®	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer™	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result

Category	Artifact Type	Description
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "sl_model_file"

Example: ["sl_model_file "zc_file"]

artifactAddress — Address of artifact

padv.util.ArtifactAddress object

Address of artifact, specified as an padv.util.ArtifactAddress object. Note that the address is relative to the project root.

The padv.util.ArtifactAddress object represents the file address of an artifact in your project, but also has properties that contain information about the artifact change tracking. For more information, see padv.util.ArtifactAddress and “Exclude Files from Change Tracking in Process Advisor”.

Example:

```
padv.util.ArtifactAddress(fullfile("02_Models", "AHRV_Voter", "specification", "AHRV_Voter.slx"))
```

Data Types: string

Properties

Alias — Human-readable name for artifact

empty string (default) | string

Human-readable name for the artifact in the Process Advisor user interface, specified as a string.

If you want to customize how artifact names appear in the Process Advisor, create a custom query that updates the values of the Alias property for each padv.Artifact object that the query returns.

Data Types: string

Type — Type of artifact

"sl_model_file" | "m_file" | "ma_config_file" | "sl_test_case" | ...

Type of artifact, specified as one or more of the values listed in this table. To specify multiple values, use an array. The build system uses the artifact type to identify and categorize the different file types and modeling constructs in your project.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component

Category	Artifact Type	Description
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "sl_model_file"

Example: ["sl_model_file" "zc_file"]

Parent – Reference to parent artifact

padv.Artifact object

Reference to parent artifact, specified as a padv.Artifact object.

ArtifactAddress – Address of artifact in project

padv.util.ArtifactAddress object

Address of artifact in project, specified as a padv.util.ArtifactAddress object.

Object Functions

Object Function	Description
getTypes	Get artifact type. TYPES = getTypes(artifactObj)
getKey	Get unique key for artifact. A key is a unique address for a file. KEY = getKey(artifactObj)

Object Function	Description
hasType	Check if artifact has type. TYPE = hasType(artifactObj)

Examples

Run Task Associated with Model

Suppose you have a process model with several tasks, but right now you only want to run test cases associated with a single model. You can use a `padv.Artifact` object to specify the model and use the `runprocess` function to run the test cases for that model.

Open the Process Advisor example project, which contains an example process model.

```
processAdvisorExampleStart
```

The example process contains a **Run Tests** task (`padv.builtin.task.RunTestsPerTestCase`) that runs the test cases in the project.

Create a `padv.Artifact` object that represents the model that you want to run. For this example, the artifact type is `"sl_model_file"` because the artifact is a Simulink model and the address is the path to model `AHRS_Voter.slx`, relative to the project root.

```
model = padv.Artifact(...
"sl_model_file",...
fullfile("02_Models","AHRS_Voter","specification","AHRS_Voter.slx"));
```

Run the **Run Tests** task on the test cases associated with the model `AHRS_Voter.slx` by specifying the name-value arguments of the `runprocess` function.

```
runprocess(...
Tasks = "padv.builtin.task.RunTestsPerTestCase",...
FilterArtifact = model)
```

The build system only runs the test cases associated with the specified model.

See Also

`generateProcessTasks` | `padv.util.ArtifactAddress` | `runprocess`

Topics

- “Automate and Run Tasks with Process Advisor”
- “Programmatically Run Tasks”
- “Find Artifacts with Queries”
- “Create Custom Queries”

padv.Process Class

Namespace: padv

Group tasks and subprocesses in process model

Description

This class requires CI/CD Automation for Simulink Check.

A `padv.Process` object represents a group of tasks and subprocesses in your process model. By default, if you do not add any processes to your process model, your process model has a default process "CIPipeline". To create other processes in your process model, create a new process object by using the method `addProcess`. You can group tasks and other subprocesses inside a specified process by using `addTask` and `addSubprocess`. You can specify a dependency or desired execution order between tasks and subprocesses inside your process by using either `addDependsOnRelationship` or `addRunsAfterRelationship`. For more information, see "Manage Multiple Build and Verification Workflows Using Processes".

The `padv.Process` class is a `handle` class.

Creation

Syntax

```
process = padv.Process(Name)
process = padv.Process( ____,Name=Value)
```

Description

`process = padv.Process(Name)` represents a process, named `Name`, inside a process model. Each process in the process model must have a unique `Name`.

`process = padv.Process(____,Name=Value)` sets properties using one or more name-value arguments. For example, `padv.Process("myProcess",Title="My Process")` creates a process with the title `My Process` in Process Advisor.

Specify optional pairs of arguments as `Name1=Value1, . . . ,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Properties

DescriptionCSH — Path to process documentation

string

Path to process documentation, returned as a string.

```
Example: padv.Process("myProcess",DescriptionCSH =
fullfile(pwd,"myHelpFiles","myProcessDocumentation.pdf"))
```

Data Types: string

DescriptionText – Process description

string

Process description, returned as a string.

Example: `padv.Process("myProcess",DescriptionText = "This is my process.")`

Data Types: string

Title – Human readable name that appears in Process Advisor app

string

Human readable name that appears in the **Processes** drop-down menu in the Process Advisor app, returned as a string. By default, the Process Advisor app uses the **Name** property of the process as the **Title**.

Example: `padv.Process("myProcess",Title = "My Process")`

Data Types: string

Name – Unique identifier for process

string

Unique identifier for the process, returned as a string. When you specify the **Name**, you specify the **Name** property of the process object.

Each process in the process model must have a unique **Name**.

Example: `padv.Process("myProcess")`

Data Types: string

Methods

Public Methods

<code>addTask</code>	Add task to process. <code>myProcess.addTask("myTask");</code>
<code>addSubprocess</code>	Add subprocess to process. <code>myProcess.addSubprocess("mySubprocess");</code>

addDependsOnRelationship	<p>Create dependency between two tasks or subprocesses.</p> <pre>myProcess.addDependsOnRelationship(... Source=taskB, ... Dependency=taskA);</pre> <p>The build system always runs the Dependency before the Source. Use this method when one task or subprocess cannot start without another task or subprocess finishing first. Otherwise, if you only want to specify a preferred task execution order, you can use <code>addRunsAfterRelationship</code> instead.</p>
addRunsAfterRelationship	<p>Specify predecessor for task or subprocess.</p> <pre>myProcess.addRunsAfterRelationship(... Source=taskB, ... Predecessor=taskA);</pre> <p>When you run your process, the build system runs the Predecessor before the Source when possible. But if you force run the Source, the build system runs that task or subprocess independently. Use this method for tasks and subprocesses that you prefer to run in a specific order, but do not have a strict dependency. If a task must run before another task to run successfully, use <code>addDependsOnRelationship</code> instead.</p>

Examples

Create New Process Inside Process Model

You can use `addProcess` to create a new process and add that process to the process model. `addProcess` returns a `padv.Process` object that can represent a group of tasks and subprocesses in a process model.

For example, this process model creates a new process, `ProcessA`, adds tasks to the process, and adds a dependency between those tasks.

```
function processmodel(pm)
    % Defines the project's processmodel

    arguments
        pm padv.ProcessModel
    end

    % Create and add process to process model
    processA = pm.addProcess("ProcessA");

    % Add tasks to Process A
```

```
taskA = processA.addTask("taskA");
taskB = processA.addTask("taskB");

% Add dependency between tasks inside Process A
processA.addDependsOnRelationship(...
    Source = taskB,...
    Dependency = taskA);

end
```

See Also

`padv.ProcessModel` | `runprocess`

Topics

"Automate and Run Tasks with Process Advisor"

"Modify Default Process Model to Fit Your Process"

"Overview of Process Model"

"Manage Multiple Build and Verification Workflows Using Processes"

padv.ProcessModel

Define tasks and process for project

Description

This object requires CI/CD Automation for Simulink Check. A `padv.ProcessModel` object represents the process model that defines the tasks and process for a project. A *task* performs an action and is a single step in your process. A *process* is a series of tasks that run in a specific order. The process model defines the tasks that you can perform on the project, and the order and relationships between tasks in the process. You can use tasks and queries to dynamically perform actions and find artifacts in the project. Use the `addTask` object function to add tasks to the process model. You can use the function `runprocess` to run the tasks defined in the process model. For more information, see “Overview of Process Model”.

Creation

Syntax

```
pm = padv.ProcessModel()
```

Description

`pm = padv.ProcessModel()` creates an empty process model object, `pm`.

Properties

TaskNames — Tasks added to process model object

string array

Tasks added to process model object, returned as string array.

Use the object function `addTask` to add a task instance to a process model.

Example: `["padv.builtin.task.GenerateSimulinkWebView" "padv.builtin.task.RunModelStandards"]`

Data Types: `string`

QueryNames — Queries added to process model object

string array

Queries added to process model object, returned as string array.

Use the object function `addQuery` to add a query instance to a process model.

Example: `["padv.builtin.query.FindModels" "padv.builtin.query.FindProjectFile"]`

Data Types: `string`

ProcessNames — Processes in process model

string

Processes in the process model, specified as a string.

For more information about processes, see “Manage Multiple Build and Verification Workflows Using Processes”.

Example: ["CIPipeline" "ProcessA"]

Data Types: string

DefaultQueryName — Default query for tasks added to process model object

"padv.builtin.query.FindProjectFile" (default) | name of padv.Query query

Default query for tasks added to process model, specified as the name of a padv.Query query.

Example: "padv.builtin.query.FindModels"

Data Types: string

DefaultOutputDirectory — Default output directory for results

fullfile("\$PROJECTROOT\$", "PA_Results") (default) | string array

Default output directory, specified as a string array. Set the default output directory to a path inside your project. The path can be either a relative or absolute path. Consider using the path relative to the project root to promote consistency across local environments and CI systems, and allow for more portable builds.

The default property value uses the token \$PROJECTROOT\$ as a placeholder for dynamic path resolution during run-time. The build system considers paths that use tokens as absolute paths. For more information, see “Dynamically Resolve Paths with Tokens”.

By default, Process Advisor and the build system output results in a folder PA_Results in the project root.

Example: fullfile("\$PROJECTROOT\$", "Process_Results")

Data Types: string

JUnitReportName — Name of generated JUnit-style XML report

"\$TASKNAME\$_\$ITERATIONARTIFACT\$_JUnit.xml" (default) | string array

Name of generated JUnit-style XML report , specified as a string array.

By default, the generated JUnit report for a task has the format *taskName_iterationArtifact_JUnit.xml*.

The default property value uses the tokens \$TASKNAME\$ and \$ITERATIONARTIFACT\$ as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Example: "\$TASKNAME\$_\$ITERATIONARTIFACT\$_JUnitReport.xml"

Data Types: string

JUnitReportPath — Location for JUnit-style XML report

fullfile("\$DEFAULTOUTPUTDIR\$", "junit") (default) | string array

Location for JUnit-style XML report, specified as a string array.

The default property value uses the token `$DEFAULTOUTPUTDIR$` as a placeholder for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Example: `fullfile("$DEFAULTOUTPUTDIR$", "junit", "reports")`

Data Types: `string`

DefaultProcessId — Name of default process for project

`string`

Name of default process for project, specified as a string.

The first time that you open Process Advisor, the app opens to the default process. Otherwise, Process Advisor re-opens your last opened process. By default, several APIs use the default process unless you specify a different process in the process model. For more information, see “Manage Multiple Build and Verification Workflows Using Processes”.

Example: `"ProcessA"`

Data Types: `string`

EnablePerformanceChecks — Turn on performance improvement checks

`1 (true) (default) | 0 (false)`

Turn on performance improvement checks for the process model, specified as a numeric or logical `1 (true)` or `0 (false)`.

When `EnablePerformanceChecks` is `true`, the build system identifies and warns you about inefficient process model code. For example, the build system can generate a warning if multiple tasks in the process model use the same query but do not share the same query object.

Example: `false`

Data Types: `logical`

DefaultDryRunResults — Default task results when task dry-runs

`padv.TaskResult()` (default) | `padv.TaskResult` object

Default task results when task dry-runs, specified as a `padv.TaskResult` object.

If you dry-run a task that does not have a dry-run behavior specified, the task returns the default dry-run results specified by `DefaultDryRunResults`. To specify a dry-run behavior for a task, you can use the `dryRun` method for class-based tasks or the `DryRunAction` function for function-based tasks.

By default, `DefaultDryRunResults` returns a `padv.TaskResult` object. You can create a different set of default dry-run results by creating and using a `padv.TaskResult` object with different property values. For example, to have the default dry-run results be failing task results with specific result values in the **Details** column, in your process model you can create a `padv.TaskResult` object and update the value of the `DefaultDryRunResults` property:

```
res = padv.TaskResult;
res.Status = padv.TaskStatus.Fail;
res.ResultValues = struct(...
    "Pass", 1, ...
    "Warn", 2, ...
```

```
"Fail",3);
pm.DefaultDryRunResults = res;
```

Example: `padv.TaskResult`

RootFileName — Name of process model file

string

Name of process model file, returned as a string.

`RootFileName` uses `processmodel.m` as the name of the process model file, unless a `processmodel.p` file exists. If you have both a P-code file and a `.m` file, the P-code file takes precedence over the corresponding `.m` file for execution, even after modifications to the `.m` file.

The default name of the process model file is specified by `DefaultRootFileName`.

Data Types: string

DefaultRootFileName — Default name of process model file

"processmodel.m" (default) | string

Default name of process model file, specified as a string.

Data Types: string

Object Functions

reset	Removes tasks and queries from process model <code>pm = padv.ProcessModel();</code> <code>reset(pm);</code>
reload	Load process model by executing process model file for project <code>pm = padv.ProcessModel();</code> <code>reload(pm);</code>
addProcess	Add process to process model <code>processA = pm.addProcess("processA");</code>
addSubprocess	Add subprocess instance to process model <code>addSubprocess(pm, "MySubprocess");</code>
addTask	Add task instance to process model <code>addTask(pm, "MyCustomTask", ...</code> <code>Action=@SayHello, ...</code> <code>IterationQuery=padv.builtin.query.FindModels);</code> When you use <code>addTask</code> on a process model object, the function adds the task to the default process. To add a task to a specific process inside the process model, use <code>addTask</code> on the process object. For information, see <code>addTask</code> .

addQuery	Add query instance to process model <code>addQuery(pm, "MyCustomQuery")</code> For information, see <code>addQuery</code> .
findProcess	Find process in process model <code>pm = getprocess;</code> <code>ci = pm.findProcess("CIPipeline")</code>
findQuery	Find query instance by name <code>pm = padv.ProcessModel();</code> <code>QUERY = findQuery(pm, ...</code> <code>"padv.builtin.query.FindModels")</code>
findTask	Find task instance by name <code>pm = padv.ProcessModel();</code> <code>TASK = findTask(pm, ...</code> <code>"padv.builtin.task.RunModelStandards");</code>
exists	Check if process model exists for project <code>[FOUND, PATH] = padv.ProcessModel.exists()</code>

Examples


Add Tasks to Process Model Object

You can add tasks directly to your process model by using the object function `addTask` for the `padv.ProcessModel` object in your process model. When you add a task using this approach, the `padv.ProcessModel` object automatically creates a process named `CIPipeline` and adds the task to that process. For information on how to create your own processes, see “Manage Multiple Build and Verification Workflows Using Processes”.

Open the Process Advisor example project.

```
processAdvisorExampleStart
```

The model `AHRS_Voter` opens with the Process Advisor pane to the left of the Simulink canvas.

In the Process Advisor pane, click the **Edit process model**  button to open the `processmodel.m` file for the project.

Replace the contents of the `processmodel.m` file with this code:

```
function processmodel(pm)
    arguments
        pm padv.ProcessModel
    end

    addTask(pm, "taskA");
    addTask(pm, "taskB");

end
```

The function `addTask` adds the task objects to the `padv.ProcessModel` object.

Use the function `getprocess` to get the process model object for the project.

```
pm = getprocess;
```

Get the task object for "taskA" defined in the current process model.

```
taskAObj = findTask(pm, "taskA");
```

`taskAObj` is a `padv.Task` object that you can use to view the properties of the task "taskA".


Add Processes to Process Model

You can add processes to your process model by using the object function `addProcess` for the `padv.ProcessModel` object inside your process model. Then you can add tasks to those processes by using the `addTask` function.

Open the Process Advisor example project.

```
processAdvisorExampleStart
```

The model `AHRS_Voter` opens with the Process Advisor pane to the left of the Simulink canvas.

In the Process Advisor pane, click the **Edit process model**  button to open the `processmodel.m` file for the project.

Replace the contents of the `processmodel.m` file with this code:

```
function processmodel(pm)
    % Defines process model for project by setting up proceses and tasks

    arguments
        pm padv.ProcessModel
    end

    % --- Processes ---
    processA = pm.addProcess("ProcessA");
    processB = pm.addProcess("ProcessB");

    % --- Tasks ---
    taskA1 = processA.addTask("TaskA1");
    taskB1 = processB.addTask("TaskB1");
    taskA2 = processA.addTask("TaskA2");
    taskB2 = processB.addTask("TaskB2");

end
```

The function `addProcess` adds processes to the `padv.ProcessModel` object. The function `addTask` adds example tasks to specific processes.

For information on how to create your own processes, see "Manage Multiple Build and Verification Workflows Using Processes".

Change Default Dry Run Results

Suppose that you want the default dry run results to show failing task results with specific result values in the **Details** column.

By default, if a task does not have a dry run functionality defined, the task returns the default dry run results specified by the `padv.ProcessModel` property `DefaultDryRunResults`. You can create a different set of default dry run results by creating and using a `padv.TaskResult` object with different property values.

Inside your process model, you can create a `padv.TaskResult` object and update the value of the `DefaultDryRunResults` property of the process model. In this example, the process model also includes a custom task, `MyCustomTask`, that does not specify any specialized dry run functionality.

```
function processmodel(pm)
    % Defines the project's processmodel
    arguments
        pm padv.ProcessModel
    end

    res = padv.TaskResult;
    res.Status = padv.TaskStatus.Fail;
    res.ResultValues = struct(...
        "Pass",1,...
        "Warn",2,...
        "Fail",3);
    pm.DefaultDryRunResults = res;

    pm.addTask("MyCustomTask");

end
```

In Process Advisor, dry run the task `MyCustomTask`. Since there is no specialized dry run behavior for the task, the task uses the default dry run results specified by `DefaultDryRunResults`.

Tasks	I/O	Details
▼ MyCustomTask		✓ 5 △ 10 ✗ 15
AHRs_Voter		✓ 1 △ 2 ✗ 3
Actuator_Control		✓ 1 △ 2 ✗ 3
Flight_Control		✓ 1 △ 2 ✗ 3
InnerLoop_Control		✓ 1 △ 2 ✗ 3
OuterLoop_Control		✓ 1 △ 2 ✗ 3

Tips

By default, if you make a change to the process model file, Process Advisor marks task results as outdated because the tasks in the updated process model might not match the tasks that generated the task results from the previous version of the process model. If you do not want changes to the process model to make task results outdated, you can open the **Settings** in Process Advisor and clear **Add process model as dependency** in the user settings. For more information, see “Specify Settings for Process Advisor and Build System” and `padv.UserSettings`.

For more information on artifact tracking, see “Exclude Files from Change Tracking in Process Advisor”.

See Also

`addTask` | `createprocess` | `getprocess` | `runprocess`

Topics

“Automate and Run Tasks with Process Advisor”

“Modify Default Process Model to Fit Your Process”

“Overview of Process Model”

“Add Tasks to Process”

“Reconfigure Task Behavior”

“Define Task Relationships”

“Manage Multiple Build and Verification Workflows Using Processes”

addQuery

Namespace: padv

Add query instance to process model

Syntax

```
queryObj = addQuery(pm, queryNameOrInstance)
queryObj = addQuery( ____, Name=Value)
```

Description

`queryObj = addQuery(pm, queryNameOrInstance)` adds the query specified by `queryNameOrInstance` to the process model. You can access the query using the query object `queryObj`.

This functionality requires CI/CD Automation for Simulink Check.

`queryObj = addQuery(____, Name=Value)` specifies the properties of the query using one or more `Name=Value` arguments.

Input Arguments

pm — Process for project

`padv.ProcessModel` object (default) |

Process for project, specified as a `padv.ProcessModel` object.

Example: `pm = padv.ProcessModel`

queryNameOrInstance — Name or instance of query

string | `padv.Query` object

Name or instance of a query, specified as a string or `padv.Query` object.

Example: "NameOfMyQuery"

Example: `padv.builtin.query.FindModels`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example:

DefaultArtifactType — Artifact type returned by query

"padv_output_file" (default) | valid value for the `Type` property of a `padv.Artifact` object

Artifact type returned by the query, specified as a valid value for the `Type` property of a `padv.Artifact` object.

Example: `DefaultArtifactType = "sl_model_file"`

Title — Human readable name

Name property of query (default) | string

Human readable name for the query, specified as a string. By default, the `Title` property of the query is the same as the `Name`.

Example: `Title = "My Query"`

Data Types: `string`

FunctionHandle — Handle to function that runs when you run query object

`function_handle`

Handle to function that runs when you run query object, specified as a `function_handle`.

When you call the `run` function on a query object, `run` runs the function specified by the `function_handle`.

Example: `FunctionHandle = @FunctionForQuery`

Data Types: `function_handle`

Parent — Initial query run before iteration query

[0×0 `string`] (default) | `padv.Query` object | Name of `padv.Query` object

Initial query run before iteration query, specified as either a `padv.Query` object or the `Name` of a `padv.Query` object. When you specify a `padv.Query` object as the iteration query for a task, the `Parent` query is the initial query that the build system runs before running the specified iteration query.

For example, the built-in `query.padv.builtin.query.FindModelsWithTestCases` has the `Parent` query `padv.builtin.query.FindModels`. If you specify `padv.builtin.query.FindModelsWithTestCases` as the iteration query for a task, you are specifying that you want the task to run once for each model with a test case. The build system runs the `Parent` query `padv.builtin.query.FindModels` first, to find the models in the project, and then the build system runs the iteration query `padv.builtin.query.FindModelsWithTestCases` to find the models with test cases.

The build system ignores the `Parent` query when you specify a query as an input query or dependency query for a task.

Example: `Parent = "padv.builtin.query.FindModels"`

SortArtifacts — Setting for automatically sorting artifacts by address

`true` or `1` (default) | `false` or `0`

Setting for automatically sorting artifacts by address, specified as a numeric or logical `1` (`true`) or `0` (`false`). When a query returns artifacts, the artifacts should be in a consistent order. By default, the build system sorts artifacts by the artifact address.

Alternatively, you can sort artifacts in a different order by overriding the internal `sortArtifacts` method in a subclass that defines a custom sort behavior.

For more information, see “Sort Artifacts in Specific Order”.

The build system automatically calls the `sortArtifacts` method when using the process model. The `sortArtifacts` method expects two input arguments: a `padv.Query` object and a list of `padv.Artifact` objects returned by the `run` function. The `sortArtifacts` method should return a list of sorted `padv.Artifact` objects.

Example: `SortArtifacts = false`

Data Types: `logical`

Output Arguments

queryObj – Query object

`padv.Query` object

Query object, returned as a `padv.Query` object.

For more information, see `padv.Query`.

See Also

`addTask` | `createprocess` | `getprocess` | `padv.ProcessModel` | `padv.Query` | `runprocess`

Topics

“Automate and Run Tasks with Process Advisor”

“Modify Default Process Model to Fit Your Process”

“Overview of Process Model”

“Find Artifacts with Queries”

“Create Custom Queries”

addTask

Namespace: padv

Add task instance to process model

Syntax

```
taskObj = addTask(pm, taskNameOrInstance)
taskObj = addTask( ____, Name=Value)
```

Description

`taskObj = addTask(pm, taskNameOrInstance)` adds the task specified by `taskNameOrInstance` to the process model. You can access the task using the task object `taskObj`. For more information, see “Add Tasks to Process”.

This functionality requires CI/CD Automation for Simulink Check.

`taskObj = addTask(____, Name=Value)` specifies the properties of the task using one or more `Name=Value` arguments.

Examples


Add Tasks to Process Model

You can use the `addTask` function to create function-based tasks directly in the process model.

Open the Process Advisor example project.

```
processAdvisorExampleStart
```

The model `AHRS_Voter` opens with the Process Advisor pane to the left of the Simulink canvas.

In the Process Advisor pane, click the **Edit process model**  button to open the `processmodel.m` file for the project.

Replace the contents of the `processmodel.m` file with this code:

```
function processmodel(pm)
    arguments
        pm padv.ProcessModel
    end

    addTask(pm, "MyCustomTask", Action=@SayHello, ...
        IterationQuery=padv.builtin.query.FindModels);

end

function results = SayHello(~)
    disp("Hello, World!");
```

```

    results = padv.TaskResult;
    results.ResultValues.Pass = 1;
end

```

This code adds a task, `MyCustomTask` to the process model while specifying that the task runs the function `SayHello` one time for each model found in the project. The function `SayHello` also specifies the results returned by the task.

Input Arguments

pm — Process for project

`padv.ProcessModel` object (default)

Process for project, specified as a `padv.ProcessModel` object.

Example: `pm = padv.ProcessModel`

taskNameOrInstance — Name or instance of task

string | `padv.Task` object

Name or instance of a task, specified as a string or `padv.Task` object.

Example: `"NameOfMyTask"`

Example: `padv.builtin.task.RunModelStandards`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example:

```

addTask(pm, "RunOnceForEachModel", IterationQuery=padv.builtin.query.FindModels
)

```

Title — Human readable name that appears in Process Advisor app

Name property of task (default) | string

Human readable name that appears in the **Tasks** column of the Process Advisor app, specified as a string. By default, the Process Advisor app uses the `Name` property of the task as the `Title`.

Example: `"My Task"`

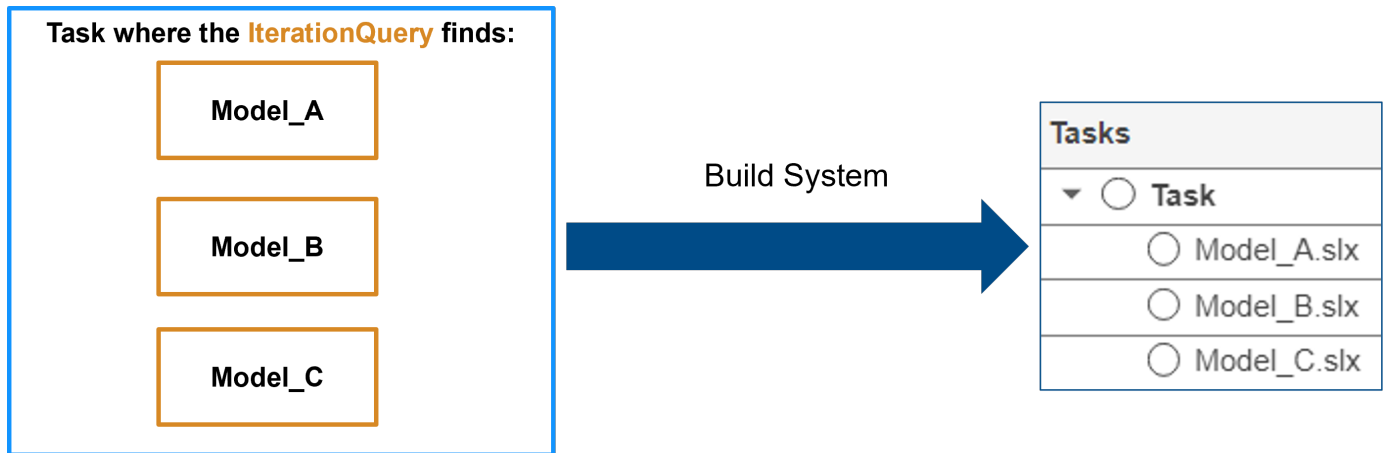
Data Types: `string`

IterationQuery — Artifacts that task iterates over

`padv.Query` object | name of `padv.Query` object

Artifacts that task iterates over, specified as a `padv.Query` object or the name of a `padv.Query` object. By default, task objects run one time and are associated with the project. When you specify `IterationQuery`, the task runs one time *for each* artifact specified by the `padv.Query`. In the Process Advisor app, the artifacts specified by `IterationQuery` appear under task title.

For example, if the `IterationQuery` for a task finds three models, `Model_A`, `Model_B`, and `Model_C`, the build system creates three task iterations under the title of the task in the **Tasks** column.



Each of the artifacts under the task title represents a *task iteration*.

For examples of different IterationQuery values:

- If you have a task where the IterationQuery uses `padv.builtin.query.FindModels` to find each of the models in the project, the build system creates a task iteration for each model.
- If you have a task where the IterationQuery uses `padv.builtin.query.FindProjectFile` to find the project file, the build system creates a task iteration for the project file.
- If you have a task where the IterationQuery uses `padv.builtin.query.FindTopModels` to find top models in the project, the build system creates a task iteration for each top model.

Tasks	
Task iteration for each model	<ul style="list-style-type: none"> Task with IterationQuery = <code>padv.builtin.query.FindModels</code> <ul style="list-style-type: none"> AHRS_Voter.slx Actuator_Control.slx Flight_Control.slx InnerLoop_Control.slx OuterLoop_Control.slx
Task iteration for the project	<ul style="list-style-type: none"> Task with IterationQuery = <code>padv.builtin.query.FindProjectFile</code> <ul style="list-style-type: none"> ProcessAdvisorExample.prj
Task iteration for the top model	<ul style="list-style-type: none"> Task with IterationQuery = <code>padv.builtin.query.FindTopModels</code> <ul style="list-style-type: none"> Flight_Control.slx

Example: `IterationQuery = padv.builtin.query.FindModels`
 Data Types: `string`

InputQueries – Inputs to task

`padv.Query` object | name of `padv.Query` object | array of `padv.Query` objects

Inputs to the task, specified as:

- a `padv.Query` object
- the name of `padv.Query` object
- an array of `padv.Query` objects
- an array of names of `padv.Query` objects

By default, `padv.Task` does not have inputs. When you specify `InputQueries`, the task uses the artifacts specified by the specified query or queries as an input.

Suppose a task runs once for each model in the project and you want to provide the models as inputs to the task. If you specify `InputQueries` as the built-in query `padv.builtin.query.GetIterationArtifact`, the query returns each artifact that the tasks iterates over, which in this example is each of the models in the project.

Example: `InputQueries = padv.builtin.query.GetIterationArtifact`

InputDependencyQuery — Artifact dependencies for task inputs

`padv.Query` object | name of `padv.Query` object

Artifact dependencies for task inputs, specified as a `padv.Query` object or the name of a `padv.Query` object.

The build system runs the query specified by `InputDependencyQuery` to find the dependencies for the task inputs, since those dependencies can impact if task results are up-to-date. Typically, you specify `InputDependencyQuery` as `padv.builtin.query.GetDependentArtifacts` to get the dependent artifacts for each task input. For example, if you specify a model as an input to a task and you specify `InputDependencyQuery` as `padv.builtin.query.GetDependentArtifacts`, the build system can find artifacts, such as data dictionaries, that the model uses.

Example: `InputDependencyQuery = padv.builtin.query.GetDependentArtifacts`

Action — Function that task runs

function handle

Function that the task runs, specified as the function handle. When you run the task, the task runs the function specified by the function handle.

For example, if you want the task to run the function `myFunction`, specify `Action` as `@myFunction`.

Example: `Action = @myFunction`

Data Types: `function_handle`

RequiredIterationArtifactType — Type of artifact

"sl_model_file" | "m_file" | "ma_config_file" | "sl_test_case" | ...

Type of artifact, specified as one or more of the values listed in this table. To specify multiple values, use an array. The build system uses the artifact type to identify and categorize the different file types and modeling constructs in your project.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file

Category	Artifact Type	Description
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result

Category	Artifact Type	Description
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "sl_model_file"

Example: ["sl_model_file "zc_file"]

Licenses – List of licenses that task requires

string array

List of licenses that the task requires, specified as a string array.

Example: Licenses = ["matlab_report_gen" "simulink_report_gen"]

Data Types: string

AllLicenseRequired – Setting to require all licenses for task

true or 1 (default) | false or 0

Setting to require all licenses for task, specified as a numeric or logical 1 (true) or 0 (false). By default, all licenses in the Licenses property of the task are required for the task to run. Specify 0 (false) if the task can run without all licenses listed in the Licenses property.

Example: AllLicenseRequired = false

Data Types: logical

DescriptionText – Task description

string

Task description, specified as a string.

Example: "This task runs myScript."

Data Types: string

DescriptionCSH – Path to task documentation

string

Path to task documentation, specified as a string.

Example: DescriptionCSH =
fullfile(pwd,"taskHelpFiles","myTaskDocumentation.pdf")

Data Types: string

Output Arguments

taskObj — Task object

`padv.Task` object

Task object, returned as a `padv.Task` object.

For more information, see `padv.Task`.

See Also

`createprocess` | `getprocess` | `padv.Task` | `padv.ProcessModel` | `runprocess`

Topics

“Automate and Run Tasks with Process Advisor”

“Modify Default Process Model to Fit Your Process”

“Overview of Process Model”

“Add Tasks to Process”

“Reconfigure Task Behavior”

“Define Task Relationships”

padv.Query

Select set of artifacts from project

Description

This object requires CI/CD Automation for Simulink Check. A `padv.Query` object represents a query that you can use to select a set of artifacts from a project. Use the input arguments to define the set of artifacts that the query selects. Queries can either be function-based or class-based. Use `FunctionHandle` to specify a function for a function-based query or use inheritance for a class-based query. For more information, see “Find Artifacts with Queries” and “Create Custom Queries”.

Creation

Syntax

```
Q = padv.Query(Name)
Q = padv.Query( ___, Name = Value)
```

Description

`Q = padv.Query(Name)` creates a query object with the name `Name`.

`Q = padv.Query(___, Name = Value)` specifies query properties using one or more name-value arguments. For example, `DefaultArtifactType = "sl_model_file"` changes the default artifact type for the query from a generic output file, `"padv_output_file"`, to a model file, `"sl_model_file"`.

Input Arguments

Name — Unique identifier for query

character vector | string

Unique identifier for query, specified as character vector or string. You can only specify a query name when you create a query object. You cannot change the query name after you create the query object.

Each query in the process model must have a unique name.

Example: `"CustomQueryForArtifacts"`

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `DefaultArtifactType = "sl_model_file"`

Title – Human-readable name for query

character vector | string

Human-readable name for query, specified as character vector or string.

Example: Title = "Custom Query for Artifacts"

Data Types: char | string

DefaultArtifactType – Expected artifact type

"padv_output_file" (default) | valid value for the Type property of a padv.Artifact object

Expected artifact type, specified as one or more of the values listed in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram

Category	Artifact Type	Description
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

`padv.Task` objects use the `DefaultArtifactType` to confirm that the artifacts output by the query are the types of artifacts required by the `padv.Task` object.

When you use the `run` function on a query object, the `DefaultArtifactType` is the default value for artifacts returned by the function.

Example: `DefaultArtifactType = "sl_model_file"`

Parent — Initial query run before iteration query

`padv.Query` object | Name of `padv.Query` object

Initial query run before iteration query, specified as either a `padv.Query` object or the Name of a `padv.Query` object. When you specify a `padv.Query` object as the iteration query for a task, the Parent query is the initial query that the build system runs before running the specified iteration query.

For example, the built-in query `padv.builtin.query.FindModelsWithTestCases` has the Parent query `padv.builtin.query.FindModels`. If you specify `padv.builtin.query.FindModelsWithTestCases` as the iteration query for a task, you are specifying that you want the task to run once for each model with a test case. The build system runs the Parent query `padv.builtin.query.FindModels` first, to find the models in the project, and then the build system runs the iteration query `padv.builtin.query.FindModelsWithTestCases` to find the models with test cases.

The build system ignores the Parent query when you specify a query as an input query or dependency query for a task.

Example: `"padv.builtin.query.FindModels"`

ShowFileExtension — Show file extensions for returned artifacts

`0 (false) | 1 (true)`

Show file extensions in the `Alias` property of returned artifacts, specified as a numeric or logical `1 (true)` or `0 (false)`. The `Alias` property controls the display name for the artifact in the **Tasks** column in Process Advisor.

By default, queries strip file extensions from the `Alias` property of each task iteration artifact. To show file extensions for all artifacts in the **Tasks** column, select the project setting **Show file extensions**. To keep file extensions in the results for a specific query, specify the query property `ShowFileExtension` as `true`.

Example: `true`

Data Types: `logical`

SortArtifacts — Setting for automatically sorting artifacts by address

`true or 1 (default) | false or 0`

Setting for automatically sorting artifacts by address, specified as a numeric or logical `1 (true)` or `0 (false)`. When a query returns artifacts, the artifacts should be in a consistent order. By default, the build system sorts artifacts by the artifact address.

Alternatively, you can sort artifacts in a different order by overriding the internal `sortArtifacts` method in a subclass that defines a custom sort behavior.

For more information, see “Sort Artifacts in Specific Order”.

The build system automatically calls the `sortArtifacts` method when using the process model. The `sortArtifacts` method expects two input arguments: a `padv.Query` object and a list of `padv.Artifact` objects returned by the run function. The `sortArtifacts` method should return a list of sorted `padv.Artifact` objects.

Example: `SortArtifacts = false`

Data Types: `logical`

FunctionHandle — Handle to function that runs when you run query object

`function_handle`

Handle to function that runs when you run query object, specified as a `function_handle`.

When you call the run function on a query object, run runs the function specified by the `function_handle`.

Example: `FunctionHandle = @FunctionForQuery`

Data Types: `function_handle`

See Also

`addInputQueries` | `padv.ProcessModel` | `padv.Task`

Topics

“Overview of Process Model”

“Find Artifacts with Queries”

“Create Custom Queries”

“Reconfigure Task Behavior”

“Automate and Run Tasks with Process Advisor”

run

Namespace: padv

Return artifacts from query

Syntax

```
artifacts = run(queryObj)
artifacts = run(queryObj, inputArtifact)
```

Description

`artifacts = run(queryObj)` returns the artifacts in the project folder that match the criteria specified by the query `queryObj`. For more information, see “Find Artifacts with Queries”.

This function requires CI/CD Automation for Simulink Check.

Typically, you use queries inside your process model and the build system automatically runs the queries to find artifacts, but you can manually call the `run` function to run a query outside of your process model to confirm which artifacts the query returns.

`artifacts = run(queryObj, inputArtifact)` returns the artifacts in the project folder that match the criteria specified by the query `queryObj` and are associated with the artifact `inputArtifact`. If you use the query as an iteration query or dependency query, the build system can use `inputArtifact` to determine the scope of the artifacts that the query returns, which can be helpful for queries that need an input artifact from a parent query.

Examples

Test Query Outside Process Model

Although you typically use queries inside your process model, you can run queries outside of your process model to confirm which artifacts the query returns.

- 1 Open a project. For this example, you can open the Process Advisor example project.

```
processAdvisorExampleStart
```

- 2 Create an instance of a query. For this example, you can create an instance of the built-in query `padv.builtin.query.FindArtifacts`. You can use the arguments of the query to filter the query results. For example, you can use the `IncludeLabel` argument to have the query only return artifacts that use the `Design` project label from the `Classification` project label category.

```
q = padv.builtin.query.FindArtifacts(...
IncludeLabel = {'Classification', 'Design'});
```

- 3 Run the query and inspect the array of artifacts that the query returns.

```
artifacts = run(q)
```

```
artifacts =
    1x24 Artifact array with properties:
        Type
        Parent
        ArtifactAddress
        Alias
```

Input Arguments

queryObj — Query object

`padv.Query` object | built-in query object

Query object, specified as a `padv.Query` object, built-in query object, or an object whose class inherits from either the `padv.Query` class or a built-in query class.

Example: `q = padv.Query("myQueryName")`

Example: `q = padv.builtin.query.FindArtifacts`

inputArtifact — Input artifact that query needs

`padv.Artifact`

Input artifact that the query needs, specified as a `padv.Artifact` object.

Output Arguments

artifacts — Artifacts that query returns

`padv.Artifact`

Artifacts that query returns, returned as an array of `padv.Artifact` objects.

See Also

`addInputQueries` | `padv.ProcessModel` | `padv.Query` | `padv.Task`

Topics

“Overview of Process Model”

“Find Artifacts with Queries”

“Create Custom Queries”

“Reconfigure Task Behavior”

“Automate and Run Tasks with Process Advisor”

padv.Subprocess

Group of tasks and subprocesses in process

Description

This object requires CI/CD Automation for Simulink Check. A `padv.Subprocess` object represents a group of tasks and subprocesses in your process. You can have multiple processes inside your process model. In your process model, use the object functions `addTask` and `addSubprocess` to group tasks and other subprocesses inside your subprocess. You can specify a dependency between tasks or a desired execution order by using either `dependsOn` or `runsAfter`. Use `dependsOn` when a subprocess cannot start without another task or subprocess finishing first. Otherwise, if you only want to specify a preferred execution order, you can use `runsAfter` instead.

Creation

Syntax

```
subprocessObject = padv.Subprocess(Name)  
subprocessObject = padv.Subprocess( ____,Name=Value)
```

Description

`subprocessObject = padv.Subprocess(Name)` represents a subprocess, named `Name`, in a process. Each subprocess in a process must have a unique `Name`.

`subprocessObject = padv.Subprocess(____,Name=Value)` sets properties using one or more name-value arguments. For example, `padv.Subprocess("mySubprocess",Title="My Subprocess")` creates a subprocess with the title `My Subprocess` in Process Advisor.

Specify optional pairs of arguments as `Name1=Value1, . . . ,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Properties

Name — Unique identifier for subprocess

string

Unique identifier for subprocess, returned as a string. When you specify the `Name`, you specify the `Name` property of the subprocess object.

Each subprocess must have a unique `Name`.

Example: `padv.Subprocess("mySubprocess")`

Data Types: string

Title — Human readable name that appears in Process Advisor app

string

Human readable name that appears in the **Tasks** column of the Process Advisor app, returned as a string. By default, the Process Advisor app uses the Name property of the subprocess as the Title.

Example: `padv.Subprocess("mySubprocess",Title = "My Subprocess")`

Data Types: string

DescriptionText — Subprocess description

string

Subprocess description, returned as a string.

Example: `padv.Subprocess("mySubprocess",DescriptionText = "This is my subprocess.")`

Data Types: string

DescriptionCSH — Path to subprocess documentation

string

Path to subprocess documentation, returned as a string.

Example: `padv.Subprocess("mySubprocess",DescriptionCSH = fullfile(pwd,"subprocessHelpFiles","mySubprocessDocumentation.pdf"))`

Data Types: string

RequiredIterationArtifactType — Type of artifact

"sl_model_file" | "m_file" | "zc_file" | ...

Type of artifact, specified as one or more of the values listed in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table

Category	Artifact Type	Description
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
"sl_subsystem"	Subsystem	
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
"sl_test_seq"	Test Sequence	

Category	Artifact Type	Description
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "m_file"

Example: ["sl_model_file" "zc_file"]

LaunchToolAction – Function that launches a tool

function handle

Function that launches a tool, returned as the function handle.

When the property `LaunchToolAction` is specified, you can point to the subprocess in the Process Advisor app and click the ellipsis (...) and then **Open Tool Name** to open the tool associated with the subprocess.

Example: `padv.Subprocess("mySubprocess", LaunchToolAction = @openTool)`

Data Types: `function_handle`

LaunchToolText – Description of action that LaunchToolAction property performs

"Launch Tool" (default) | string scalar

Description of the action that the `LaunchToolAction` property performs, returned as a string scalar.

Example: `padv.Subprocess("mySubprocess", LaunchToolAction = @openTool, LaunchToolText = "Open tool.")`

Data Types: `string`

Enabled – Controls if the padv.Subprocess is enabled in the process model

true or 1 (default) | false or 0

Controls if the `padv.Subprocess` is enabled in the process model, returned as a numeric or logical 1 (true) or 0 (false).

Example: `padv.Subprocess("mySubprocess", Enabled = false)`

Data Types: `logical`

DryRunLicenseCheckout – Dry run checks out product license

`logical.empty` (default) | true or 1 | false or 0

Dry run checks out product license, returned as a numeric or logical 1 (true) or 0 (false).

For more information, see "Dry Run Tasks to Test Process Model".

Example: `true`

Data Types: `logical`

OutputDirectory – Location for standard outputs that tasks in subprocess produce

string

Location for standard outputs that tasks in the subprocess produce, specified as a string.

Example: `fullfile("folder", "subfolder")`

Data Types: string

CacheDirectory — Location for additional cache files that tasks in subprocess produce
string

Location for additional cache files that tasks in the subprocess produce, specified as a string. The cache directory can contain temporary files that do not need to be either saved in the task results or archived by a CI system.

Example: `fullfile("folder", "subfolder")`

Data Types: string

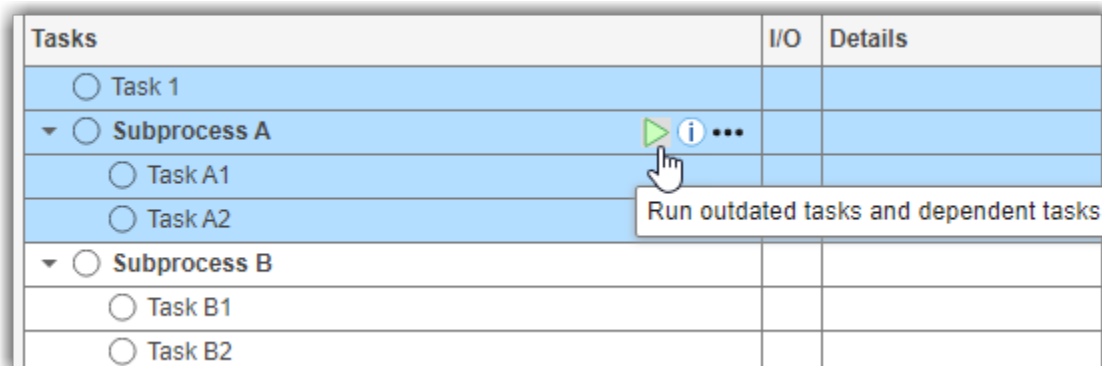
Object Functions

addTask	Add task to subprocess object. See “Group Tasks with Subprocesses”. addTask(subprocessObject, taskNameOrInstance)
addSubprocess	Add subprocess to subprocess object. addSubprocess(subprocessObject, subprocessNameOrInstance)
dependsOn	Specify tasks or subprocesses that the subprocess object depends on. See “Group Tasks with Subprocesses”. dependsOn(subprocessObject, DEPENDENCIES)
runsAfter	Specify tasks or subprocesses that the subprocess object runs after. runsAfter(subprocessObject, PREDECESSORS)

Examples

Group Tasks Inside Subprocess

You can use a subprocess to group related tasks, create a hierarchy of tasks, and share parts of a process. A *subprocess* is a self-contained sequence of tasks, inside a process or other subprocess, that can run standalone.



To group the tasks in your process model, in the process model, add a subprocess by using `addSubprocess` on your process model object.

```
spA = pm.addSubprocess("Subprocess A");
```

Add your tasks directly to the subprocess by using `addTask`.

```
tA1 = spA.addTask("Task A1");
tA2 = spA.addTask("Task A2");
```

Note You do not need to add the task to both the subprocess and process model.

Specify the relationship between the tasks and subprocesses in your process.

You can use either `dependsOn` or `runsAfter` to define a relationship.

For example, the following process model defines a process in which Task 1 runs, then Subprocess A, and then Subprocess B.

```
function processmodel(pm)
    % Defines the project's processmodel

    arguments
        pm padv.ProcessModel
    end

    t1 = pm.addTask("Task 1");

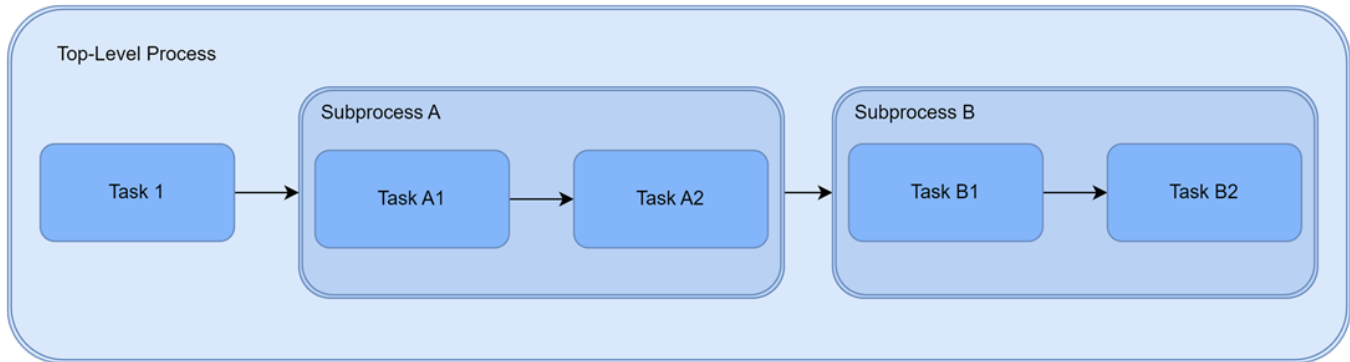
    spA = pm.addSubprocess("Subprocess A");
        tA1 = spA.addTask("Task A1");
        tA2 = spA.addTask("Task A2");
    spB = pm.addSubprocess("Subprocess B");
        tB1 = spB.addTask("Task B1");
        tB2 = spB.addTask("Task B2");

    % Relationships
    spA.dependsOn(t1);
        tA2.dependsOn(tA1);
    spB.dependsOn(spA);
        tB2.dependsOn(tB1);

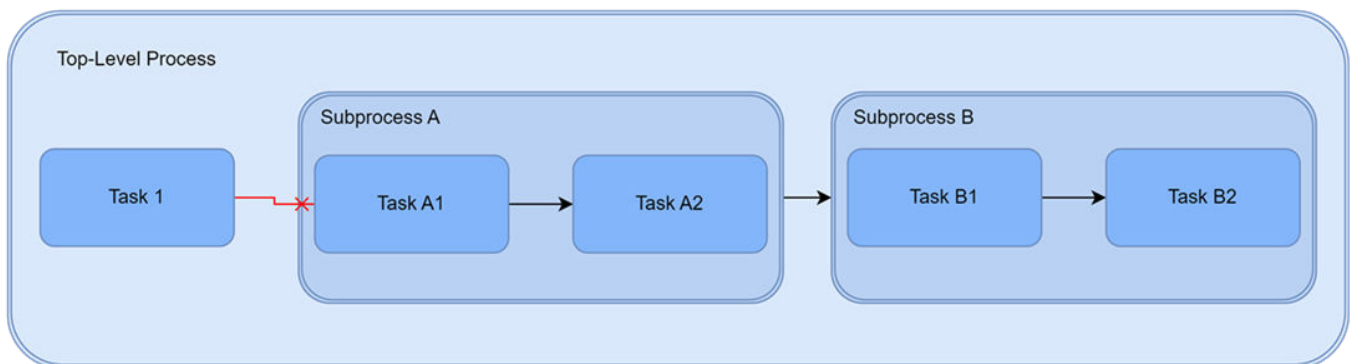
end
```

The build system executes each of the tasks inside a subprocess before exiting the subprocess.

The following diagram shows a graphical representation of the relationships defined by that process model.



Note Relationships cannot cross any subprocess boundaries. For example, in this process model, you cannot directly specify that Task A1 depends on Task 1 because that relationship would enter into Subprocess A, crossing the subprocess boundary.



See Also

`padv.ProcessModel` | `padv.Task`

Topics

“Group Tasks with Subprocesses”

“Manage Multiple Build and Verification Workflows Using Processes”

padv.Task Class

Namespace: padv

Single step in process

Description

A `padv.Task` object represents a single step in a `padv.ProcessModel` process. For example, a `padv.Task` object could represent a step like checking modeling standards, running tests, generating code, or performing a custom action. `padv.Task` objects can accept project artifacts as inputs, perform actions, generate assessments, and return project artifacts as outputs. You can add a task to your process model by using the function `addTask`. You can specify task inputs by using `addInputQueries`. You can specify a dependency between tasks or a desired execution order by using either `dependsOn` or `runsAfter`. Use `dependsOn` when a task cannot start without another task finishing first. Otherwise, if you only want to specify a preferred execution order, you can use `runsAfter` instead. You can execute tasks as part of a pipeline. Use the `runprocess` function to generate and run a pipeline of tasks. For more information, see “Overview of Process Model”.

Creation

Syntax

```
taskObject = padv.Task(Name)
taskObject = padv.Task( ____, Name=Value)
```

Description

`taskObject = padv.Task(Name)` represents a task, named `Name`, in a `padv.ProcessModel` process. Each task object in a process must have a unique `Name`.

`taskObject = padv.Task(____, Name=Value)` sets properties using one or more name-value arguments. For example, `padv.Task("myTask", IterationQuery=padv.builtin.query.FindModels)` creates a task object named `myTask` that runs once for each model.

Specify optional pairs of arguments as `Name1=Value1, . . . , NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Properties

Name — Unique identifier for task in process

string

Unique identifier for task in process, specified as a string. When you specify the `Name`, you specify the `Name` property of the task object.

Each task in the process model must have a unique **Name**. After you specify a **Name** for a `padv.Task` object, you cannot change the **Name**.

Example: `padv.Task("myTask")` creates a task with the **Name** `myTask`

Data Types: `string`

Title — Human readable name that appears in Process Advisor app

`string`

Human readable name that appears in the **Tasks** column of the Process Advisor app, specified as a `string`. By default, the Process Advisor app uses the **Name** property of the task as the **Title**.

Example: `padv.Task("myTask", Title = "My Task")`

Data Types: `string`

DescriptionText — Task description

`string`

Task description, specified as a `string`.

Example: `padv.Task("myTask", DescriptionText = "This is my task.")`

Data Types: `string`

DescriptionCSH — Path to task documentation

`string`

Path to task documentation, specified as a `string`.

Example: `padv.Task("myTask", DescriptionCSH = fullfile(pwd, "taskHelpFiles", "myTaskDocumentation.pdf"))`

Data Types: `string`

Action — Function that task can run

`function handle`

Function that task can run, specified as the function handle.

If the task is defined in a function, the build system runs the function specified by **Action**. If the task is defined in a class, the build system ignores the **Action** and runs the `run` method for the class instead. The built-in tasks are defined in classes, so the build system calls the `run` method for those tasks.

Example: `padv.Task("myTask", Action = @myFunction)`

Data Types: `function_handle`

DryRunAction — Function that task can use during dry-run

`function handle`

Function that task can use during dry-run, specified as the function handle.

If the task is defined in a function, the build system dry-runs by calling the function specified by **DryRunAction**. If the task is defined in a class, the build system ignores the **DryRunAction** and dry-runs by calling the `dryRun` method for the class instead. The built-in tasks are defined in classes, so the build system calls the `dryRun` method for those tasks.

Example: `padv.Task("myTask",DryRunAction = @myFunction)`

Data Types: `function_handle`

DryRunLicenseCheckout – Dry-runs check out product licenses associated with tasks in process

`logical.empty` (default) | `true` or `1` | `false` or `0`

Dry-runs check out product licenses associated with tasks in process, returned as a numeric or logical `1` (true) or `0` (false).

To perform a dry-run, you can specify the `runprocess` argument `DryRun` as `true`.

Example: `true`

Data Types: `logical`

RequiredIterationArtifactType – Artifact type that task can run on

`"sl_model_file"` | `"m_file"` | `"zc_file"` | ...

Type of artifact, specified as one or more of the values listed in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart

Category	Artifact Type	Description
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
System Composer	"sl_subsystem"	Subsystem
	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
Tests	"zc_file"	System Composer architecture file
	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
"sl_test_suite"	Simulink Test suite	
"sl_test_suite_result"	Simulink Test suite result	

Example: `padv.Task("myTask", RequiredIterationArtifactType = "sl_model_file")`

Data Types: string

IterationQuery – Artifacts that task iterates over

`padv.Query object | name of padv.Query object`

Artifacts that task iterates over, specified as a `padv.Query` object or the name of a `padv.Query` object. By default, task objects run one time and are associated with the project. When you specify `IterationQuery`, the task runs one time *for each* artifact returned by the `padv.Query`. In the Process Advisor app, the artifacts returned by `IterationQuery` appear under task title.

For example, if the `IterationQuery` for a task finds three models, `Model_A`, `Model_B`, and `Model_C`, the build system creates three task iterations under the title of the task in the **Tasks** column.

For more information, see “Overview of Process Model” and “Find Artifacts with Queries”.

Example: `padv.Task("myTask", IterationQuery = padv.builtin.query.FindModels)`

Data Types: `string`

InputDependencyQuery — Artifact dependencies for task inputs

`padv.Query` object | name of `padv.Query` object

Artifact dependencies for task inputs, specified as a `padv.Query` object or the name of a `padv.Query` object.

The build system runs the query specified by `InputDependencyQuery` to find the dependencies for the task inputs, since those dependencies can impact if task results are up-to-date. Typically, you specify `InputDependencyQuery` as `padv.builtin.query.GetDependentArtifacts` to get the dependent artifacts for each task input. For example, if you specify a model as an input to a task and you specify `InputDependencyQuery` as `padv.builtin.query.GetDependentArtifacts`, the build system can find artifacts, such as data dictionaries, that the model uses.

For more information, see “Overview of Process Model” and “Find Artifacts with Queries”.

Example: `InputDependencyQuery = padv.builtin.query.GetDependentArtifacts`

Licenses — List of licenses that task requires

`string` array

List of licenses that the task requires, specified as a `string` array.

Example: `padv.Task("myTask", Licenses = ["matlab_report_gen" "simulink_report_gen"])`

Data Types: `string`

LaunchToolAction — Function that launches a tool

function handle | cell array of `function_handle` objects

Function that launches a tool, specified as the function handle or a cell array of `function_handle` objects. For each action that you specify in `LaunchToolAction`, you must have corresponding text specified in `LaunchToolText`.

When the property `LaunchToolAction` is specified, you can point to the task in the Process Advisor app and click the ellipsis (...) and then **Open Tool Name** to open the tool associated with the task.

For tasks that are not built-in tasks, the task options show the ellipsis (...) and then **Launch Tool**.

Example: `@openTool`

Example: `{@openToolA,@openToolB}`

Data Types: `cell` | `function_handle`

LaunchToolText — Description of action that LaunchToolAction property performs

"Launch Tool" (default) | `string`

Description of the action that the `LaunchToolAction` property performs, specified as a string. For each action that you specify in `LaunchToolAction`, you must have corresponding text specified in `LaunchToolText`.

Example: "Open Tool"

Example: ["Open Tool A", "Open Tool B"]

Data Types: `string`

Enabled — Controls if the `padv.Task` is enabled in the process model

`true` or `1` (default) | `false` or `0`

Controls if the `padv.Task` is enabled in the process model, specified as a numeric or logical `1` (`true`) or `0` (`false`).

For an example, see "Disable Task in Process Model" on page 2-59.

Example: `padv.Task("myTask", Enabled = false)`

Data Types: `logical`

AlwaysRun — Always force task to run, even if the task results are already up to date

`false` or `0` (default) | `true` or `1`

Always force task to run, even if the task results are already up to date, specified as a numeric or logical `0` (`false`) or `1` (`true`).

Example: `padv.Task("myTask", AlwaysRun = true)`

Data Types: `logical`

TrackOutputs — Track changes to output files

`true` or `1` (default) | `false` or `0`

Track changes to output files, specified as a numeric or logical `1` (`true`) or `0` (`false`).

By default, the build system tracks changes to outputs files from tasks unless the files are outside the project. If you make a change to an output file, the task status and results are marked as outdated. If you specify `TrackOutputs` as `false`, changes that you make to the task output files do not make the task status and results outdated.

For more information, see "Turn Off Change Tracking for Task Outputs" on page 2-59 and "Exclude Files from Change Tracking in Process Advisor".

Example: `false`

Data Types: `logical`

InputQueries — Inputs to task

`padv.Query` object | name of `padv.Query` object | array of `padv.Query` objects

Inputs to the task, specified as:

- a `padv.Query` object
- the name of `padv.Query` object
- an array of `padv.Query` objects
- an array of names of `padv.Query` objects

By default, `padv.Task` does not have inputs. When you specify `InputQueries`, the task uses the artifacts returned by the specified query or queries as inputs. Suppose a task runs once for each model in the project and you want to provide the models as inputs to the task. If you specify `InputQueries` as the built-in query `padv.builtin.query.GetIterationArtifact`, the query returns each artifact that the tasks iterates over, which in this example is each of the models in the project.

To add an input query to an existing task object, you can use `addInputQueries`.

For more information, see “Overview of Process Model” and “Find Artifacts with Queries”.

```
Example: padv.Task("myTask", InputQueries =
padv.builtin.query.GetIterationArtifact)
```

OutputDirectory — Location for standard outputs that the task produces

"" (default) | string array

Location for standard outputs that the task produces, specified as a string.

Built-in tasks automatically specify `OutputDirectory`. If you do not specify `OutputDirectory` for a custom task, the build system stores task outputs in the `DefaultOutputDirectory` specified by `padv.ProcessModel`.

Data Types: `string`

CacheDirectory — Location for additional cache files that the task generates

string array

Location for additional cache files that the task generates, specified as a string. The cache directory can contain temporary files that do not need to be either saved in the task results or archived by a CI system.

Data Types: `string`

CISupportOutputsForTask — Type of CI-compatible result files that task generates when skipped

"JUnit" (default) | ""

Type of CI-compatible result files that the task itself generates when skipped, specified as either:

- "JUnit" — JUnit-style XML report for task results.
- "" — None. The build system generates a JUnit-style XML report for the task instead.

CISupportOutputsByTask — Type of CI-compatible result files that task generates when run

"" (default) | "JUnit"

Type of CI-compatible result files that the task itself generates when run, specified as either:

- "JUnit" — JUnit-style XML report for task results.
- "" — None. The build system generates a JUnit-style XML report for the task instead.

Methods

Object Functions

Object Function	Description
addInputQueries	<p>Add the input artifacts returned by <code>inputQueries</code> as inputs to the task represented by <code>taskObj</code>.</p> <p><code>addInputQueries(taskObj, inputQueries)</code></p>
dependsOn	<p>Create a dependency between a task, <code>taskObj</code>, and dependencies, <code>dependencies</code>.</p> <p><code>dependsOn(taskObj, dependencies)</code></p> <p>The build system always runs the <code>dependencies</code> before the <code>taskObj</code>. Use <code>dependsOn</code> when one task cannot start without another task finishing first. Otherwise, if you only want to specify a preferred task execution order, you can use <code>runsAfter</code> instead.</p>

Object Function	Description
dryRun	<p>Dry run the task to validate task inputs and generate representative task outputs without actually running the task. The <code>dryRun</code> method runs on the current task instance <code>this</code> with task inputs <code>inputArtifacts</code> and returns a task result <code>result</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. By default, the task returns the default dry run results specified by the <code>padv.ProcessModel</code> property <code>DefaultDryRunResults</code>. If you inherit from this class, make sure to use the same method signature inside your custom task:</p> <pre>function result = dryRun(taskObj, inputArtifacts) ... end</pre> <p>For more information, see “Dry Run Tasks to Test Process Model”.</p> <p>Inside the <code>dryRun</code> method for a task, you can resolve tokens, like <code>\$ITERATIONARTIFACT\$</code>, to their absolute path by using the <code>resolvePath</code> method. For example, <code>reportPath = convertStringsToChars(taskObj.resolvePath(taskObj.ReportPath));</code>. For more information on tokens, see “Dynamically Resolve Paths with Tokens”.</p>
resolvePath	<p>Resolve path that contains tokens.</p> <pre>resolvedPath = resolvePath(taskObj, pathWithTokens)</pre> <p>You can only call this function during the execution of the <code>run</code> method or <code>dryRun</code> method.</p> <p>For more information on tokens, see “Dynamically Resolve Paths with Tokens”.</p>

Object Function	Description
run	<p>Run task represented by <code>taskObj</code>.</p> <pre>taskResult = run(taskObj)</pre> <p>If the task requires inputs, specify the inputs using <code>inputArtifacts</code>.</p> <pre>taskResult = run(taskObj,inputArtifacts)</pre> <p>Inside the <code>run</code> method for a task, you can resolve tokens, like <code>\$ITERATIONARTIFACT\$</code>, to their absolute path by using the <code>resolvePath</code> method. For example, <code>reportPath = convertStringsToChars(taskObj.resolvePath(taskObj.ReportPath))</code>; For more information on tokens, see “Dynamically Resolve Paths with Tokens”.</p>
runsAfter	<p>Specify the preferred execution order for tasks by specifying the tasks, <code>predecessors</code>, that a task, <code>taskObj</code>, should run after.</p> <pre>runsAfter(taskObj,predecessors)</pre> <p>When you run your process, the build system runs the <code>predecessors</code> before the <code>taskObj</code> when possible. But if you force run the <code>taskObj</code>, the build system runs that task independently. Use <code>runsAfter</code> for tasks that you prefer to run in a specific order, but do not have a strict dependency. If a task must run before another task to run successfully, use <code>dependsOn</code> instead.</p>

Examples


Create Task Objects and Add Tasks to Process Model

You can use `padv.Task` to create task objects and then use the `addTask` function to add the task objects to the `padv.ProcessModel` object.

Open the Process Advisor example project.

```
processAdvisorExampleStart
```

The model `AHRS_Voter` opens with the Process Advisor pane to the left of the Simulink canvas.

In the Process Advisor pane, click the **Edit process model**  button to open the `processmodel.m` file for the project.

Replace the contents of the `processmodel.m` file with this code:

```
function processmodel(pm)
    arguments
```



```

        pm padv.ProcessModel
    end

    taskA = padv.Task("taskA");
    taskB = padv.Task("taskB");

    runsAfter(taskB,taskA);

    addTask(pm,taskA);
    addTask(pm,taskB);

end

```

This code uses `padv.Task` to create two task objects: `taskA` and `taskB`.

The object function `runsAfter` specifies that `taskB` should run after `taskA`.

The function `addTask` adds the task objects to the `padv.ProcessModel` object.

Disable Task in Process Model

If you want to disable a task, you can set the task property `Enabled` to `false`. To disable the task for a specific process, you can set the task property inside the process model. Note that by default the build system automatically disables a task if the required licenses are not available in the current MATLAB session.

Open a project. For this example, you can open the Process Advisor example project.

```
processAdvisorExampleStart
```

Edit the process model to use the following process model instead.

```

function processmodel(pm)
    % Defines the project's processmodel

    arguments
        pm padv.ProcessModel
    end

    % Add built-in task for Checking Modeling Standards
    maTask = pm.addTask(padv.builtin.task.RunModelStandards);

    % Disable task
    maTask.Enabled = false;

end

```

This process model specifies the task property `Enabled` as `false`.

In Process Advisor, click **Refresh Tasks**.

The **Check Modeling Standards** task appears dimmed in the process and the run button is unavailable.

Turn Off Change Tracking for Task Outputs

If you do not want the build system to mark a task as outdated when you make changes to task outputs, you can turn off change tracking for those task outputs. For that task, you specify the task property `TrackOutputs` as `false`.

Open a project. For this example, you can open the Process Advisor example project.

```
processAdvisorExampleStart
```

Edit the process model to use the following process model instead.

```
function processmodel(pm)
    % Defines the project's processmodel

    arguments
        pm padv.ProcessModel
    end

    % Add built-in task for Checking Modeling Standards
    maTask = pm.addTask(padv.builtin.task.RunModelStandards);

    % Turn off change tracking for task
    maTask.TrackOutputs = false;

end
```

This process model specifies the task property `TrackOutputs` as `false`.

When you run the task, the Process Advisor **I/O** column shows the outputs as **Untracked**. If you make a change to an untracked file, the build system does not mark the task as outdated.

The screenshot shows the Process Advisor interface with a table of tasks and a details popup for the 'AHRS_Voter' task.

Tasks	I/O	Details
▼ ● Check Modeling Standards	📄	✓ 28 △ 4
✓ AHRS_Voter	📄	✓ 28 △ 4
○ Actuator_Control		
○ Flight_Control		
○ InnerLoop_Control		
○ OuterLoop_Control		
▼ ○ Detect Design Errors		
○ AHRS_Voter		
○ Actuator_Control		
○ Flight_Control		
○ InnerLoop_Control		
○ OuterLoop_Control		

The details popup for 'AHRS_Voter' shows:

- Outputs: AHRS_Voter_ModelAdvisor.html
- Inputs: AHRS_Voter
- Dependencies: AHRS_Voter, bus_types, csSingleInstance, DD_AHRS_Voter, processmodel.m

For your project, make sure to review untracked outputs to check that those untracked files do not need to be tracked to maintain the task status and result information that you need for your project.

See Also

[addInputQueries](#) | [addTask](#) | [dependsOn](#) | [padv.ProcessModel](#) | [padv.TaskResult](#) | [runprocess](#) | [runsAfter](#)

Topics

[“Automate and Run Tasks with Process Advisor”](#)

[“Overview of Process Model”](#)

[“Add Tasks to Process”](#)

[“Reconfigure Task Behavior”](#)

[“Define Task Relationships”](#)

[“Create Custom Tasks”](#)

addInputQueries

Namespace: padv

Add input artifacts as inputs to task

Syntax

```
addInputQueries(taskObj, inputQueries)
```

Description

`addInputQueries(taskObj, inputQueries)` adds the input artifacts returned by `inputQueries` as inputs to the task represented by `taskObj`.

If the task already has input queries specified, `addInputQueries` adds `inputQueries` to the list of input queries in the `InputQueries` property.

This function requires CI/CD Automation for Simulink Check.

Examples

Add Inputs to Task

Use `addInputQueries` to specify the models in the project as inputs to a task.

Create a new `padv.Task` object `myTaskObj` that represents a task named `runForEachModel`.

```
myTaskObj = padv.Task("runForEachModel");
```

By default, the task does not have inputs.

Use the function `addInputQueries` to add the built-in query `padv.builtin.query.FindModels` as the input query for the task.

```
addInputQueries(myTaskObj, padv.builtin.query.FindModels);
```

When you run the task defined by `myTaskObj`, the query `padv.builtin.query.FindModels` finds each model in the project and provides the models as the input artifacts for the task.

Input Arguments

taskObj — Task object that represents task

`padv.Task` object

Task object that represents a task, specified as a `padv.Task` object.

Example: `myTaskObj = padv.Task("myTask");`

inputQueries — Queries that get input artifacts for task

`padv.Query` object | array of `padv.Query` objects

A query or queries that get the input artifacts for a task, specified as a `padv.Query` object or an array of `padv.Query` objects. Each artifact that the query or queries return becomes an input to the task.

For example, if you specify the `InputQueries` property for a task as the query `padv.builtin.query.FindModels`, the query returns each model and the models become input artifacts for the task.

For more information, see “Overview of Process Model” and “Find Artifacts with Queries”.

Example: `addInputQueries(myTaskObj, padv.builtin.query.FindModels)`

Example: `addInputQueries(myTaskObj, [padv.builtin.query.GetIterationArtifact, padv.builtin.query.GetDependentArtifacts])`

See Also

`padv.Task` | `padv.TaskResult`

Topics

“Add Tasks to Process”

“Reconfigure Task Behavior”

“Create Custom Tasks”

“Find Artifacts with Queries”

“Overview of Process Model”

dependsOn

Namespace: padv

Create dependency between tasks

Syntax

```
dependsOn(taskObj,dependencies)  
dependsOn( ____,Name=Value)
```

Description

`dependsOn(taskObj,dependencies)` creates a dependency between `taskObj` and `dependencies`. `taskObj` runs only after the tasks specified by `dependencies` run and return a task status.

This function requires CI/CD Automation for Simulink Check.

`dependsOn(____,Name=Value)` specifies how the build system handles dependencies using one or more `Name=Value` arguments.

Note You can specify the relationship between two tasks as *either* a `dependsOn` relationship or a `runsAfter` relationship, but not both.

If you define multiple relationships between the same tasks, the build system only uses the most recent relationship and ignores previous relationships. For example:

```
dependsOn(taskA, taskB)  
dependsOn(taskB, taskA) % build system only uses this relationship
```

Examples

Create Dependency Between Two Tasks

Use the `dependsOn` function to create a dependency between two tasks in a process model.

Open the Process Advisor example project.

```
processAdvisorExampleStart
```

Open the `processmodel.m` file. The file is at the root of the project.

Replace the contents of the `processmodel.m` file with the following code:

```
function processmodel(pm)  
    arguments  
    pm padv.ProcessModel  
end
```

```

TaskA = padv.Task("TaskA");
TaskB = padv.Task("TaskB");

dependsOn(TaskB,TaskA);

addTask(pm,TaskA);
addTask(pm,TaskB);

```

end

This code uses `padv.Task` to create two task objects: `TaskA` and `TaskB`.

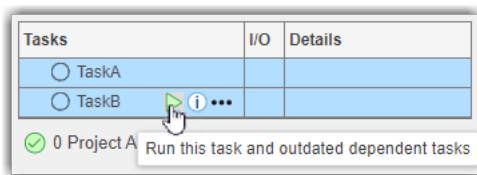
The object function `dependsOn` specifies that `TaskB` depends on `TaskA`.

The function `addTask` adds the task objects to the `padv.ProcessModel` object.

Open the Process Advisor app. In the MATLAB Command Window, enter:

```
processAdvisorWindow
```

In the **Tasks** column, point to the run button for **TaskB**. The Process Advisor app automatically highlights both tasks since **TaskA** is a dependent task. If you click the run button for **TaskB**, **TaskA** will run before **TaskB**.



Input Arguments

taskObj — Task object that represents task

`padv.Task` object

Task object that represents a task, specified as a `padv.Task` object.

Example: `myTaskObj = padv.Task("myTask");`

dependencies — Tasks that need to run before `taskObj` runs

string | character vector | `padv.Task` object

Tasks that need to run before `taskObj` runs, specified as either:

- The name of a task, specified as a string or character vector.
- A `padv.Task` object.

Example: `dependsOn(TaskB,"TaskA")`

Example: `dependsOn(TaskB,TaskA)`

Data Types: char | string

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1, . . . , NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: dependsOn(TaskB,TaskA,WhenStatus=["Pass", "Fail"])

IterationArtifactMatching – Setting that controls which dependent task iterations run

true or 1 (default) | false or 0

Setting that controls which dependent task iterations run, specified as a numeric or logical 1 (true) or 0 (false):

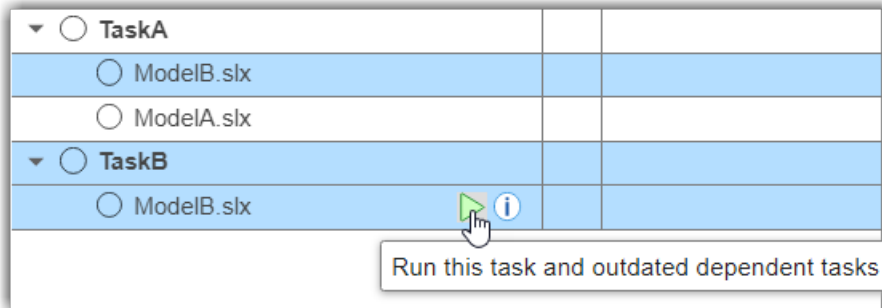
- true – When the build system runs the dependencies of a task, the build system runs only the task iterations that the tasks have in common.
- false – When the build system runs the dependencies of a task, the build system runs all task iterations. This behavior is useful when you have a task that creates new project artifacts and a task that runs on each artifact in the project. The second task depends on all project artifacts generated by the first task.

For example, suppose you have two tasks: TaskA and TaskB:

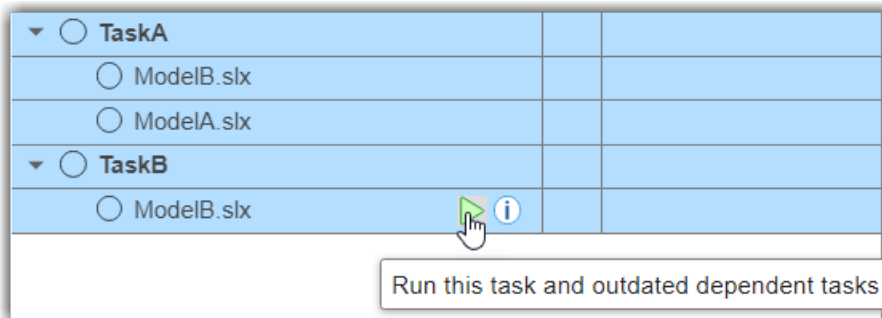
- TaskA runs on ModelA and ModelB.
- TaskB runs only on ModelB and depends on TaskA.

If you run TaskB and:

- IterationArtifactMatching is true, TaskA runs only on ModelB.



- IterationArtifactMatching is false, TaskA runs on both ModelA and ModelB.



Example: `dependsOn(TaskB,TaskA,IterationArtifactMatching=false)`

Data Types: `logical`

WhenStatus — Setting that controls when dependencies run

`"Pass"` (default) | `["Pass", "Fail"]` | `["Pass", "Fail", "Error"]`

Setting that controls when dependencies run, specified as either:

- `"Pass"` — Only run the task if the dependencies pass. For example, if TaskB depends on TaskA, TaskA needs to pass before TaskB runs. If TaskA fails or errors, TaskB does not run.
- `["Pass", "Fail"]` — Only run the task if the dependencies either pass or fail. For example, if TaskB depends on TaskA, TaskA needs to either pass or fail before TaskB runs. If TaskA errors, TaskB does not run.
- `["Pass", "Fail", "Error"]` — The task runs, even if the dependencies fail or error. For example, if TaskB depends on TaskA, TaskA can pass, fail, or error and TaskB still runs.

Example: `dependsOn(TaskB,TaskA,WhenStatus=["Pass", "Fail"])`

Data Types: `string`

Tips

In your process model, you can specify the relationship between tasks as either a `dependsOn` or `runsAfter` relationship. If you want to specify a preferred task execution order for tasks that do not depend on each other, you can specify your preferred task execution order by using the `runsAfter` method instead. For more information, see “Define Task Relationships”.

See Also

`padv.Task` | `padv.TaskResult` | `runsAfter`

Topics

“Add Tasks to Process”

“Define Task Relationships”

“Overview of Process Model”

run

Namespace: `padv`

Run task

Syntax

```
taskResult = run(taskObj)
taskResult = run(taskObj,inputArtifacts)
```

Description

`taskResult = run(taskObj)` runs the task represented by `taskObj` and returns the result from the task.

This function requires CI/CD Automation for Simulink Check.

How a task runs depends on how the you define the task. You can define tasks using a function or a class:

- Function-based tasks — Runs the function specified by the `Action` property of the task.
- Class-based task — Runs the `run` function implemented inside the class definition.

By default, when you create a `padv.Task` object, the task is a function-based task, even if you do not specify an `Action` property for the task.

`taskResult = run(taskObj,inputArtifacts)` uses the artifacts specified by `inputArtifacts` as inputs to the task. The `InputQueries` property of the task specifies the query that provides the `inputArtifacts` for the task.

Input Arguments

taskObj — Task object that represents task

`padv.Task` object

Task object that represents a task, specified as a `padv.Task` object.

Example: `myTaskObj = padv.Task("myTask");`

inputArtifacts — Artifacts that are inputs to task

cell array of `padv.Artifact` objects

Artifacts that are inputs to the task, specified as a cell array of `padv.Artifact` objects.

If you specified the `InputQueries` property for a task, the `InputQueries` automatically passes a cell array of `padv.Artifact` objects to `inputArtifacts` when you run the task.

Output Arguments

taskResult — Result from task

TaskResult object

Result from the task, returned as a TaskResult object.

See Also

`padv.Task` | `padv.TaskResult`

Topics

“Add Tasks to Process”

“Reconfigure Task Behavior”

“Define Task Relationships”

“Create Custom Tasks”

runsAfter

Namespace: padv

Specify preferred execution order for tasks

Syntax

```
runsAfter(taskObj, predecessors)  
runsAfter( ____, Name=Value)
```

Description

`runsAfter(taskObj, predecessors)` specifies a preferred execution order for tasks. If possible, the build system runs the predecessor tasks, specified by `predecessors`, before the task represented by `taskObj`.

This function requires CI/CD Automation for Simulink Check.

`runsAfter(____, Name=Value)` specifies how the build system handles the preferred execution order using one or more `Name=Value` arguments.

Note You can specify the relationship between two tasks as *either* a `dependsOn` relationship or a `runsAfter` relationship, but not both.

If you define multiple relationships between the same tasks, the build system only uses the most recent relationship and ignores previous relationships. For example:

```
runsAfter(taskA, taskB)  
runsAfter(taskB, taskA) % build system only uses this relationship
```

Examples

Specify Preferred Execution Order for Two Tasks

Use the `runsAfter` function to specify that one task should run after another task.

Open the Process Advisor example project.

```
processAdvisorExampleStart
```

Open the `processmodel.m` file. The file is at the root of the project.

Replace the contents of the `processmodel.m` file with the following code:

```
function processmodel(pm)  
    arguments  
    pm padv.ProcessModel  
end
```

```

FirstTask = padv.Task("FirstTask");
SecondTask = padv.Task("SecondTask");

runsAfter(SecondTask,FirstTask);

addTask(pm,FirstTask);
addTask(pm,SecondTask);

```

end

This code uses `padv.Task` to create two task objects: `FirstTask` and `SecondTask`.

The object function `runsAfter` specifies that `SecondTask` should run after `FirstTask`.

The function `addTask` adds the task objects to the `padv.ProcessModel` object.

Open the Process Advisor app. In the MATLAB Command Window, enter:

```
processAdvisorWindow
```

In the toolstrip, click the **Run All** button. You can see that **SecondTask** runs after **FirstTask**.

Input Arguments

taskObj — Task object that represents task

`padv.Task` object

Task object that represents a task, specified as a `padv.Task` object.

Example: `myTaskObj = padv.Task("myTask");`

predecessors — Tasks that should run before `taskObj` runs

string | character vector | `padv.Task` object

Tasks that should run before `taskObj` runs, specified as either:

- The name of a task, specified as a string or character vector.
- A `padv.Task` object.

Example: `runsAfter(SecondTask,"FirstTask")`

Example: `runsAfter(SecondTask,FirstTask)`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `runsAfter(SecondTask,FirstTask,StrictOrdering=true)`

IterationArtifactMatching — Setting that controls which predecessor task iterations run

true or 1 (default) | false or 0

Setting that controls which predecessor task iterations run, specified as a numeric or logical 1 (true) or 0 (false):

- `true` — When the build system runs the predecessors of a task, the build system runs only the task iterations that the tasks have in common.
- `false` — When the build system runs the predecessor of a task, the build system runs all task iterations. This behavior is useful when you have a task that creates new project artifacts and a task that runs on each artifact in the project. The second task should run after all project artifacts are generated by the first task.

For example, suppose you have two tasks: `FirstTask` and `SecondTask`:

- `FirstTask` runs on `ModelA` and `ModelB`.
- `SecondTask` runs only on `ModelB` and should run after on `FirstTask`.

If you run `SecondTask` and:

- `IterationArtifactMatching` is `true`, `FirstTask` runs only on `ModelB`.
- `IterationArtifactMatching` is `false`, `FirstTask` runs on both `ModelA` and `ModelB`.

Example: `runsAfter(SecondTask,FirstTask,IterationArtifactMatching=false)`

Data Types: `logical`

StrictOrdering — Setting that controls whether build system ignores circular relationships between tasks

`false` or `0` (default) | `true` or `1`

Setting that controls whether the build system ignores circular relationships between tasks, specified as a numeric or logical `0` (`false`) or `1` (`true`). By default, if you specify a circular relationship between tasks, the build system ignores the relationship. For example, if you specify both `runsAfter(SecondTask,FirstTask)` and `runsAfter(FirstTask,SecondTask)`, the build system ignores the `runsAfter` relationship.

If you specify `StrictOrdering` as `true`, the build system generates an error when you try to run tasks that have a circular relationship.

Example: `runsAfter(SecondTask,FirstTask,StrictOrdering=true)`

Data Types: `string`

Tips

In your process model, you can specify the relationship between tasks as either a `dependsOn` or `runsAfter` relationship. If your task depends on inputs or results from another task to run successfully, specify the relationship by using the `dependsOn` method instead. For more information, see “Define Task Relationships”.

See Also

`padv.Task` | `padv.TaskResult` | `dependsOn`

Topics

“Add Tasks to Process”

“Define Task Relationships”

“Overview of Process Model”

padv.TaskResult

Create and access results from task

Description

This object requires CI/CD Automation for Simulink Check. A `padv.TaskResult` object represents the results from a task. The `run` function of a `padv.Task` creates a `padv.TaskResult` object that you can use to access the results from the task. When you create a custom task, you can specify the results from your custom task. You can also use the function `getProcessTaskResults` to view a list of the last task results for a project. The Process Advisor app uses task results to determine the task statuses, output artifacts, and detailed task results that appear in the **Tasks**, **I/O**, and **Details** columns of the app.

Creation

Syntax

```
resultObj = padv.TaskResult()
```

Description

`resultObj = padv.TaskResult()` creates a result object `resultObj` that represents the results from a task.

Properties

Status — Task result status

Pass (default) | Fail | Error

Task result status, returned as:

- **Pass** — A passing task status. The task completed successfully without failures or errors.
- **Fail** — A failing task status. The task completed, but the results were not successful.
- **Error** — An error task status. The task generated an error and did not complete.

The **Status** property determines the task status shown in the **Tasks** column in the Process Advisor app.

For custom tasks, you can specify the task result status as either:

- `padv.TaskStatus.Pass` — Sets the **Status** property to **Pass**.
- `padv.TaskStatus.Fail` — Sets the **Status** property to **Fail**.
- `padv.TaskStatus.Error` — Sets the **Status** property to **Error**.

For example, `taskResult.Status = padv.TaskStatus.Fail` sets the **Status** property of the task result object to **Fail** to represent a failing task status.

Example: `Fail`

OutputArtifacts — Artifacts created by task

`padv.Artifact` object | array of `padv.Artifact` objects

Artifacts created by the task, returned as a `padv.Artifact` object or array of `padv.Artifact` objects.

The `OutputArtifacts` property determines the output artifacts shown in the **I/O** column in the Process Advisor app.

The build system only manages output artifacts specified by the task result. For custom tasks, use the `OutputPaths` argument to define the output artifacts for the task result.

Details — Temporary storage for task-specific data

`struct`

Temporary storage for task-specific data, returned as a `struct`. The build system uses `Details` to store task-specific data that other build steps can use.

Note that `Details` are temporary. The build system does not save `Details` with the task results after the build finishes.

Data Types: `struct`

ResultValues — Number of passing, warning, and failing conditions

`struct` with `Pass: 0, Warn: 0, Fail: 0` (default) | `struct` with fields `Pass, Warn, Fail`

Number of passing, warning, and failing conditions, returned as a `struct` with fields:

- `Pass` — Number of passing conditions returned by task
- `Warn` — Number of warning conditions returned by task
- `Fail` — Number of failing conditions returned by task

The `ResultValues` property determines the detailed results shown in the **Details** column in the Process Advisor app.

For example, the task `padv.builtin.task.RunModelStandards` runs several Model Advisor checks and returns the number of passing, warning, and failing checks. If you run the task and one check passes, two checks generate a warning, and three checks fail, `ResultValue` returns:

```
ans =
```

```
  struct with fields:
```

```
    Pass: 1  
    Warn: 2  
    Fail: 3
```

Data Types: `struct`

OutputPaths — Define OutputArtifacts for task result

`string`

This property is write-only.

`OutputArtifacts` for task result, specified as a string or string array.

The build system adds each path specified by `OutputArtifacts` to the `OutputArtifacts` argument as a `padv.Artifact` object with type `padv_output_file`.

Example: `taskResultObj.OutputPaths = string(fullfile(pwd,filename))`

Example: `taskResultObj.OutputPaths = [string(fullfile(pwd,filename1)), string(fullfile(pwd,filename2))]`

Data Types: `char` | `string`

Object Functions

- `applyStatus`

Examples


Create Task Result for Custom Task

Create a `padv.TaskResult` object for a custom task that has a failing task status, outputs a single `.json` file, and 1 passing condition, 2 warning conditions, and 3 failing conditions.

Open the Process Advisor example project.

```
processAdvisorExampleStart
```

The model `AHRS_Voter` opens with the Process Advisor pane to the left of the Simulink canvas.

In the Process Advisor pane, click the **Edit process model**  button to open the `processmodel.m` file for the project.

Replace the contents of the `processmodel.m` file with this example process model code:

```
function processmodel(pm)
    % Defines the project's processmodel

    arguments
        pm padv.ProcessModel
    end

    addTask(pm,"ExampleTask",Action=@ExampleAction);

end

function taskResult = ExampleAction(~)

    % Create a task result object that stores the results
    taskResult = padv.TaskResult();

    % Specify the task status shown in the Tasks column
    taskResult.Status = padv.TaskStatus.Fail;

    % Specify the output files shown in the I/O column
    cp = currentProject;
```

```

rf = cp.RootFolder;
outputFile = fullfile(rf,"tools","sampleChecks.json");
taskResult.OutputPaths = string(outputFile);

% Specify the values shown in the Details column
taskResult.ResultValues.Pass = 1;
taskResult.ResultValues.Warn = 2;
taskResult.ResultValues.Fail = 3;

```

end

Save the `processmodel.m` file.

Go back to the Process Advisor app and click **Refresh Tasks** to update the list of tasks shown in the app.

In the top-left corner of the Process Advisor pane, switch the filter from **Model** to **Project**.

In the top-right corner of the Process Advisor pane, click **Run All**.

- The **Tasks** column shows a failing task status to the left of **ExampleTask**. This code from the example process model specifies the task status shown in the **Tasks** column:

```
taskResult.Status = padv.TaskStatus.Fail;
```

- The **I/O** column shows an output artifact associated with the task. This code from the example process model specifies the output artifact shown in the **I/O** column:

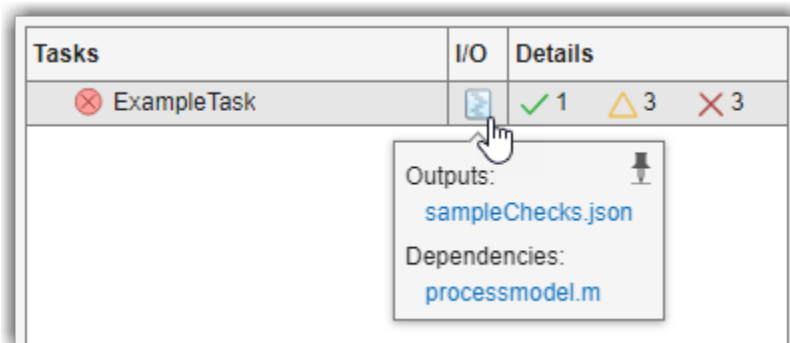
```
taskResult.OutputPaths = string(fullfile(pwd,outputFile));
```

- The **Details** column shows 1 passing condition, 2 warning conditions, and 3 failing conditions. This code from the example process model specifies the detailed task results shown in the **Details** column:

```

taskResult.ResultValues.Pass = 1;
taskResult.ResultValues.Warn = 2;
taskResult.ResultValues.Fail = 3;

```



See Also

`applyStatus` | `padv.Task`

Topics

“Add Tasks to Process”

“Create Custom Tasks”

applyStatus

Namespace: padv

Apply new task status if priority is higher

Syntax

```
applyStatus(resultObj, taskStatus)
```

Description

`applyStatus(resultObj, taskStatus)` applies a new task status `taskStatus` to the task result object `resultObj` if the priority level of `taskStatus` is higher than the current `Status` property of the task result object.

This function requires CI/CD Automation for Simulink Check.

The priority levels from lowest to highest are:

- `padv.TaskStatus.Pass`
- `padv.TaskStatus.Fail`
- `padv.TaskStatus.Error`

Note The function `applyStatus` can only change the `Status` to a higher priority status. For example, if you apply a failing status and then apply a passing status, the status remains a failing status because the priority of `padv.TaskStatus.Fail` is higher than the priority of `padv.TaskStatus.Pass`.

```
taskResult = padv.TaskResult(); % By default, Status is Pass.
applyStatus(taskResult, padv.TaskStatus.Fail); % Status changes to Fail.
applyStatus(taskResult, padv.TaskStatus.Pass); % Status remains Fail.
taskResult
```

```
taskResult =
```

```
    TaskResult with properties:
```

```
        Status: Fail
    OutputArtifacts: [0x0 padv.Artifact]
         Details: [1x1 struct]
    ResultValues: [1x1 struct]
```

To set the `Status` property of a task result object to a specific value, manually set the property to either `padv.TaskStatus.Pass`, `padv.TaskStatus.Fail`, or `padv.TaskStatus.Error`. For example, to set the `Status` of a task result object `taskResult` to `Pass`, use `taskResult.Status = padv.TaskStatus.Pass`.

Examples

Apply Status to Task Result

Use `applyStatus` to update the `Status` property of a task result object. If the status is a higher priority status, `applyStatus` updates the `Status` property of the task result object.

Create a task result object. By default, the `Status` property of the task result object is specified as `Pass`.

```
taskResult = padv.TaskResult();
```

Suppose the task needs to generate an error. Use `applyStatus` to apply an error task status, specified by `padv.TaskStatus.Error`.

```
applyStatus(taskResult, padv.TaskStatus.Error);
```

`padv.TaskStatus.Error` has a higher priority than a passing task status, so `applyStatus` updates the `Status` property of the task result object.

Apply a passing task status to the task result object. A passing task status is specified by `padv.TaskStatus.Pass`.

```
applyStatus(taskResult, padv.TaskStatus.Pass);
```

`padv.TaskStatus.Pass` does not have a higher priority than an error task status, so `applyStatus` does not change the `Status` of the task result object.

Inspect the properties of the task result object.

```
taskResult
```

Suppose you want to reset the status of the task result object to a passing task status. Manually specify the `Status` property as `padv.TaskStatus.Pass`.

```
taskResult.Status = padv.TaskStatus.Pass
```

```
taskResult =
```

```
  TaskResult with properties:
```

```
    Status: Pass
  OutputArtifacts: [0x0 padv.Artifact]
    Details: [1x1 struct]
  ResultValues: [1x1 struct]
```

The task result object now has a passing task status.

Input Arguments

resultObj — Task result object

`padv.TaskResult` object

Task result object, specified as a `padv.TaskResult` object.

taskStatus — Task status

`padv.TaskStatus.Pass` | `padv.TaskStatus.Fail` | `padv.TaskStatus.Error`

Task status, specified as `padv.TaskStatus.Pass`, `padv.TaskStatus.Fail`, or `padv.TaskStatus.Error`.

Example: `padv.TaskStatus.Fail`

See Also

`padv.Task` | `padv.TaskResult`

Topics

“Add Tasks to Process”

“Create Custom Tasks”

Build System

runprocess

Generate and run pipeline of tasks by using build system

Syntax

```
[buildResult,exitCode] = runprocess()  
[buildResult,exitCode] = runprocess(Name=Value)
```

Description

[buildResult,exitCode] = runprocess() generate a model-based design (MBD) pipeline and run the pipeline using the build system. The process model (processmodel.p or processmodel.m) defines the tasks for the pipeline.

This function requires CI/CD Automation for Simulink Check.

[buildResult,exitCode] = runprocess(Name=Value) specifies how the MBD pipeline runs using one or more Name=Value arguments.

Examples

Run MBD Pipeline

Open a project and use runprocess to generate and run the MBD pipeline using the build system.

Open the Process Advisor example project, which contains an example process model. The process model defines the tasks for the pipeline.

```
processAdvisorExampleStart
```

Generate and run the MBD pipeline and store the results in the variable results.

```
results = runprocess()
```

Run Specific Tasks

Open a project and use runprocess. To only run a specific set of tasks, provide the task names to the Tasks argument.

Open the Process Advisor example project, which contains an example process model. The process model defines the tasks for the pipeline.

```
processAdvisorExampleStart
```

Run only the tasks **Generate Simulink Web View** (padv.builtin.task.GenerateSimulinkWebView) and **Check Modeling Standards** (padv.builtin.task.RunModelStandards) by specifying the Tasks argument.


```
% run the Generate Simulink Web View task
% and the Check Modeling Standards tasks
runprocess(...
Tasks = ["padv.builtin.task.GenerateSimulinkWebView",...
"padv.builtin.task.RunModelStandards"])
```

Run Tasks Associated with Specific Artifact

Open a project and use `runprocess`. To only run the tasks associated with a specific artifact, provide a full path, relative path, or a `padv.Artifact` object to the `FilterArtifact` argument.

Open the Process Advisor example project, which contains an example process model. The process model defines the tasks for the pipeline.

```
processAdvisorExampleStart
```

Run tasks for the `AHRS_Voter` model by specifying the relative path to the model.

```
% run only the AHRS_Voter tasks
runprocess(...
FilterArtifact = fullfile(...
"02_Models", "AHRS_Voter", "specification", "AHRS_Voter.slx"))
```

Run Specific Task Iteration, Clean Task Results, and Delete Task Outputs

Open a project and run one specific task iteration in the pipeline.

Open the Process Advisor example project, which contains an example process model.

```
processAdvisorExampleStart
```

Get a list of the task iterations in the MBD pipeline.

```
tasks = generateProcessTasks;
```

Force `runprocess` to run one of the task iterations by specifying `Force` as `true` and `Tasks` as one of the tasks in `tasks`.

```
runprocess(Force=true, Tasks=tasks(1))
```

When `Force` is `true`, `runprocess` runs the pipeline, even if the pipeline already had results that were marked as up to date.

Clean task results and delete task outputs.

```
runprocess(Clean=true, DeleteOutputs=true)
```

When you clean task results and delete task outputs, it is as if the tasks were not run.

Run Specific Process

Inside your process model, you can define multiple processes for the different build and verification workflows, environments, and other situations that your team needs a defined process for. By default,

the `runprocess` function runs the default process. To run a different process, you can specify the `Process` argument.

Open the Process Advisor example project.

```
processAdvisorExampleStart
```

The model `AHRS_Voter` opens with the Process Advisor pane to the left of the Simulink canvas.

In the Process Advisor pane, click the **Edit process model**  button to open the `processmodel.m` file for the project.

Replace the contents of the `processmodel.m` file with this code:

```
function processmodel(pm)
    arguments
        pm padv.ProcessModel
    end

    % Add Processes
    processA = pm.addProcess("A");
    processB = pm.addProcess("B");

    % Add Tasks
    processA.addTask(padv.builtin.task.RunModelStandards);
    processB.addTask(padv.builtin.task.CollectMetrics);

end
```

This process model code defines two processes that each contain a task. In Process Advisor, you can refresh the tasks and see the processes in the toolstrip.

Run the process named "B" by using the `Process` argument.

```
runprocess(Process = "B")
```

For more information on processes, see "Manage Multiple Build and Verification Workflows Using Processes".

Dry Run Process with License Checkouts

Before you try to run your process in CI, you can dry run your process to validate your task inputs, generate representative task outputs, and make sure that you have the required licenses available on your CI agent.

Open the Process Advisor example project, which contains an example process model.

```
processAdvisorExampleStart
```

Perform a dry run of the process and automatically check out the licenses associated with the tasks by using the `DryRun` and `DryRunLicenseCheckout` arguments.

```
runprocess(DryRun = true, DryRunLicenseCheckout = true)
```

For more information, see “Dry Run Tasks to Test Process Model”.

Input Arguments

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, . . . , NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `[buildResult,exitCode] = runprocess(Force=true)`

Tasks — Names of tasks that you want to run

character vector | cell array of character vectors | string | string array

Names of tasks that you want to run, specified as a character vector, cell array of character vectors, string, or string array. The task name is defined by the `Name` property of the task.

Alternatively, you can specify the task iteration IDs for individual task iterations that you want to run. See `generateProcessTasks` and `createProcessTaskID`.

Note You can only run tasks that are defined in the process model. For more information, see “Overview of Process Model” and “Modify Default Process Model to Fit Your Process”.

Example: `"padv.builtin.task.GenerateSimulinkWebView"`

Example: `["padv.builtin.task.GenerateSimulinkWebView", ...
"padv.builtin.task.RunModelStandards"]`

Data Types: `char` | `string`

Process — Name of process that you want to run

`padv.ProcessModel.DefaultProcessId` (default) | character vector | string

Name of process that you want to run, specified by a character vector or string.

For more information about processes, see “Manage Multiple Build and Verification Workflows Using Processes”.

Example: `"CIPipeline"`

Data Types: `char` | `string`

Subprocesses — Names of subprocesses that you want to run

character vector | cell array of character vectors | string | string array

Names of subprocesses that you want to run, specified as a character vector, cell array of character vectors, string, or string array. The subprocess name is defined by the `Name` property of the subprocess.

Example: `"SubprocessA"`

Example: `["SubprocessA",SubprocessB"]`

Data Types: `char` | `string`

FilterArtifact — Artifacts that you want to run tasks for

string.empty (default) | string | padv.Artifact object | array of padv.Artifact objects

Artifact or artifacts that you want to run tasks for, specified as either the full path to an artifact, relative path to an artifact, a padv.Artifact object that represents an artifact, or an array of padv.Artifact objects.

Example: fullfile("C:\", "User", "projectA", "myModel.slx")

Example: fullfile("02_Models", "AHRV_Voter", "specification", "AHRV_Voter.slx")

Example:

```
padv.Artifact("sl_model_file", fullfile("02_Models", "AHRV_Voter", "specification", "AHRV_Voter.slx"))
```

Data Types: string

Force — Skip or run up-to-date task iterations

false or 0 (default) | true or 1

Skip or run up-to-date tasks, specified as a numeric or logical 0 (false) or 1 (true). By default, runprocess does not run task iterations that have up to date results.

Example: true

Data Types: logical

Isolation — Include task dependencies

false or 0 (default) | true or 1

Include task dependencies, specified as a numeric or logical 0 (false) or 1 (true).

By default, runprocess includes task dependencies when running a task. Specify Isolation as true if you want to run a task in isolation, without running task dependencies.

Note that you define task dependencies in the process model by using the function dependsOn.

Example: true

Data Types: logical

Clean — Clear task results and delete outputs

false or 0 (default) | true or 1

Clear task results and delete task outputs, specified as a numeric or logical 0 (false) or 1 (true).

If you specify Clean as true:

- The runprocess function ignores other name-value arguments, cleans the task results, and deletes task outputs.
- The OutputDirectory of the task might still contain files. The runprocess function only deletes the task outputs, specified by the OutputPaths property of the padv.TaskResult object for the task.
- You cannot specify MarkStale as true. The arguments are mutually exclusive.

Example: true

Data Types: logical

DeleteOutputs — Delete task outputs`false` or `0` (default) | `true` or `1`

Delete task outputs, specified as a numeric or logical `0` (false) or `1` (true).

Note To delete task outputs with `DeleteOutputs`, you must specify `Clean` as `true`.

Example: `true`

Data Types: `logical`

MarkStale — Mark task as outdated`false` or `0` (default) | `true` or `1`

Mark task as outdated, specified as a numeric or logical `0` (false) or `1` (true). When you mark a task as stale, the results appear outdated in the Process Advisor app.

Note If you specify `MarkStale` as `true`, then you cannot specify `Clean` as `true`. The arguments are mutually exclusive.

Example: `true`

Data Types: `logical`

DryRun — Validate inputs and generate representative outputs without running task`false` or `0` (default) | `true` or `1`

Validate task inputs and generate representative task outputs without actually running the tasks, specified as a numeric or logical `0` (false) or `1` (true). You can use a dry run to help make sure your tasks are set up as expected in the process model.

To have the dry run check out the licenses associated with the task, specify the `runprocess` argument `DryRunLicenseCheckout` as `true`.

Alternatively, you can perform dry runs from the Process Advisor app. For more information, see “Dry Run Tasks to Test Process Model”.

Example: `true`

Data Types: `logical`

DryRunLicenseCheckout — Dry runs check out product licenses associated with tasks in process`false` or `0` (default) | `true` or `1`

When the `runprocess` argument `DryRun` is `true`, dry runs check out the product licenses associated with the tasks in process, returned as a numeric or logical `1` (true) or `0` (false).

For more information on dry runs, see “Dry Run Tasks to Test Process Model”.

Example: `true`

Data Types: `logical`

ExitInBatchMode — Exit MATLAB when running in batch mode`true or 1 (default) | false or 0`

Exit MATLAB when running in batch mode, specified as a numeric or logical 1 (`true`) or 0 (`false`). By default, if you are running MATLAB in batch mode and `runprocess` finishes running, `runprocess` exits MATLAB.

The process exit codes are:

- 0 if the Status of `buildResult` is PASS
- 1 if the Status of `buildResult` is ERROR
- 2 if the Status of `buildResult` is FAIL

Example: `false`

Data Types: `logical`

GenerateReport — Automatically generate report at end of runprocess`false or 0 (default) | true or 1`

Automatically generate report after `runprocess` runs tasks, specified as a numeric or logical 1 (`true`) or 0 (`false`).

Example: `runprocess(GenerateReport = true)`

Data Types: `logical`

ReportFormat — File format for generated report`"pdf" (default) | "html" | "html-file" | "docx"`

File format for the generated report, specified as one of these values:

- `"pdf"` — PDF file
- `"html"` — HTML report, packaged as a zipped file that contains the HTML file, images, style sheet, and JavaScript® files of the report
- `"html-file"` — HTML report
- `"docx"` — Microsoft® Word document

Note that for the `runprocess` function to generate a report, you must also specify the argument `GenerateReport` as `true`.

Example: `runprocess(GenerateReport = true,ReportFormat = "html-file")`

ReportPath — Name and path of generated report`"ProcessAdvisorReport" (default) | string array`

Name and path of generated report, specified as a string array.

Note that for the `runprocess` function to generate a report, you must also specify the argument `GenerateReport` as `true`.

Example: `runprocess(GenerateReport = true,ReportPath = fullfile(pwd,"folderName","reportName"))`

Data Types: `string`

RerunFailedTasks — Rerun failed task iterations`false or 0 (default) | true or 1`

Rerun failed task iterations, specified as a numeric or logical `0` (`false`) or `1` (`true`).

Example: `true`

Data Types: `logical`

RerunErroredTasks — Rerun errored task iterations`false or 0 (default) | true or 1`

Rerun errored task iterations, specified as a numeric or logical `0` (`false`) or `1` (`true`).

Example: `true`

Data Types: `logical`

RefreshProcessModel — Automatically refresh before running tasks`true or 1 (default) | false or 0`

Automatically refresh before running tasks, specified as a numeric or logical `1` (`true`) or `0` (`false`). By default, `runprocess` refreshes before running tasks so that the run uses the current state of the process model and project. If you specify `RefreshProcessModel` as `false`, `runprocess` does not refresh before running, but the run might not include the latest changes to tasks in the process model or artifacts in the project.

Example: `false`

Data Types: `logical`

ReanalyzeProjectAnalysisIssues — Automatically reanalyze project analysis issues that have severity level of error`true or 1 (default) | false or 0`

Automatically reanalyze project analysis issues that have a severity level of error, specified as a numeric or logical `1` (`true`) or `0` (`false`).

To prevent the build system from reanalyzing project analysis issues that have a severity level of error, you can specify `ReanalyzeProjectAnalysisIssues` as `false`. This might reduce the execution time for `runprocess`, but the build system might not generate the expected task iterations or detect outdated results.

Fix the issues listed in the **Project Analysis Issues** pane of the Process Advisor app to make sure the build system can fully analyze the project, generate the expected task iterations, and detect outdated results.

Example: `false`

Data Types: `logical`

GenerateJUnitForProcess — Generate JUnit-style XML report for process`false or 0 (default) | true or 1`

Generate JUnit-style XML report for each task in process, specified as a numeric or logical `0` (`false`) or `1` (`true`).

Example: `true`

Data Types: `logical`

EnableTaskLogging — Control command-line outputs from tasks

`true` or `1` (default) | `false` or `0`

Control command-line outputs from tasks, specified as:

- A numeric or logical `0` (`false`) — Task logging is disabled.
- A numeric or logical `1` (`true`) — Task logging is enabled.

If the project setting `SuppressOutputWhenInteractive` is `true` and MATLAB is not running in batch mode, task logging is automatically disabled.

When task logging is disabled, tasks does not output information in the MATLAB Command Window.

Example: `false`

Data Types: `logical`

SuppressOutputWhenInteractive — Suppress command-line output from Process Advisor

`logical.empty` (default) | `1` or `true` | `0` or `false`

Suppress command-line output from Process Advisor during interactive MATLAB sessions, specified as either:

- An empty logical array (`logical.empty`) — No impact. `runprocess` follows the Process Advisor setting **Suppress outputs to command window**.
- A numeric or logical `1` (`true`) — Override the Process Advisor setting **Suppress outputs to command window** and suppress output to the MATLAB Command Window.
- A numeric or logical `0` (`false`) — Override the Process Advisor setting **Suppress outputs to command window** and show build logs and task execution messages in the MATLAB Command Window.

Note that this argument has no impact when you run MATLAB in batch mode, which is typically the case for CI systems.

Example: `true`

Data Types: `logical`

Output Arguments

buildResult — Results of run

`padv.BuildResult`

Results of run, returned as a `padv.BuildResult` object.

The `padv.BuildResult` object includes:

- The start time and end time of the run
- The status of the run (`Pass`, `Error`, `Fail`)
- Lists of the tasks that the passed, generated errors, were skipped, or failed during the run
- Input arguments to the run

exitCode — Exit code from run

0 | 1 | 2

Exit code from run, returned as a `double` representing the process error code.

- 0 if the Status of the `buildResult` is Pass
- 1 if the Status of the `buildResult` is Error
- 2 if the Status of the `buildResult` is Fail

Alternative Functionality**App**

You can also use the Process Advisor app to run each task or individual task iterations in the process. To open the Process Advisor app for a project, in the MATLAB Command Window, enter:

```
processAdvisorWindow
```

See Also

`padv.BuildResult` | Process Advisor | `generateReport`

Topics

“Automate and Run Tasks with Process Advisor”

“Modify Default Process Model to Fit Your Process”

“Overview of Process Model”

“Manage Multiple Build and Verification Workflows Using Processes”

“Approaches to Running Processes in CI”

createProcessTaskID

Generate ID for specific task iteration defined by process model

Syntax

```
ID = createProcessTaskID(task,artifact)
```

Description

`ID = createProcessTaskID(task,artifact)` generates the identifier, `ID`, for an individual task iteration defined by the process model. A *task iteration* is the pairing of a task, `task`, to a specific project artifact, `artifact`.

This function requires CI/CD Automation for Simulink Check.

Examples

Run One Task on One Artifact

Suppose you have a process model with several tasks, but right now you only want to run the task `padv.builtin.task.RunModelStandards` on the model `AHRS_Voter.slx`. Use the function `createProcessTaskID` to generate the ID for a specific task iteration, then use the function `runprocess` to run only that specific task iteration.

Open the **Process Advisor** example project, which contains an example process model.

```
processAdvisorExampleStart
```

Specify a task that exists in the process model. For this example, specify the built-in task for running Model Advisor checks, `padv.builtin.task.RunModelStandards`.

```
task = padv.builtin.task.RunModelStandards;
```

Use `padv.Artifact` to specify the project artifact that you want the task to run on. For this example, the artifact type is `sl_model_file` because the artifact is a Simulink model and the address is the path to `AHRS_Voter.slx`, relative to the project root.

```
artifactType = "sl_model_file";  
address = fullfile("02_Models","AHRS_Voter","specification","AHRS_Voter.slx");  
artifact = padv.Artifact(artifactType,address);
```

Use the task instance and artifact to generate the ID for the specific task iteration.

```
runModelStandards_for_AHRS_Voter = createProcessTaskID(task,artifact)
```

```
runModelStandards_for_AHRS_Voter =
```

```
"padv.builtin.task.RunModelStandards|sl_model_file|02_Models/AHRS_Voter/specification/AHRS_Voter"
```

Use the function `runprocess` to run the task iteration.

```
runprocess(Tasks = runModelStandards_for_AHRS_Voter)
```

When you specify the Tasks value as the ID for a single task iteration, the function `runprocess` runs only the specified task iteration. For this example, `runprocess` runs only the task iteration associated with the task `padv.builtin.task.RunModelStandards` and the artifact `AHRS_Voter.slx`.

Note Alternatively, instead of creating and then running the task iterations, you can directly specify the Task and FilterArtifact arguments of the `runprocess` function to run the task on a specific artifact:

```
runprocess(...
Tasks = "padv.builtin.task.RunModelStandards",...
FilterArtifact = fullfile("02_Models","AHRS_Voter","specification","AHRS_Voter.slx"))
```

But note that you can only run the tasks if the tasks are defined in the process model and the artifacts exist in the project.

Input Arguments

task — Task name or subclass of `padv.Task`

string | character vector | `padv.Task` object

Either:

- Name of task, specified as a string or character vector. The name of a task is stored in the `Name` property of the task. For example, `"name_of_my_custom_task"`.
- Subclass of `padv.Task`, specified as a `padv.Task` object. Built-in tasks are subclasses of `padv.Task`. For example, you can specify the `padv.Task` object `padv.builtin.task.RunModelStandards` for the task argument.

Example: `"name_of_my_custom_task"`

Example: `"padv.builtin.task.RunModelStandards"`

Example: `padv.builtin.task.RunModelStandards`

Data Types: `char` | `string`

artifact — File in project

`padv.Artifact` object

File in project, specified as a `padv.Artifact` object.

Example: `padv.Artifact("project","ProcessAdvisorExample.prj")`

Example: `padv.Artifact("sl_model_file", "02_Models/AHRS_Voter/specification/AHRS_Voter.slx")`

Output Arguments

ID — Identifier for task iteration defined by process model

string

Identifier for task iteration defined by the process model, returned as a string.

IDs take the form: "*taskNameOrObject|fileType|relativePath*", where *relativePath* is the path relative to the project root.

Example IDs:

- "myCustomProjectTask|project|ProcessAdvisorExample.prj"
- "padv.builtin.task.RunModelStandards|sl_model_file|02_Models/AHRS_Voter/specification/AHRS_Voter.slx"
- "padv.builtin.task.RunTestsPerTestCase|sl_test_case|ced877ff-cfb8-4fa8-9bbf-aaa29b1d926b"

Alternative Functionality

App

You can also use the Process Advisor app to run individual task iterations in the process. To open the Process Advisor app for a project, in the MATLAB Command Window, enter:

```
processAdvisorWindow
```

See Also

`generateProcessTasks` | `getProcessTaskResults` | `runprocess`

Topics

"Automate and Run Tasks with Process Advisor"

"Programmatically Run Tasks"

generateProcessTasks

Get list of IDs for task iterations in MBD pipeline

Syntax

```
IDs = generateProcessTasks()  
IDs = generateProcessTasks(Name=Value)
```

Description

`IDs = generateProcessTasks()` returns identifiers, `IDs`, for each of the task iterations in the model-based design (MBD) pipeline.

By default, `generateProcessTasks` returns an ID for each combination of tasks and associated project artifacts in the MBD pipeline.

This function requires CI/CD Automation for Simulink Check.

`IDs = generateProcessTasks(Name=Value)` filters the list of IDs using one or more `Name=Value` arguments.

Examples

List IDs for Each Task Iteration in MBD Pipeline

Suppose you have a process model that adds several tasks to the process. Use the function `generateProcessTasks` to list the IDs for each task iteration in the MBD pipeline.

Open the Process Advisor example project, which contains an example process model.

```
processAdvisorExampleStart
```

List the IDs for each task iteration in the MBD pipeline.

```
IDs = generateProcessTasks()
```

Run Each Task Associated with an Artifact

Suppose you have a process model that adds several tasks to the process, but right now you only want to run the tasks associated with one specific artifact. You can use the function `generateProcessTasks`, but filter the list of IDs to only include task iterations associated with a specific model in the project, `AHRS_Voter.slx`.

Open the **Process Advisor** example project, which contains an example process model.

```
processAdvisorExampleStart
```

Use `padv.Artifact` to specify the project artifact that you want the task to run on. For this example, the artifact type is `sl_model_file` because the artifact is a Simulink model and the address is the path to `AHRS_Voter.slx`, relative to the project root.

```
artifactType = "sl_model_file";  
address = fullfile("02_Models","AHRS_Voter","specification","AHRS_Voter.slx");  
artifact = padv.Artifact(artifactType,address);
```

Get a list of the IDs for the task iterations in the MBD pipeline, but filter the list to include only task iterations associated with the artifact `AHRS_Voter.slx`.

```
IDs_AHRS_Voter = generateProcessTasks(FilterArtifact=artifact);
```

Use the function `runprocess` to run only the task iterations associated with the artifact `AHRS_Voter.slx`.

```
runprocess(Tasks=IDs_AHRS_Voter)
```

When you specify the `Tasks` value as a list of IDs for task iterations, the function `runprocess` runs only the specified task iterations. For this example, `runprocess` runs only the task iterations associated with the artifact `AHRS_Voter.slx`.

Note Alternatively, instead of generating and then running the task iterations, you can directly specify the `FilterArtifact` argument of the `runprocess` function to run the tasks associated with the artifact:

```
runprocess(FilterArtifact = fullfile("02_Models","AHRS_Voter","specification","AHRS_Voter.slx"))
```

But note that you can only run the tasks if the tasks are defined in the process model and the artifacts exist in the project.

Input Arguments

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `generateProcessTasks(Tasks = "padv.builtin.task.GenerateSimulinkWebView")`

FilterArtifact — Artifacts that you want to run tasks for

`string.empty` (default) | `string` | `padv.Artifact` object | array of `padv.Artifact` objects

Artifact or artifacts that you want to generate IDs for, specified as either the full path to an artifact, relative path to an artifact, a `padv.Artifact` object that represents an artifact, or an array of `padv.Artifact` objects.

Example: `fullfile("C:\", "User", "projectA", "myModel.slx")`

Example: `fullfile("02_Models", "AHRS_Voter", "specification", "AHRS_Voter.slx")`

Example:

```
padv.Artifact("sl_model_file",fullfile("02_Models","AHRS_Voter","specification", "AHRS_Voter.slx"))
```

Data Types: string

Process — Name of process that you want to generate IDs for

`padv.ProcessModel.DefaultProcessId` (default) | character vector | string

Name of process that you want to generate IDs for, specified by a character vector or string.

Example: "CIPipeline"

Data Types: char | string

Subprocesses — Names of subprocesses that you want to generate IDs for

character vector | cell array of character vectors | string | string array

Names of subprocesses that you want to generate IDs for, specified as a character vector, cell array of character vectors, string, or string array. The subprocess name is defined by the Name property of the subprocess.

Example: "SubprocessA"

Example: ["SubprocessA", SubprocessB"]

Data Types: char | string

Tasks — Names of tasks that you want to generate IDs for

character vector | cell array of character vectors | string | string array

Names of tasks that you want to generate IDs for, specified as a character vector, cell array of character vectors, string, or string array. The task name is defined by the Name property of the task.

Example: "padv.builtin.task.GenerateSimulinkWebView"

Example: ["padv.builtin.task.GenerateSimulinkWebView", ...
"padv.builtin.task.RunModelStandards"]

Data Types: char | string

Output Arguments

IDs — Identifiers for task iterations defined by process model

string

Identifiers for task iterations in the MBD pipeline, returned as a string.

IDs take the form: "*taskNameOrObject* | *fileType* | *relativePath*", where *relativePath* is the path relative to the project root.

Example IDs:

- "myCustomProjectTask|project|ProcessAdvisorExample.prj"
- "padv.builtin.task.RunModelStandards|sl_model_file|02_Models/AHRS_Voter/specification/AHRS_Voter.slx"
- "padv.builtin.task.RunTestsPerTestCase|sl_test_case|ced877ff-cfb8-4fa8-9bbf-aaa29b1d926b"

Alternative Functionality

App

You can also use the Process Advisor app to run individual task iterations in the process or to view task iterations for a specific model.

- To open the Process Advisor app for a project, in the MATLAB Command Window, enter:
`processAdvisorWindow`
- To open the Process Advisor app for a specific model, provide the name of the model, *modelName*, to the function `processadvisor`:

```
processadvisor(modelName)
```

See Also

`createProcessTaskID` | `getProcessTaskResults` | Process Advisor | `processadvisor` | `processAdvisorWindow` | `runprocess`

Topics

“Automate and Run Tasks with Process Advisor”
“Programmatically Run Tasks”

getProcessTaskResults

Get available task results and result details for task iterations in MBD pipeline

Syntax

```
[IDsWithTaskResults,taskResults,taskResultsOutdated] =
getProcessTaskResults()
[IDsWithTaskResults,taskResults,taskResultsOutdated] = getProcessTaskResults(
Name=Value)
```

Description

[IDsWithTaskResults,taskResults,taskResultsOutdated] = getProcessTaskResults() returns available task results and result details for the task iterations in the MBD pipeline. The function returns the identifiers for task iterations that have task results, `IDsWithTaskResults`, the current task results, `taskResults`, and a logical value that indicates if the task results are outdated, `taskResultsOutdated`.

If you do not have task results, use the function `runprocess` to run tasks and generate results. The function `getProcessTaskResults` only returns information related to task iterations that are defined in the process model. If you have task results from a task iteration that is not in the process model, the function does not return information related to those task results.

This function requires CI/CD Automation for Simulink Check.

[IDsWithTaskResults,taskResults,taskResultsOutdated] = getProcessTaskResults(Name=Value) specifies options using one or more name-value arguments.

Examples

Get Output Artifacts from Task Results

Get the available task results for a task iteration and use the result details to find information about the output artifacts of the task iteration.

Open the Process Advisor example project, which contains an example process model.

```
processAdvisorExampleStart
```

List the IDs for each task iteration in the MBD pipeline.

```
IDs = generateProcessTasks();
```

Run the first task iteration in the list.

```
runprocess(Tasks=IDs(1))
```

For this example, the build system runs the task `padv.builtin.task.GenerateSimulinkWebView` for the model `AHRS_Voter.slx`.

Get the available task results and result details.

```
[IDsWithResults,results,outedated] = getProcessTaskResults()
```

```
IDsWithResults =
```

```
    "padv.builtin.task.GenerateSimulinkWebView|sl_model_file|02_Models/AHRS_Voter/specification/"
```

```
results =
```

```
    TaskResult with properties:
```

```
        Status: Pass
    OutputArtifacts: [1x1 padv.Artifact]
        Details: [1x1 struct]
    ResultValues: [1x1 struct]
```

```
outedated =
```

```
    logical
```

```
    0
```

Get the output artifacts from the result. For this example, the result is a Simulink Web View for the model `AHRS_Voter.slx`.

```
webView = results.OutputArtifacts
```

```
webView =
```

```
    Artifact with properties:
```

```
        Type: "padv_output_file"
    Parent: [0x0 padv.Artifact]
    ArtifactAddress: [1x1 padv.util.ArtifactAddress]
    Alias: ""
```

Get Output Artifacts from Task Results for Specific Model

Get the available task results for a specific model.

Open the Process Advisor example project, which contains an example process model.

```
processAdvisorExampleStart
```

Check modeling standards for the model `AHRS_Voter.slx` by using the built-in task `padv.builtin.task.RunModelStandards`. The task uses Model Advisor to run checks on the model.

```
runprocess(...
Tasks = "padv.builtin.task.RunModelStandards",...
FilterArtifact = fullfile("02_Models","AHRS_Voter","specification","AHRS_Voter.slx"));
```

Get the task results and result details.

```
[IDsWithResults,results,outdated] = getProcessTaskResults(...
Tasks = "padv.builtin.task.RunModelStandards",...
FilterArtifact = fullfile("02_Models","AHRV_Voter","specification","AHRV_Voter.slx"))

IDsWithResults =

    "padv.builtin.task.RunModelStandards|sl_model_file|ProcessAdvisorExample|02_Models/AHRV_Voter

results =

    TaskResult with properties:

        Status: Pass
    OutputArtifacts: [1x1 padv.Artifact]
        Details: [1x1 struct]
    ResultValues: [1x1 struct]

outdated =

    logical

    0
```

Input Arguments

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `[~,results,~] = getProcessTaskResults(Tasks="maTask", FilterArtifact=fullfile("models","myModel.slx"));`

Tasks — Names of tasks that you want to run

character vector | cell array of character vectors | string | string array

Names of tasks that you want to run, specified as a character vector, cell array of character vectors, string, or string array. The task name is defined by the `Name` property of the task.

Alternatively, you can specify the task iteration IDs for individual task iterations that you want to run. See `generateProcessTasks` and `createProcessTaskID`.

Note You can only run tasks that are defined in the process model.

Example: `"padv.builtin.task.GenerateSimulinkWebView"`

Example: `["padv.builtin.task.GenerateSimulinkWebView",... "padv.builtin.task.RunModelStandards"]`

Data Types: `char` | `string`

Process — Name of process that you want to run

`padv.ProcessModel.DefaultProcessId` (default) | character vector | string

Name of process that you want to run, specified by a character vector or string.

Example: "CIPipeline"

Data Types: char | string

Subprocesses — Names of subprocesses that you want to run

character vector | cell array of character vectors | string | string array

Names of subprocesses that you want to run, specified as a character vector, cell array of character vectors, string, or string array. The subprocess name is defined by the `Name` property of the subprocess.

Example: "SubprocessA"

Example: ["SubprocessA", SubprocessB"]

Data Types: char | string

FilterArtifact — Artifacts that you want to run tasks for

string.empty (default) | string | padv.Artifact object | array of padv.Artifact objects

Artifact or artifacts that you want to run tasks for, specified as either the full path to an artifact, relative path to an artifact, a `padv.Artifact` object that represents an artifact, or an array of `padv.Artifact` objects.

Example: `fullfile("C:\", "User", "projectA", "myModel.slx")`

Example: `fullfile("02_Models", "AHRV_Voter", "specification", "AHRV_Voter.slx")`

Example:

`padv.Artifact("sl_model_file", fullfile("02_Models", "AHRV_Voter", "specification", "AHRV_Voter.slx"))`

Data Types: string

Output Arguments

IDsWithTaskResults — Identifiers for task iterations that have task results and are defined in process model

string | string array

Identifiers for task iterations that have task results and are defined in the process model, returned as a string or string array.

- If you do not have task results for task iterations in your process model, `IDsWithTaskResults` returns an empty array, []. You can use the function `runprocess` to run tasks and generate results.
- If you have task results for task iterations that are not in your process model, `IDsWithTaskResults` returns an empty array, [].
- If you have task results for task iterations that are in your process model, `IDsWithTaskResults` returns the IDs for the task iterations that have task results.

IDs take the form: "*taskNameOrObject* | *fileType* | *relativePath*", where *relativePath* is the path relative to the project root.

Example IDs:

- "myCustomProjectTask|project|ProcessAdvisorExample.prj"
- "padv.builtin.task.RunModelStandards|sl_model_file|02_Models/AHRS_Voter/specification/AHRS_Voter.slx"
- "padv.builtin.task.RunTestsPerTestCase|sl_test_case|ced877ff-cfb8-4fa8-9bbf-aaa29b1d926b"

taskResults – Results for task iterations

padv.TaskResult | padv.TaskResult array

Results for task iterations, returned as a padv.TaskResult or padv.TaskResult array.

- If you do not have task results for task iterations in your process model, taskResults returns an empty array, [].
- If you have task results for task iterations that are not in your process model, taskResults returns an empty array, [].
- If you have task results for task iterations that are in your process model, taskResults returns a padv.TaskResult or padv.TaskResult array.

padv.TaskResult objects contain properties for the result status, output artifacts, details, and result values for the number of passing, warning, and failing results for task iterations.

taskResultsOutdated – Whether task results are outdated or up-to-date

logical | logical array

Status of task results, returned as a logical value or logical array. Values of 1 indicate that the results for the task iteration are outdated and might not reflect the current state of the project or task. Values of 0 indicate that the results for the task iteration are up-to-date. The result is an empty array, [], when there are not task results.

See Also

createProcessTaskID | generateProcessTasks | Process Advisor | processadvisor | processAdvisorWindow | runprocess

Topics

“Automate and Run Tasks with Process Advisor”
 “Programmatically Run Tasks”

padv.BuildResult

Result from build system build

Description

This object requires CI/CD Automation for Simulink Check. Use the build result, `padv.BuildResult`, to find the properties of the build system build, including a list of the tasks that the build system ran and the settings the build system used.

Creation

Syntax

```
buildResultObj = padv.BuildResult()
```

Description

`buildResultObj = padv.BuildResult()` stores the results from a build system build.

Properties

StartTime — Start time of build

`datetime`

Start time of build, returned as `datetime`.

Example: 09-Aug-2022 14:32:05

Data Types: `datetime`

EndTime — End time of build

`datetime`

End time of build, returned as `datetime`.

Example: 09-Aug-2022 14:32:37

Data Types: `datetime`

Status — Overall status for build

`Pass (default) | Fail | Error`

Overall status for build, returned as the `padv.TaskStatus` enumeration value:

- `Error` if a task iteration in the build returns an error.
- `Fail` if none of the task iterations in the build return an error, but at least one task iteration fails.
- `Pass` if none of the task iterations were run, or if none of the task iterations in the build return an error or fail.

Example: Pass

ResultValues — Task iteration result values

[1×1 struct] (default)

Task iteration result values, returned as a structure array with fields:

- Pass
- Warn
- Fail

For example, if the build runs one task iteration and the task iteration returns one passing result and five warning results, the structure array contains:

```
struct with fields:
    Pass: 1
    Warn: 5
    Fail: 0
```

Data Types: struct

PassTasks — IDs for task iterations that passed during the build

cell array

IDs for task iterations that passed during the build, returned as a cell array.

If the build system runs one task iteration and the task iteration passes, `PassTasks` returns a one-dimensional cell array. For example, if the build system only ran the task `padv.builtin.task.GenerateCode` on the model `AHRS_Voter.slx` and the task iteration passed, `PassTasks` returns:

```
{'padv.builtin.task.GenerateCode|sl_model_file|02_Models/AHRS_Voter/specification/AHRS_Voter.slx'}
```

If multiple task iterations pass, `PassTasks` returns one cell for each task iteration that passed. For example:

```
{'padv.builtin.task.GenerateCode|sl_model_file|02_Models/AHRS_Voter/specification/AHRS_Voter.slx',
'padv.builtin.task.GenerateCode|sl_model_file|02_Models/Actuator_Control/specification/Actuator_Control.slx',
'padv.builtin.task.GenerateCode|sl_model_file|02_Models/Flight_Control/specification/Flight_Control.slx',
'padv.builtin.task.GenerateCode|sl_model_file|02_Models/InnerLoop_Control/specification/InnerLoop_Control.slx',
'padv.builtin.task.GenerateCode|sl_model_file|02_Models/OuterLoop_Control/specification/OuterLoop_Control.slx'}
```

Data Types: cell

ErrorTasks — IDs for task iterations that returned an error during the build

cell array

IDs for task iterations that returned an error during the build, returned as a cell array.

If the build system runs one task iteration and the task iteration returns an error, `ErrorTasks` returns a one-dimensional cell array. For example, if the build system tried to run a custom task, `customTask`, on the model `AHRS_Voter.slx`, but the task iteration returned an error, `ErrorTasks` returns:

```
{'customTask|sl_model_file|02_Models/AHRS_Voter/specification/AHRS_Voter.slx'}
```

If multiple task iterations error, `ErrorTasks` returns one cell for each task iteration that returned an error. For example:

```
{'customTask|sl_model_file|02_Models/AHRS_Voter/specification/AHRS_Voter.slx' }
{'customTask|sl_model_file|02_Models/Actuator_Control/specification/Actuator_Control.slx' }
{'customTask|sl_model_file|02_Models/Flight_Control/specification/Flight_Control.slx' }
{'customTask|sl_model_file|02_Models/InnerLoop_Control/specification/InnerLoop_Control.slx'}
{'customTask|sl_model_file|02_Models/OuterLoop_Control/specification/OuterLoop_Control.slx'}
```

Data Types: `cell`

SkippedTasks — IDs for task iterations that the build system skipped

`cell array`

IDs for task iterations that the build system skipped, returned as a cell array. The build system skips task iterations that already have up-to-date results, unless you specify `Force` as `true` when you call the function `runprocess`.

If the build system skips one task iteration, `SkippedTasks` returns a one-dimensional cell array. For example, if you instructed the build system to run the task `padv.builtin.task.GenerateCode` on the model `AHRS_Voter.slx`, but the task iteration already had up-to-date results, `SkippedTasks` returns:

```
{'padv.builtin.task.GenerateCode|sl_model_file|02_Models/AHRS_Voter/specification/AHRS_Voter.slx'}
```

If the build system skips multiple task iterations, `SkippedTasks` returns one cell for each task iteration that the build system skipped. For example:

```
{'padv.builtin.task.GenerateCode|sl_model_file|02_Models/AHRS_Voter/specification/AHRS_Voter.slx'
{'padv.builtin.task.GenerateCode|sl_model_file|02_Models/Actuator_Control/specification/Actuator_
{'padv.builtin.task.GenerateCode|sl_model_file|02_Models/Flight_Control/specification/Flight_Con
{'padv.builtin.task.GenerateCode|sl_model_file|02_Models/InnerLoop_Control/specification/InnerLo
{'padv.builtin.task.GenerateCode|sl_model_file|02_Models/OuterLoop_Control/specification/OuterLo
```

Data Types: `cell`

FailedTasks — IDs for task iterations that failed during the build

`cell array`

IDs for task iterations that failed during the build, returned as a cell array.

If the build system runs only one task iteration and the task iteration fails, `FailedTasks` returns a one-dimensional cell array. For example, if the build system ran the task `padv.builtin.task.GenerateCode` on the model `AHRS_Voter.slx` and the task iteration failed, `FailedTasks` returns:

```
{'padv.builtin.task.GenerateCode|sl_model_file|02_Models/AHRS_Voter/specification/AHRS_Voter.slx'}
```

If multiple task iterations fail, `FailedTasks` returns one cell for each task iteration that failed. For example:

```
{'padv.builtin.task.GenerateCode|sl_model_file|02_Models/AHRS_Voter/specification/AHRS_Voter.slx'
{'padv.builtin.task.GenerateCode|sl_model_file|02_Models/Actuator_Control/specification/Actuator_
{'padv.builtin.task.GenerateCode|sl_model_file|02_Models/Flight_Control/specification/Flight_Con
{'padv.builtin.task.GenerateCode|sl_model_file|02_Models/InnerLoop_Control/specification/InnerLo
{'padv.builtin.task.GenerateCode|sl_model_file|02_Models/OuterLoop_Control/specification/OuterLo
```

Data Types: `cell`

InputArgs — Input arguments that defined how the build system ran the build

[1×1 struct] (default) | structure array

Input arguments that defined how the build system ran the build, returned as a structure array with fields:

- **TasksToBuild** — List of task iteration IDs that you want the build system to run
- **Isolation** — Setting to include or ignore task dependencies
- **Force** — Setting to skip or run up-to-date task iterations
- **RerunFailedTasks** — Setting to ignore or rerun failed task iterations
- **RerunErroredTasks** — Setting to ignore or rerun task iterations that returned an error

For example, the `InputArgs` for a build result could return:

```
struct with fields:
    TasksToBuild: [1×5 string]
    Isolation: 0
    Force: 0
    RerunFailedTasks: 0
    RerunErroredTasks: 0
```

For more information, see `runprocess`.

Data Types: struct

Examples**Get List of Passed Task Iterations and Build Settings**

Open a project, run a build, and use the build result, `padv.BuildResult`, to get a list of the passed task iterations and the settings that the build system used when running the build.

Open the Process Advisor example project, which contains an example process model.

```
processAdvisorExampleStart
```

Generate a list of the tasks defined by the process model.

```
tasks = generateProcessTasks;
```

Run the first five task iterations in `tasks` and specify `Force` as `true`.

```
buildResult = runprocess(Force=true,Tasks=tasks(1:5))
```

Use the build result, `buildResult`, to get a list of the task iterations that passed.

```
passed = buildResult.PassTasks'
```

```
passed =
```

```
5×1 cell array
```

```
{'padv.builtin.task.GenerateSimulinkWebView|sl_model_file|02_Models/AHRS_Voter/specification,
{'padv.builtin.task.GenerateSimulinkWebView|sl_model_file|02_Models/Actuator_Control/specifi
```

```
{'padv.builtin.task.GenerateSimulinkWebView|sl_model_file|02_Models/Flight_Control/specifica
{'padv.builtin.task.GenerateSimulinkWebView|sl_model_file|02_Models/InnerLoop_Control/specif
{'padv.builtin.task.GenerateSimulinkWebView|sl_model_file|02_Models/OuterLoop_Control/specif
```

When you used the function `runprocess`, you specified `Force` as `true`. You can see that information in the `InputArgs` property of the build result, `buildResult`.

```
runprocessInputs = buildResult.InputArgs
```

```
runprocessInputs =
```

```
    struct with fields:
```

```
        TasksToBuild: ["padv.builtin.task.GenerateSimulinkWebView|sl_model_file|02_Models/AHRS_V
            Isolation: 0
            Force: 1
        RerunFailedTasks: 0
        RerunErroredTasks: 0
```

The build result shows that the `Force` setting was `1` (`true`) when the build system ran.

You can export a build report that summarizes the task statuses, task results, and other information about the task execution by using `padv.ProcessAdvisorReportGenerator` to specify options for the report and `generateReport` to generate the report.

```
rptObj = padv.ProcessAdvisorReportGenerator;
generateReport(rptObj)
```

Alternatively, in the Process Advisor toolstrip, in the **Export** section, click **Report**. In the Export Report dialog box, you can specify options for the report and export a report for the current process by clicking **Export**.

See Also

[generateProcessTasks](#) | [generateReport](#) | [runprocess](#) | [Process Advisor](#)

Topics

“Automate and Run Tasks with Process Advisor”

“Programmatically Run Tasks”

“Specify Settings for Process Advisor and Build System”

“Best Practices for Effective Builds”

padv.Preferences

(To be removed) Specify settings for build system

Description

This object requires CI/CD Automation for Simulink Check. There are several settings that you can use to customize the behavior of the build system. These behaviors impact how the Process Advisor app and `runprocess` function run tasks. For example, you can use settings to use incremental builds, enable model caching, and customize other behaviors. The build system saves these settings in `padv.Preferences`. You can use the preferences, `padv.Preferences`, to specify settings for the Process Advisor app and settings for how the `runprocess` function runs builds.

Note The `padv.Preferences` class will be removed in a future release. Use the new classes `padv.ProjectSettings` and `padv.UserSettings` instead. The new classes allow you to programmatically control the settings for incremental builds, build system logging, and other behaviors, without needing to create a project startup script to persist run-time settings.

For information, see the “Version History” on page 3-34 for `padv.Preferences`.

Creation

Syntax

Description

`P = padv.Preferences()` gets the handle to the global preferences object, `P`. There is only one set of preference properties.

The `padv.Preferences` class is a `handle` class.

Properties

Project Settings

These settings are stored in the project and are shared with everyone using the project.

IncrementalBuild — Automatically detect changes and mark task results as outdated

`1 (true) | 0 (false)`

Automatically detect changes and mark task results as outdated, specified as a numeric or logical `1 (true)` or `0 (false)`.

When `IncrementalBuild` is `true` and you make a change to an artifact in your project, the build system marks impacted task results as outdated.

This property is equivalent to the **Incremental build** setting in the Process Advisor Settings dialog box.

Example: `true`

Data Types: `logical`

EnableModelCaching — Allow build system to cache models during build

0 (false) (default) | 1 (true)

Allow the build system to cache models during a build, specified as a numeric or logical 1 (true) or 0 (false).

If you specify the property `EnableModelCaching` as `true`, you allow the build system to cache models instead of reloading the same models multiple times within a build. For more information, see “Cache Models and Other Artifacts Used During Build”.

This property is equivalent to the **Enable model caching** setting in the Process Advisor Settings dialog box.

Example: `true`

Data Types: `logical`

MaxNumModelsInCache — Maximum number of models in cache

1 (default) | positive value

Maximum number of models in the model cache, specified as a positive value.

Example: 2

MaxNumTestResultsInCache — Maximum number of test results in cache

20 (default) | positive value

Maximum number of test results in the cache, specified as a positive value.

Example: 30

SuppressOutputWhenInteractive — Suppress command-line output from Process Advisor

0 (false) (default) | 1 (true)

Suppress command-line output from Process Advisor during interactive MATLAB sessions, specified as a numeric or logical 1 (true) or 0 (false).

You can use this setting to suppress command-line outputs from the build system, such as the build log and task execution messages from Process Advisor and the `runprocess` function.

Note that the build system automatically ignores this setting when you run MATLAB in batch mode, which is typically the case for CI systems.

This property is equivalent to the **Suppress outputs to command window** setting in the Process Advisor Settings dialog box.

Example: `true`

Data Types: `logical`

Run-Time Settings

DetectDuplicateOutputs — Generate error message when multiple tasks attempt to write to same output file

1 (true) (default) | 0 (false)

Setting that controls whether the build system generates an error message when multiple tasks attempt to write to the same output file, specified as a numeric or logical 1 (`true`) or 0 (`false`).

By default, the build system generates an error if multiple tasks attempt to write to the same output file.

This property is equivalent to the **Detect duplicate outputs** setting in the Process Advisor Settings dialog box.

Example: `false`

Data Types: `logical`

GarbageCollectTaskOutputs — Setting for automatically cleaning task results for tasks and artifacts that do not match current process model or project

`true` or 1 (default) | `false` or 0

Setting for automatically cleaning task results for tasks and artifacts that do not match current process model or project, specified as a numeric or logical 1 (`true`) or 0 (`false`).

By default, when you use the build system, the build system cleans task results that are not relevant for the current process model or project. For example, if you had task results from a specific task and then you remove that task from the process model, the build system automatically deletes the task results associated with the task. If you had task results associated with a specific project artifact and then you removed that artifact from the project, the build system automatically deletes the task results associated with the artifact. Note that the build system does not delete generated artifacts like generated code.

If you specify `GarbageCollectTaskOutputs` as `false`, the build system does not automatically clean task results associated with tasks and artifacts that are not in the current process model or project.

This property is equivalent to the **Garbage collect task outputs** setting in the Process Advisor Settings dialog box.

Example: `false`

Data Types: `logical`

FilteredDigitalThreadMessages — List of filtered digital thread messages

[13×1 `string`] (default) | `string`

List of filtered digital thread messages, specified as a string.

By default, Process Advisor and the build system do not display certain messages from the digital thread. You can add or remove messages in the list, or reset the list of filtered messages, by using the `padv.Preferences` object functions. For information, see the “Object Functions” on page 3-32 for `padv.Preferences`.

Data Types: `string`

ShowDetailedErrorMessage — Setting to show more information in error messages

`false` or 0 (default) | `true` or 1

Setting to show more information in error messages, specified as a numeric or logical 0 (`false`) or 1 (`true`).

By default, error messages from the build system are not verbose.

If you specify `ShowDetailedErrorMessages` as `true`, the build system shows full stack traces in error messages. You might want to see full stack traces when you are debugging a process model.

This property is equivalent to the **Show detailed error messages** setting in the Process Advisor Settings dialog box.

Example: `true`

Data Types: `logical`

TrackProcessModel — Setting for tracking changes to process model

`true` or `1` (default) | `false` or `0`

Setting for tracking changes to process model, specified as a numeric or logical `1` (`true`) or `0` (`false`).

By default, if you make a change to the process model file, the build system marks each task status and task result as outdated because the tasks in the updated process model might not match the tasks that generated the task results from the previous version of the process model. For example, if you ran the built-in task `padv.builtin.task.RunModelStandards` with the default Model Advisor configuration, updated the process model to specify a different Model Advisor configuration file for the task, and then ran the task again, the task results are now outdated because they are the task results from the default configuration.

If you specify `TrackProcessModel` as `false` and make a change to the process model, the build system will not mark the task statuses and task results as outdated.

This property is equivalent to the **Add process model as dependency** setting in the Process Advisor Settings dialog box.

Example: `false`

Data Types: `logical`

Object Functions

- `addFilteredDigitalThreadMessages(obj, IssueId)` adds the message, specified by the issue ID `IssueId`, to the list of filtered messages in the property `FilteredDigitalThreadMessages`.

To get a list of issue messages and issue IDs, use the function `getArtifactIssues`:

```
metric_engine = metric.Engine();
issues = getArtifactIssues(metric_engine)
issuesMessages = issues.IssueMessage
issueIDs = issues.IssueId
```

Suppose that you want to filter out the issue message associated with the issue ID `"alm:artifact_service:CannotResolveElement"`. You can use the function `addFilteredDigitalThreadMessages` to add the issue message to the list of filtered messages:

```
p = padv.Preferences;
addFilteredDigitalThreadMessages(p,...
"alm:artifact_service:CannotResolveElement")
```

- `removeFilteredDigitalThreadMessages(obj, IssueId)` removes the message, specified by `messageID`, to the list of filtered messages in the property `FilteredDigitalThreadMessages`.

For example:

```
p = padv.Preferences;
removeFilteredDigitalThreadMessages(p, ...
"alm:simulink_handlers:ModelCallbacksDeactivated")
```

- `resetFilteredDigitalThreadMessages(obj)` resets the list of filtered messages in the property `FilteredDigitalThreadMessages`.

For example:

```
p = padv.Preferences;
resetFilteredDigitalThreadMessages(p)
```

Examples

Specify Preferences for Builds

Use `padv.Preferences` to specify preferences for the Process Advisor app and build system.

Create a `padv.Preferences` object.

```
PREF = padv.Preferences
```

Specify `IncrementalBuild` as 0.

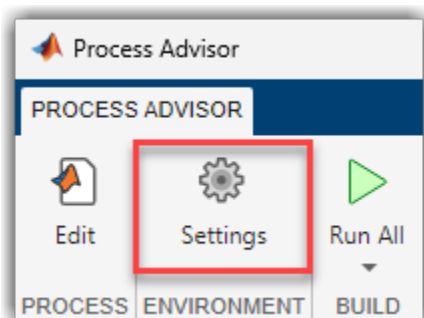
```
PREF.IncrementalBuild = 0;
```

Now, when you run tasks, incremental builds are disabled and the build system forces tasks to run, even if the tasks have up to date results.

Alternative Functionality

App

In Process Advisor, in the toolstrip, click **Settings** to access and change the settings for the build system.



Version History

R2022b: `padv.Preferences` class will be removed in a future release

Warns starting in R2022b

The class `padv.Preferences` will be removed in a future release. Update your code to replace instances of `padv.Preferences` with either `padv.UserSettings.get()` or `padv.ProjectSettings.get()`, depending on which property you need to access.

<code>padv.Preferences</code> Property	Update
DetectDuplicateOutputs	Replace instances of <code>padv.Preferences</code> with <code>padv.UserSettings.get()</code> .
GarbageCollectTaskOutputs	
ShowDetailedErrorMessages	
TrackProcessModel	
FilteredDigitalThreadMessages	Replace instances of <code>padv.Preferences</code> with <code>padv.ProjectSettings.get()</code> .
IncrementalBuild	
EnableModelCaching	
MaxNumModelsInCache	
MaxNumTestResultsInCache	
SuppressOutputWhenInteractive	

For example:

Functionality	Use This Instead
<pre>% changing run-time setting p1 = padv.Preferences; p1.DetectDuplicateOutputs = false;</pre>	<pre>p1 = padv.UserSettings.get(); p1.DetectDuplicateOutputs = false;</pre>
<pre>% changing project setting p1 = padv.Preferences; p1.IncrementalBuild = false;</pre>	<pre>p1 = padv.ProjectSettings.get(); p1.IncrementalBuild = false;</pre>

See Also

`padv.ProjectSettings` | `padv.UserSettings`

Topics

“Automate and Run Tasks with Process Advisor”

“Programmatically Run Tasks”

“Specify Settings for Process Advisor and Build System”

padv.ProjectSettings Class

Namespace: padv

Build system settings for project

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.ProjectSettings` class is a handle class.

Creation

Syntax

```
padv.ProjectSettings
```

Description

`padv.ProjectSettings` is a handle class that you can use to customize the behavior of the build system. These behaviors impact how the Process Advisor app and `runprocess` function run tasks. For example, you can use the project settings to use incremental builds, enable model caching, and customize other behaviors. For more information, see “Specify Settings for Process Advisor and Build System”.

Project settings are persistent, are stored in the project, and are shared with everyone using the project. There is only one set of project settings for a project. To get the active project settings object, use the `get` method.

To specify settings that apply only to your machine, use `padv.UserSettings` instead.







Properties

IncrementalBuild — Automatically detect changes and mark task results as outdated







```
1 (true) | 0 (false)
```

Automatically detect changes and mark task results as outdated, specified as either:

- `1 (true)` — When you make a change to an artifact in your project, Process Advisor and the build system mark impacted task results as outdated. Incremental builds can help you reduce the number of task iterations that you need to re-run by identifying and running only the task iterations with outdated results. If the task iteration results are up-to-date, the build system and the Process Advisor app skip the task iteration. In Process Advisor, the task status icons in the **Tasks** column indicate whether the task results are up-to-date or outdated. Up-to-date task results have task status icons that are green for tasks that pass and red for tasks that fail or generate errors. Outdated task results have task status icons that are gray.

Tasks	
	Task A (up-to-date)
	Task B (up-to-date)
	Task C (up-to-date)
	Task D (outdated)
	Task E (outdated)
	Task F (outdated)

- `0 (false)` — Process Advisor and the build system do not identify results as up-to-date or outdated, and effectively force run task iterations in the project. In Process Advisor, the task status icons in the **Tasks** column appear in black and white because the app is not showing which results are up-to-date or outdated. These statuses only indicate whether the task passed, failed, generated an error, or did not run.

Tasks	
	Task A (up-to-date)
	Task B (up-to-date)
	Task C (up-to-date)
	Task D (outdated)
	Task E (outdated)
	Task F (outdated)

This property is equivalent to the **Incremental build** setting in the Process Advisor Settings dialog box.

Example: `true`

Data Types: `logical`

EnableModelCaching — Allow build system to cache models during build

`0 (false) | 1 (true)`

Allow the build system to cache models during a build, specified as a numeric or logical 1 (true) or 0 (false).

If you specify the property `EnableModelCaching` as true, you allow the build system to cache models instead of reloading the same models multiple times within a build. For more information, see “Cache Models and Other Artifacts Used During Build”.

This property is equivalent to the **Enable model caching** setting in the Process Advisor Settings dialog box.

Example: true

Data Types: logical

MaxNumModelsInCache — Maximum number of models in cache

1 (default) | positive value

Maximum number of models in the model cache, specified as a positive value.

For more information, see “Cache Models and Other Artifacts Used During Build”.

Example: 2

MaxNumTestResultsInCache — Maximum number of test results in cache

20 (default) | positive value

Maximum number of test results in the cache, specified as a positive value.

For more information, see “Cache Models and Other Artifacts Used During Build”.

Example: 30

SuppressOutputWhenInteractive — Suppress command-line output from Process Advisor

0 (false) (default) | 1 (true)

Suppress command-line output from Process Advisor during interactive MATLAB sessions, specified as a numeric or logical 1 (true) or 0 (false).

You can use this setting to suppress command-line outputs from the build system, such as the build log and task execution messages from Process Advisor and the `runprocess` function.

Note that the build system automatically ignores this setting when you run MATLAB in batch mode, which is typically the case for CI systems.

This property is equivalent to the **Suppress outputs to command window** setting in the Process Advisor Settings dialog box. If you want to override this setting when you use the function `runprocess`, you can use the `runprocess` argument `SuppressOutputWhenInteractive`.

Example: true

Data Types: logical

ShowFileExtension — Show file extensions for task iteration artifacts

0 (false) (default) | 1 (true)

Show file extensions for task iteration artifacts, specified as a numeric or logical 1 (true) or 0 (false).

By default, queries strip file extensions from the `Alias` property of each task iteration artifact. The `Alias` property controls the display name for the artifact in the **Tasks** column in Process Advisor.

To show file extensions for all task iteration artifacts in the **Tasks** column, specify this setting as `true`. To keep file extensions in the results for a specific query, specify the query property `ShowFileExtension` as `true`.

This property is equivalent to the **Show file extensions** setting in the Process Advisor Settings dialog box.


Example: `true`


Data Types: `logical`

HandleUntrackedIO — Build system behavior when there are untracked I/O files

`"Warn"` (default) | `"Allow"` | `"Error"`

Build system behavior when there are untracked I/O files, specified as either:

- `"Allow"` — Do not generate warnings or errors for untracked I/O files.
- `"Warn"` — Generate a warning if a task has untracked I/O files. In Process Advisor, the **I/O** column shows a warning icon .
- `"Error"` — Generate an error if a task has untracked I/O files.

Note that if you make a change to an untracked file, Process Advisor and the build system *do not* mark the task as outdated. Make sure that task inputs or outputs that appear as **Untracked**  do not need to be tracked to maintain the task status and result information that you need for your project.

If you change the value of `HandleUntrackedIO`, the build system uses that behavior the next time you run a task. This property is equivalent to the **Untracked dependency behavior** setting in the Process Advisor Settings dialog box.

Example: `"Allow"`

FilteredDigitalThreadMessages — List of filtered digital thread messages

`[13×1 string]` (default) | `string`

List of filtered digital thread messages, specified as a string.

By default, Process Advisor and the build system do not display certain messages from the digital thread. You can add or remove messages in the list, or reset the list of filtered messages, by using the methods for `padv.ProjectSettings`. For information, see [Filter Messages](#).

Data Types: `string`

DetectMultipleProcessModels — Detect multiple process model files

`"Off"` (default) | `"Error"` | `"Warning"`

Detect multiple process model files, specified as either:

- `"Off"` — The build system does not generate an error or warning when there are multiple process model files on the project path.
- `"Error"` — The build system generates an error when there are multiple process model files on the project path.

- "Warning" — The build system generates a warning when there are multiple process model files on the project path.

To avoid unexpected behavior, make sure only one processmodel file is on the project path.

Example: "Warning"

Data Types: string

Methods

Public Methods

Get or Reset Settings for Project

Method	Description
get	Get build system settings for current project <pre>PREF = padv.ProjectSettings.get()</pre>
resetToDefaultValues	Reset build system settings for current project <pre>PREF.resetToDefaultValues()</pre> <p>To see the changes, use the get method to get the latest setting values.</p> <pre>PREF = padv.ProjectSettings.get()</pre>

Filter Messages

Method	Description
addFilteredDigitalThreadMessages	Add message to list of filtered messages <pre>ps = padv.ProjectSettings.get(); ps.addFilteredDigitalThreadMessages(... "alm:artifact_service:CannotResolveElement");</pre> <p>To get a list of issue messages and issue IDs, use the function getArtifactIssues:</p> <pre>metric_engine = metric.Engine(); issues = getArtifactIssues(metric_engine) issuesMessages = issues.IssueMessage issueIDs = issues.IssueId</pre>
removeFilteredDigitalThreadMessages	Remove message from list of filtered messages <pre>ps = padv.ProjectSettings.get(); ps.removeFilteredDigitalThreadMessages(... "alm:simulink_handlers:ModelCallbacksDeactivated");</pre>
resetFilteredDigitalThreadMessages	Reset list of filtered messages <pre>ps = padv.ProjectSettings.get(); ps.resetFilteredDigitalThreadMessages();</pre>

Examples

Get Build System Settings for Project

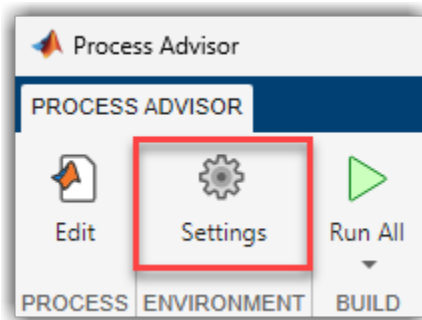
Get for build system settings for the currently open project.

```
PREF = padv.ProjectSettings.get()
```

Alternative Functionality

App

In Process Advisor, in the toolstrip, click **Settings** to access and change the settings for the build system.



For more information, see “Specify Settings for Process Advisor and Build System”.

See Also

Process Advisor | `padv.UserSettings`

Topics

“Automate and Run Tasks with Process Advisor”

“Programmatically Run Tasks”

“Specify Settings for Process Advisor and Build System”

padv.UserSettings Class

Namespace: padv

Build system settings for user

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.UserSettings` class is a `handle` class.

Creation

Syntax

```
padv.UserSettings
```

Description

`padv.UserSettings` is a `handle` class that you can use to customize the behavior of the build system on your machine. These behaviors impact how the Process Advisor app and `runprocess` function run tasks on your machine. For example, you can use the user settings to show detailed error messages, remove the process model as a dependency, and customize other behaviors. For more information, see “Specify Settings for Process Advisor and Build System”.

User settings are persistent and do not reset when you restart MATLAB or call `clear classes`. There is only one set of user settings. To get the active user settings object, use the `get` method.

To specify settings that apply to everyone that uses your project, use `padv.ProjectSettings` instead.

Properties

DetectDuplicateOutputs — Generate error message when multiple tasks attempt to write to same output file

1 (true) (default) | 0 (false)

Setting that controls whether the build system generates an error message when multiple tasks attempt to write to the same output file, specified as a numeric or logical 1 (true) or 0 (false).

By default, the build system generates an error if multiple tasks attempt to write to the same output file.

This property is equivalent to the **Detect duplicate outputs** setting in the Process Advisor Settings dialog box.

Example: `false`

Data Types: `logical`

GarbageCollectTaskOutputs — Setting for automatically cleaning task results for tasks and artifacts that do not match current process model or project`true or 1 (default) | false or 0`

Setting for automatically cleaning task results for tasks and artifacts that do not match current process model or project, specified as a numeric or logical 1 (`true`) or 0 (`false`).

By default, when you use the build system, the build system cleans task results that are not relevant for the current process model or project. For example, if you had task results from a specific task and then you remove that task from the process model, the build system automatically deletes the task results associated with the task. If you had task results associated with a specific project artifact and then you removed that artifact from the project, the build system automatically deletes the task results associated with the artifact. Note that the build system does not delete generated artifacts like generated code.

If you specify `GarbageCollectTaskOutputs` as `false`, the build system does not automatically clean task results associated with tasks and artifacts that are not in the current process model or project.

This property is equivalent to the **Garbage collect task outputs** setting in the Process Advisor Settings dialog box.

Example: `false`

Data Types: `logical`

ShowDetailedErrorMessages — Setting to show more information in error messages`false or 0 (default) | true or 1`

Setting to show more information in error messages, specified as a numeric or logical 0 (`false`) or 1 (`true`).

By default, error messages from the build system are not verbose.

If you specify `ShowDetailedErrorMessages` as `true`, the build system shows full stack traces in error messages. You might want to see full stack traces when you are debugging a process model.

This property is equivalent to the **Show detailed error messages** setting in the Process Advisor Settings dialog box.

Example: `true`

Data Types: `logical`

TrackProcessModel — Setting for tracking changes to process model`true or 1 (default) | false or 0`

Setting for tracking changes to process model, specified as a numeric or logical 1 (`true`) or 0 (`false`).

By default, if you make a change to the process model file, the build system marks each task status and task result as outdated because the tasks in the updated process model might not match the tasks that generated the task results from the previous version of the process model. For example, if you ran the built-in task `padv.builtin.task.RunModelStandards` with the default Model Advisor configuration, updated the process model to specify a different Model Advisor configuration file for the task, and then ran the task again, the task results are now outdated because they are the task results from the default configuration.

If you specify `TrackProcessModel` as `false` and make a change to the process model, the build system will not mark the task statuses and task results as outdated.

This property is equivalent to the **Add process model as dependency** setting in the Process Advisor Settings dialog box.



Example: `false`

Data Types: `logical`

DryRunDefaultMode — Setting that controls whether Process Advisor runs or dry runs tasks by default

0 (`false`) (default) | 1 (`true`)

Setting that controls whether the Process Advisor app runs or dry runs tasks by default, specified as either:

- 0 (`false`) — Process Advisor shows a run button  for each task and the **Run All** button in the toolstrip.
- 1 (`true`) — Process Advisor shows a dry run button  for each task and the **Dry Run All** button in the toolstrip.

You can dry run a task to quickly validate task inputs and generate representative outputs without performing the full task operation. The built-in tasks each have a `dryRun` method defined to help evaluate the task inputs and outputs for that specific task. Dry runs can be helpful for quickly testing out your process model and CI pipeline. For more information, see “Dry Run Tasks to Test Process Model”.

Note This property does not impact the `runprocess` function. To perform a dry run with the `runprocess` function, specify the `DryRun` argument of the function.

This setting is the same as the **Run All > Set Dry Run as Default** option in the Process Advisor toolstrip.

Example: `true`

Data Types: `logical`

Methods

Public Methods

Get Settings for User

Method	Description
get	Get build system settings for current user <code>PREF = padv.UserSettings.get()</code>
resetToDefaultValues	Reset build system settings for current user <code>PREF.resetToDefaultValues()</code> To see the changes, use the get method to get the latest setting values. <code>PREF = padv.UserSettings.get()</code>

Examples

Get Build System Settings for User

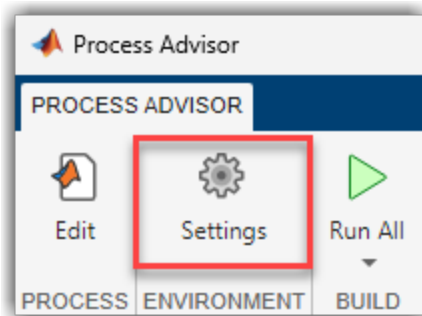
Get for build system settings for the current user.

```
PREF = padv.UserSettings.get()
```

Alternative Functionality

App

In Process Advisor, in the toolbar, click **Settings** to access and change the settings for the build system.



For more information, see “Specify Settings for Process Advisor and Build System”.

See Also

Process Advisor | `padv.ProjectSettings`

Topics

“Automate and Run Tasks with Process Advisor”

“Programmatically Run Tasks”

“Specify Settings for Process Advisor and Build System”

Pipeline Generator

padv.pipeline.generatePipeline

Namespace: padv.pipeline

Generate pipeline file for CI platform

Syntax

```
generatorResults = padv.pipeline.generatePipeline(pipelineGeneratorOptions)
generatorResults = padv.pipeline.generatePipeline( ____,processName)
```

Description

`generatorResults = padv.pipeline.generatePipeline(pipelineGeneratorOptions)` generates a pipeline file for the CI platform and options specified by `pipelineGeneratorOptions`. The function `padv.pipeline.generatePipeline` is a pipeline generator that can automatically generate a pipeline file. The generated pipeline file can configure a pipeline that runs your process in CI.

`generatorResults = padv.pipeline.generatePipeline(____,processName)` generates a pipeline file for the process specified by `processName`. By default, the pipeline generator generates a pipeline file for the default process in the process model.

Examples

Generate YML File for GitHub Pipeline

Suppose that you want to run your process using GitHub®.

```
padv.pipeline.generatePipeline(padv.pipeline.GitHubOptions)
```

The generated pipeline file is 'simulink_pipeline.yml'.

For information on how to use the pipeline generator to integrate into GitHub, see “Integrate Process into GitHub”.

Generate YML File for GitLab Pipeline

Suppose that you want to run your process using GitLab®.

```
padv.pipeline.generatePipeline(padv.pipeline.GitLabOptions)
```

The generated pipeline file is 'simulink_pipeline.yml'.

For information on how to use the pipeline generator to integrate into GitLab, see “Integrate Process into GitLab”.

Generate Jenkinsfile for Jenkins Pipeline

Suppose that you want to run your process using Jenkins®.

```
padv.pipeline.generatePipeline(padv.pipeline.JenkinsOptions)
```

The generated pipeline file is 'simulink_pipeline'.

For information on how to use the pipeline generator to integrate into Jenkins, see “Integrate Process into Jenkins”.

Input Arguments

pipelineGeneratorOptions — Options for generating CI pipeline

```
padv.pipeline.GitHubOptions object | padv.pipeline.GitLabOptions object |
padv.pipeline.JenkinsOptions object
```

Options for generating CI pipeline, specified as either a:

- `padv.pipeline.GitHubOptions` object — Generate a YML file for GitHub. For more information, see “Integrate Process into GitHub”.
- `padv.pipeline.GitLabOptions` object — Generate a YML file for GitLab. For more information, see “Integrate Process into GitLab”.
- `padv.pipeline.JenkinsOptions` object — Generate a Jenkinsfile for Jenkins. For more information, see “Integrate Process into Jenkins”.

Example: `padv.pipeline.generatePipeline(padv.pipeline.GitLabOptions)`

processName — Process name

```
string
```

Process name, specified as a string.

Example: "CIPipeline"

Data Types: `string`

Output Arguments

generatorResults — Results from pipeline generator

```
padv.pipeline.GeneratorResults object
```

Results from pipeline generator, returned as a `padv.pipeline.GeneratorResults` object. The filename for the generated pipeline file is stored in the property `GeneratedPipelineFiles`.

See Also

```
padv.pipeline.GitHubOptions | padv.pipeline.GitLabOptions |
padv.pipeline.JenkinsOptions
```

Topics

“How Pipeline Generation Works”
 “Integrate Process into GitHub”
 “Integrate Process into GitLab”

“Integrate Process into Jenkins”

padv.pipeline.GitHubOptions

Options for generating pipeline configuration file for GitHub

Description

Use the `padv.pipeline.GitHubOptions` object to represent the desired options for generating a GitHub pipeline configuration file. To generate a GitHub pipeline configuration file, use `padv.pipeline.GitHubOptions` as an input argument to the `padv.pipeline.generatePipeline` function. For information on how to use the pipeline generator to integrate into a GitHub CI system, see “Integrate Process into GitHub”.

Creation

Description

`options = padv.pipeline.GitHubOptions` returns configuration options for generating a GitHub pipeline configuration file.

`options = padv.pipeline.GitHubOptions(Name=Value)` sets properties using one or more name-value arguments. For example, `padv.pipeline.GitHubOptions(RunnerLabels = "Linux")` creates an options object that specifies that a generated pipeline configuration file use Linux as the GitHub Action runner label.

Properties

RunnerLabels — GitHub runner labels

"self-hosted" (default) | string

GitHub runner labels, specified as a string.

The labels determine which GitHub runner can execute the job. For more information, see <https://docs.github.com/en/actions/using-jobs/choosing-the-runner-for-a-job#targeting-runners-in-a-group>.

Example: `padv.pipeline.GitHubOptions(RunnerLabels = "Linux")`

Data Types: string

ArtifactZipFileName — Name of ZIP file for job artifacts

"padv_artifacts.zip" (default) | string

Name of ZIP file for job artifacts, specified as a string.

Example: `padv.pipeline.GitHubOptions(ArtifactZipFileName = "my_job_artifacts.zip")`

Data Types: string

RetentionDays — How many days GitHub stores workflow artifacts

"30" (default) | string

How many days GitHub stores workflow artifacts, specified as a string. This property corresponds to the job keyword "retention-days" in GitHub. After the specified number of retention days, the artifacts expire and GitHub deletes the artifacts.

Example: `padv.pipeline.GitHubOptions(RetentionDays = "90")`

Data Types: string

GeneratedYMLFileName — File name of generated GitLab pipeline file

"simulink_pipeline" (default) | string

File name of generated GitLab pipeline file, specified as a string.

By default, the generated pipeline generates into the subfolder **derived > pipeline**, relative to the project root. To change where the pipeline file generates, specify `GeneratedPipelineDirectory`.

Example: `padv.pipeline.GitHubOptions(GeneratedYMLFileName = "padv_generated_pipeline_file")`

Data Types: string

MatlabInstallationLocation — Path to MATLAB installation location

"PATH_TO_MATLAB" (default) | string

Path to MATLAB installation location, specified as a string.

Make sure the path that you specify uses the MATLAB root folder location and file separators for the operating system of your GitHub runner.

Example: "C:\Program Files\MATLAB\R2023a\bin"

Example: "/usr/local/MATLAB/R2023a/bin"

Example: "/Applications/MATLAB_R2023a.app/bin"

Data Types: string

EnableArtifactCollection — When to collect build artifacts

"always", 1, or true (default) | "never", 0, or false | "on_success" | "on_failure"

When to collect build artifacts, specified as:

- "never", 0, or false — Never collect artifacts
- "on_success" — Only collect artifacts when the pipeline succeeds
- "on_failure" — Only collect artifacts when the pipeline fails
- "always", 1, or true — Always collect artifacts

If the pipeline collects artifacts, the child pipeline contains a job, `Collect_Artifacts`, that compresses the build artifacts into a ZIP file and attaches the file to the job.

Example: `padv.pipeline.GitHubOptions(EnableArtifactCollection=false)`

Data Types: logical | string

ShellEnvironment — Shell environment GitHub uses to launch MATLAB

"bash" (default) | "pwsh"

Shell environment GitHub uses to launch MATLAB, specified as one of these values:

- "bash" — UNIX® shell script
- "pwsh" — PowerShell Core script

Example: `padv.pipeline.GitHubOptions(ShellEnvironment = "pwsh")`

Data Types: string

CheckoutSubmodules — Checkout Git™ submodules

"false" (default) | "true" | "recursive"

Checkout Git submodules at the beginning of each pipeline stage, specified as either:

- "false"
- "true"
- "recursive"

This property uses the GitHub Action `checkout@v3`. For information about the submodule input values, see <https://github.com/marketplace/actions/checkout-submodules>.

Example: `padv.pipeline.GitHubOptions(CheckoutSubmodules = "true")`

Data Types: string

RunprocessCommandOptions — Options for runprocess command

`padv.pipeline.RunProcessOptions` (default) | `padv.pipeline.RunProcessOptions` object

Options for `runprocess` command, specified as a `padv.pipeline.RunProcessOptions` object. `padv.pipeline.RunProcessOptions` has properties for the name-value arguments in the `runprocess` function, except for the arguments `Tasks`, `Process`, `Subprocesses`, and `FilterArtifact`.

For example, to have the pipeline generator use a command like `runprocess(DryRun = true)` in GitHub, you can create a `padv.pipeline.RunProcessOptions` object, specify the property values, and pass the object to `padv.pipeline.GitHubOptions`:

```
rpo = padv.pipeline.RunProcessOptions;
rpo.DryRun = true;
gho = padv.pipeline.GitHubOptions(RunprocessCommandOptions = rpo);
```

Example: `padv.pipeline.RunProcessOptions`

PipelineArchitecture — Number of stages and grouping of tasks in CI pipeline

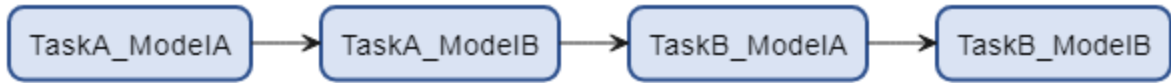
`padv.pipeline.Architecture.SingleStage` (default) |
`padv.pipeline.Architecture.SerialStages` |
`padv.pipeline.Architecture.SerialStagesGroupPerTask` |
`padv.pipeline.Architecture.IndependentModelPipelines`

Number of stages and grouping of tasks in CI pipeline, specified as either:

- `padv.pipeline.Architecture.SingleStage` — The pipeline has a single stage, named **Runprocess**, that runs each of the tasks in the process.

Runprocess

- `padv.pipeline.Architecture.SerialStages` — The pipeline has one stage for each task iteration in the process.



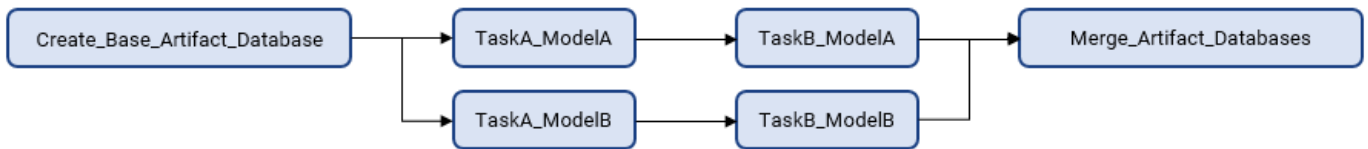
- `padv.pipeline.Architecture.SerialStagesGroupPerTask` — The pipeline has one stage for each task in the process.



- `padv.pipeline.Architecture.IndependentModelPipelines` — The pipeline contains parallel, downstream pipelines for each model. Each downstream pipeline independently runs the tasks associated with that model. The pipeline also has stages for creating and merging the artifact database files that store information about the artifacts in your project.

To make sure the jobs run in parallel, make sure that you either:

- Have multiple agents available
- Configure your agent to run multiple jobs concurrently



Example: `padv.pipeline.GitHubOptions(PipelineArchitecture = padv.pipeline.Architecture.SerialStages)`

MatlabLaunchCmd — Command to start MATLAB program

"matlab" (default) | string

Command to start MATLAB program, specified as a string.

Use this property to specify how the pipeline starts the MATLAB program. This property defines how the script in the generated pipeline file launches MATLAB.

Example: `padv.pipeline.GitHubOptions(MatlabLaunchCmd = "matlab")`

Data Types: string

MatlabStartupOptions — Command-line startup options for MATLAB

"-nodesktop -logfile output.log" (default) | string

Command-line startup options for MATLAB, specified as a string.

Use this property to specify the command-line startup options that the pipeline uses when starting the MATLAB program. This property defines the command-line startup options that appear next to the `-batch` option and `MatlabLaunchCmd` value in the "script" section of the generated pipeline file. The pipeline starts MATLAB with the specified startup options.

By default, the support package launches MATLAB using the `-batch` option. If you need to run MATLAB without the `-batch` option, specify the property `AddBatchStartupOption` as `false`.

Note If you run MATLAB using the `-nodisplay` option or you use a machine that does not have a display (like many CI runners and Docker® containers), you should set up a virtual display server before you include the following built-in tasks in your process model:

- **Generate SDD Report**
- **Generate Simulink Web View**
- **Generate Model Comparison**

For information, see “Set Up Virtual Display Machines Without Displays”.

Example: `padv.pipeline.GitHubOptions(MatlabStartupOptions = "-nodesktop -logfile mylogfile.log")`

Data Types: `string`

AddBatchStartupOption — Specify whether to open MATLAB using `-batch` startup option
`1` (true) (default) | `0` (false)

Specify whether to open MATLAB using `-batch` startup option, specified as a numeric or logical `0` (false) or `1` (true).

By default, the support package launches MATLAB in CI using the `-batch` startup option.

If you need to launch MATLAB with options that are not compatible with `-batch`, specify `AddBatchStartupOption` as `false`.

Example: `padv.pipeline.GitHubOptions(AddBatchStartupOption = false)`

Data Types: `logical`

GeneratedPipelineDirectory — Specify where the generated pipeline file generates
`fullfile("derived","pipeline")` (default) | `string`

Specify where the generated pipeline file generates, specified as a string.

This property defines the directory where the generated pipeline file generates.

By default, the generated pipeline file is named `"simulink_pipeline.yml"`. To change the name of the generated pipeline file, specify `GeneratedYMLFileName`.

Example: `padv.pipeline.GitHubOptions(GeneratedPipelineDirectory = fullfile("derived","pipeline","test"))`

Data Types: `string`

GenerateReport — Generate Process Advisor build report
`true` or `1` (default) | `false` or `0`

Generate Process Advisor build report, specified as a numeric or logical `1` (true) or `0` (false).

Example: `padv.pipeline.GitHubOptions(GenerateReport = false)`

Data Types: `logical`

ReportFormat — File format for generated report

"pdf" (default) | "html" | "html-file" | "docx"

File format for the generated report, specified as one of these values:

- "pdf" — PDF file
- "html" — HTML report, packaged as a zipped file that contains the HTML file, images, style sheet, and JavaScript files of the report
- "html-file" — HTML report
- "docx" — Microsoft Word document

Example: `padv.pipeline.GitHubOptions(ReportFormat = "html-file")`

ReportPath — Name and path of generated report

"ProcessAdvisorReport" (default) | string array

Name and path of generated report, specified as a string array.

By default, the report path uses a relative path to the project root and the pipeline generator generates a report `ProcessAdvisorReport.pdf`.

Example: `padv.pipeline.GitHubOptions(ReportPath = "myReport")`

Data Types: `string`

StopOnStageFailure — Stop running pipeline after stage fails

0 (false) (default) | 1 (true)

Stop running pipeline after stage fails, specified as a numeric or logical 0 (false) or 1 (true).

By default, the pipeline continues to run, even if a stage in the pipeline fails.

Example: `padv.pipeline.GitHubOptions(StopOnStageFailure = true)`

Data Types: `logical`

CheckOutdatedResultsAfterMerge — Check for outdated results after merge

1 (true) (default) | 0 (false)

Check for outdated results after merge, specified as a numeric or logical 1 (true) or 0 (false).

When specified as `true`, the pipeline checks if task results are still up-to-date after merging artifact database files from parallel jobs. Outdated results are not expected if the merge is successful. When there are outdated results, there could be an issue with the merge.

Example: `false`

Data Types: `logical`

Examples

Specify GitHub Configuration Options When Generating Pipeline Configuration File

Create a `padv.pipeline.GitHubOptions` object and change the options. When you generate a pipeline configuration file, the file uses the specified options.

This example shows how to use the pipeline generator API. For information on how to use the pipeline generator to integrate into a GitHub CI system, see “Integrate Process into GitHub”.

Load a project. For this example, you can load a Process Advisor example project. In the MATLAB Command Window, enter:

```
processAdvisorExampleStart
```

Specify your GitHub pipeline configuration options by creating a `padv.pipeline.GitHubOptions` object and modifying the object properties. For example, if you have a GitHub runner that uses a MATLAB installation at `/opt/matlab/r2023a`:

```
GitHubOptions = padv.pipeline.GitHubOptions  
GitHubOptions.MatlabInstallationLocation = "/opt/matlab/r2023a";
```

Generate a GitHub pipeline configuration file by using the function `padv.pipeline.generatePipeline` with the specified options.

```
padv.pipeline.generatePipeline(GitHubOptions);
```

Note Calling `padv.pipeline.generatePipeline(GitHubOptions)` is equivalent to calling `padv.pipeline.generateGitHubPipeline(GitHubOptions)`.

By default, the generated pipeline configuration file is named `simulink_pipeline.yml` and is located under the project root, in the subfolder **derived > pipeline**.

The `GeneratedYMLFileName` and `GeneratedPipelineDirectory` properties of the `padv.pipeline.GitHubOptions` object control the name and location of the generated pipeline configuration file.

For information on how to use the pipeline generator to integrate into a GitHub CI system, see “Integrate Process into GitHub”.

See Also

`padv.pipeline.generatePipeline`

Topics

“Approaches to Running Processes in CI”

“How Pipeline Generation Works”

“Integrate Process into GitHub”

“Tips for Setting Up CI Agents”

padv.pipeline.GitLabOptions

Options for generating pipeline configuration file for GitLab

Description

This object requires CI/CD Automation for Simulink Check. Use the `padv.pipeline.GitLabOptions` object to represent the desired options for generating a GitLab pipeline configuration file. To generate a GitLab pipeline configuration file, use `padv.pipeline.GitLabOptions` as an input argument to the `padv.pipeline.generatePipeline` function. For information on how to use the pipeline generator to integrate into a GitLab CI system, see “Integrate Process into GitLab”.

Creation

Syntax

```
options = padv.pipeline.GitLabOptions
options = padv.pipeline.GitLabOptions(Name=Value)
```

Description

`options = padv.pipeline.GitLabOptions` returns configuration options for generating a GitLab pipeline configuration file.

`options = padv.pipeline.GitLabOptions(Name=Value)` sets properties using one or more name-value arguments. For example, `padv.pipeline.GitLabOptions(Tags="high_memory")` creates an options object that specifies that a generated pipeline configuration file use `high_memory` as the GitLab CI/CD tag.

Properties

Tags — GitLab CI/CD tags

string | string array

GitLab CI/CD tags, specified as a string or string array. Use this property to specify the tags that appear next to the `tags` keyword in a generated GitLab pipeline configuration file.

The GitLab CI/CD tags select a GitLab Runner for a job. The property `Tags` specifies which CI/CD tags appear next to the `tags` keyword in a generated pipeline configuration file.

For more information on the `tags` keyword, see <https://docs.gitlab.com/ee/ci/yaml/#tags>.

Example: `options = padv.pipeline.GitLabOptions(Tags="high_memory")`

Data Types: string

EnableArtifactCollection — When to collect build artifacts

"always", 1, or true (default) | "never", 0, or false | "on_success" | "on_failure"

When to collect build artifacts, specified as:

- "never", 0, or false — Never collect artifacts
- "on_success" — Only collect artifacts when the pipeline succeeds
- "on_failure" — Only collect artifacts when the pipeline fails
- "always", 1, or true — Always collect artifacts

If the pipeline collects artifacts, the child pipeline contains a job, `Collect_Artifacts`, that compresses the build artifacts into a ZIP file and attaches the file to the job.

This property creates an "artifacts" section in the generated pipeline file. For more information, see the GitLab documentation: <https://docs.gitlab.com/ee/ci/yaml/#artifacts>.

Example: `padv.pipeline.GitLabOptions(EnableArtifactCollection="on_failure")`

Data Types: logical | string

ArtifactZipFileName — Name of ZIP file for job artifacts

"padv_artifacts.zip" (default) | string

Name of ZIP file for job artifacts, specified as a string.

This property specifies the file name that appears next to the "name" keyword in the generated pipeline file. For more information, see the GitLab documentation for "artifacts:name": <https://docs.gitlab.com/ee/ci/yaml/#artifactsname>.

Example: `padv.pipeline.GitLabOptions(ArtifactZipFileName = "my_job_artifacts.zip")`

Data Types: string

ArtifactsExpireIn — How long GitLab stores job artifacts before the artifacts expire

"30 days" (default) |

How long GitLab stores job artifacts before the artifacts expire, specified as a string.

Use this property to specify how long GitLab stores job artifacts before the artifacts expire and GitLab deletes the artifacts. This property specifies the expiry time that appears next to the "expire_in" keyword in the generated pipeline file. For a list of valid possible inputs, see the GitLab documentation for "artifacts:expire_in": https://docs.gitlab.com/ee/ci/yaml/#artifactsexpire_in.

Example: `padv.pipeline.GitLabOptions(ArtifactsExpireIn = "60 days")`

Data Types: string

ArtifactsWhen — When GitLab uploads job artifacts

"always" (default) | "on_success" | "on_failure"

Warning This property will be removed in a future release. Use the property `EnableArtifactCollection` instead.

When GitLab uploads job artifacts, specified as either:

- "on_success"
- "on_failure"
- "always"

Use this property to specify when GitLab uploads job artifacts. This property specifies the input that appears next to the "when" keyword in the generated pipeline file. For more information, see the GitLab documentation for "artifacts:when": <https://docs.gitlab.com/ee/ci/yaml/#artifactswhen>.

Example: `padv.pipeline.GitLabOptions(ArtifactsWhen = "on_success")`

GeneratedYMLFileName — File name of generated GitLab pipeline file

"simulink_pipeline" (default) | string

File name of generated GitLab pipeline file, specified as a string.

By default, the generated pipeline generates into the subfolder **derived > pipeline**, relative to the project root. To change where the pipeline file generates, specify `GeneratedPipelineDirectory`.

Example: `padv.pipeline.GitLabOptions(GeneratedYMLFileName = "padv_generated_pipeline_file")`

Data Types: string

RunprocessCommandOptions — Options for runprocess command

`padv.pipeline.RunProcessOptions` (default) | `padv.pipeline.RunProcessOptions` object

Options for runprocess command, specified as a `padv.pipeline.RunProcessOptions` object. `padv.pipeline.RunProcessOptions` has properties for the name-value arguments in the runprocess function, except for the arguments `Tasks`, `Process`, `Subprocesses`, and `FilterArtifact`.

For example, to have the pipeline generator use a command like `runprocess(DryRun = true)` in GitLab, you can create a `padv.pipeline.RunProcessOptions` object, specify the property values, and pass the object to `padv.pipeline.GitLabOptions`:

```
rpo = padv.pipeline.RunProcessOptions;
rpo.DryRun = true;
glo = padv.pipeline.GitLabOptions(RunprocessCommandOptions = rpo);
```

Example: `padv.pipeline.GitLabOptions(RunprocessCommandOptions = padv.pipeline.RunProcessOptions)`

PipelineArchitecture — Number of stages and grouping of tasks in CI pipeline

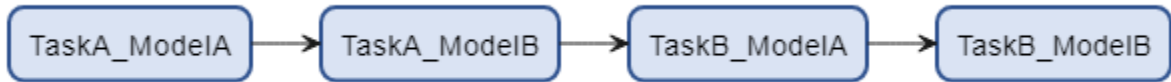
`padv.pipeline.Architecture.SingleStage` (default) |
`padv.pipeline.Architecture.SerialStages` |
`padv.pipeline.Architecture.SerialStagesGroupPerTask` |
`padv.pipeline.Architecture.IndependentModelPipelines`

Number of stages and grouping of tasks in CI pipeline, specified as either:

- `padv.pipeline.Architecture.SingleStage` — The pipeline has a single stage, named **Runprocess**, runs each of the tasks in the process.

Runprocess

- `padv.pipeline.Architecture.SerialStages` — The pipeline has one stage for each task iteration in the process.



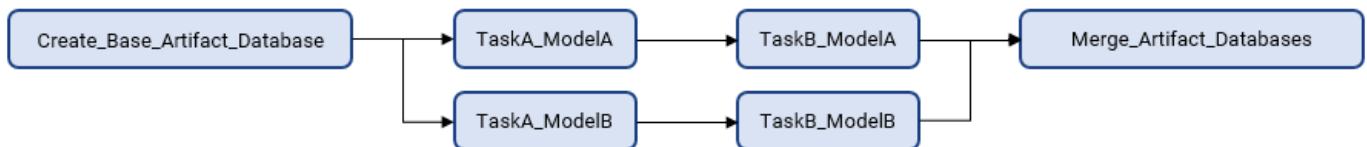
- `padv.pipeline.Architecture.SerialStagesGroupPerTask` — The pipeline has one stage for each task in the process.



- `padv.pipeline.Architecture.IndependentModelPipelines`— The pipeline contains parallel, downstream pipelines for each model. Each downstream pipeline independently runs the tasks associated with that model. The pipeline also has stages for creating and merging the artifact database files that store information about the artifacts in your project.

To make sure the jobs run in parallel, make sure that you either:

- Have multiple agents available
- Configure your agent to run multiple jobs concurrently



For more information on pipeline architectures, see “Integrate Process into GitLab”.

Example: `padv.pipeline.GitLabOptions(PipelineArchitecture = padv.pipeline.Architecture.SerialStages)`

ForceRunAllTasks — Pipeline runs both up to date and outdated tasks

0 (false) (default) | 1 (true)

Pipeline runs both up to date and outdated tasks, specified as a numeric or logical 1 (true) or 0 (false).

The property defines the Force argument for the `runprocess` function in the generated pipeline file.

Example: `padv.pipeline.GitLabOptions(ForceRunAllTasks=true)`

Data Types: logical

ExitInBatchMode — Exits MATLAB if MATLAB was run with the -batch startup option

1 (true) (default) | 0 (false)

Exits MATLAB if MATLAB was run with the `-batch` startup option, specified as a numeric or logical 0 (false) or 1 (true).

This property defines the `ExitInBatchMode` argument for the `runprocess` function in the generated pipeline file.

Example: `padv.pipeline.GitLabOptions(ExitInBatchMode=false)`

Data Types: `logical`

RerunFailedTasks — Treats tasks which previously failed as being outdated

`0 (false) (default) | 1 (true)`

Treats tasks which previously failed as being outdated, specified as a numeric or logical `1 (true)` or `0 (false)`.

This property defines the `RerunFailedTasks` argument for the `runprocess` function in the generated pipeline file.

Example: `padv.pipeline.GitLabOptions(RerunFailedTasks=true)`

Data Types: `logical`

RerunErroredTasks — Treats tasks which previously generated errors as outdated

`0 (false) (default) | 1 (true)`

Treats tasks which previously generated errors as outdated, specified as a numeric or logical `1 (true)` or `0 (false)`.

This property defines the `RerunErroredTasks` argument for the `runprocess` function in the generated pipeline file.

Example: `padv.pipeline.GitLabOptions(RerunErroredTasks=true)`

Data Types: `logical`

MatlabLaunchCmd — Command to start MATLAB program

`"matlab" (default) | string`

Command to start MATLAB program, specified as a string.

Use this property to specify how the pipeline starts the MATLAB program. This property defines how the script in the generated pipeline file launches MATLAB.

Example: `padv.pipeline.GitLabOptions(MatlabLaunchCmd = "matlab")`

Data Types: `string`

MatlabStartupOptions — Command-line startup options for MATLAB

`"-nodesktop -logfile output.log" (default) | string`

Command-line startup options for MATLAB, specified as a string.

Use this property to specify the command-line startup options that the pipeline uses when starting the MATLAB program. This property defines the command-line startup options that appear next to the `-batch` option and `MatlabLaunchCmd` value in the `"script"` section of the generated pipeline file. The pipeline starts MATLAB with the specified startup options.

By default, the support package launches MATLAB using the `-batch` option. If you need to run MATLAB without the `-batch` option, specify the property `AddBatchStartupOption` as `false`.

Note If you run MATLAB using the `-nodisplay` option, you should set up a virtual display server before you include the following built-in tasks in your process model:

- **Generate SDD Report**
- **Generate Simulink Web View**
- **Generate Model Comparison**

For information, see “Set Up Virtual Display Machines Without Displays”.

Example: `padv.pipeline.GitLabOptions(MatlabStartupOptions = "-nodesktop -logfile mylogfile.log")`

Data Types: string

AddBatchStartupOption — Specify whether to open MATLAB using -batch startup option

1 (true) (default) | 0 (false)

Specify whether to open MATLAB using `-batch` startup option, specified as a numeric or logical 0 (false) or 1 (true).

By default, the support package launches MATLAB in CI using the `-batch` startup option.

If you need to launch MATLAB with options that are not compatible with `-batch`, specify `AddBatchStartupOption` as `false`.

Example: `padv.pipeline.GitLabOptions(AddBatchStartupOption = false)`

Data Types: logical

GeneratedPipelineDirectory — Specify where the generated pipeline file generates

`fullfile("derived", "pipeline")` (default) | string

Specify where the generated pipeline file generates, specified as a string.

This property defines the directory where the generated pipeline file generates.

By default, the generated pipeline file is named `"simulink_pipeline.yml"`. To change the name of the generated pipeline file, specify `GeneratedYMLFileName`.

Example: `padv.pipeline.GitLabOptions(GeneratedPipelineDirectory = fullfile("derived", "pipeline", "test"))`

Data Types: string

GenerateJUnitForProcess — Generate JUnit-style XML reports for process

true or 1 (default) | false or 0

Generate JUnit-style XML reports for each task in the process, specified as a numeric or logical 1 (true) or 0 (false).

JUnit reports allow you see which tests failed in CI without having to examine the job logs.

If you generate JUnit reports, GitLab shows test failures directly in the merge request and pipeline detail view. For more information on how GitLab displays JUnit results, see the GitLab documentation: https://docs.gitlab.com/ee/ci/testing/unit_test_reports.html#view-unit-test-reports-on-gitlab.

Example: `padv.pipeline.GitLabOptions(GenerateJUnitForProcess = false)`

Data Types: `logical`

GenerateReport — Generate Process Advisor build report

`true` or `1` (default) | `false` or `0`

Generate Process Advisor build report, specified as a numeric or logical `1` (`true`) or `0` (`false`).

Example: `padv.pipeline.GitLabOptions(GenerateReport = false)`

Data Types: `logical`

ReportFormat — File format for generated report

`"pdf"` (default) | `"html"` | `"html-file"` | `"docx"`

File format for the generated report, specified as one of these values:

- `"pdf"` — PDF file
- `"html"` — HTML report, packaged as a zipped file that contains the HTML file, images, style sheet, and JavaScript files of the report
- `"html-file"` — HTML report
- `"docx"` — Microsoft Word document

Example: `padv.pipeline.GitLabOptions(ReportFormat = "html-file")`

ReportPath — Name and path of generated report

`"$PROJECTROOT$/ProcessAdvisorReport"` (default) | string array

Name and path of generated report, specified as a string array.

By default, the report path uses a relative path to the project root and the pipeline generator generates a report `ProcessAdvisorReport.pdf`.

Example: `padv.pipeline.GitLabOptions(ReportPath = "myReport")`

Data Types: `string`

StopOnStageFailure — Stop running pipeline after stage fails

`0` (`false`) (default) | `1` (`true`)

Stop running pipeline after stage fails, specified as a numeric or logical `0` (`false`) or `1` (`true`).

By default, the pipeline continues to run, even if a stage in the pipeline fails.

Example: `padv.pipeline.GitLabOptions(StopOnStageFailure = true)`

Data Types: `logical`

CheckOutdatedResultsAfterMerge — Check for outdated results after merge

`1` (`true`) (default) | `0` (`false`)

Check for outdated results after merge, specified as a numeric or logical `1` (`true`) or `0` (`false`).

When specified as `true`, the pipeline checks if task results are still up-to-date after merging artifact database files from parallel jobs. Outdated results are not expected if the merge is successful. When there are outdated results, there could be an issue with the merge.

Example: `false`

Data Types: `logical`

Examples

Specify GitLab Configuration Options When Generating Pipeline Configuration File

Create a `padv.pipeline.GitLabOptions` object and change the options. When you generate a pipeline configuration file, the file uses the specified options.

This example shows how to use the pipeline generator API. For information on how to use the pipeline generator to integrate into a GitLab CI system, see “Integrate Process into GitLab”.

Load a project. For this example, you can load a Process Advisor example project. In the MATLAB Command Window, enter:

```
processAdvisorExampleStart
```

Create a `padv.pipeline.GitLabOptions` object for generating a GitLab pipeline configuration file. Specify a GitLab CI/CD tag of `high_memory`, specify that the function `runprocess` should not automatically exit MATLAB after the pipeline finishes running, and a single stage pipeline architecture.

```
GitLabOptions = padv.pipeline.GitLabOptions(...
Tags = "high_memory",...
ExitInBatchMode = 0,...
PipelineArchitecture = padv.pipeline.Architecture.SingleStage);
```

Generate a GitLab pipeline configuration file by using the function `padv.pipeline.generatePipeline` with the specified options.

```
padv.pipeline.generatePipeline(GitLabOptions);
```

Note Calling `padv.pipeline.generatePipeline(GitLabOptions)` is equivalent to calling `padv.pipeline.generateGitLabPipeline(GitLabOptions)`.

By default, the generated pipeline file is named `simulink_pipeline.yml` and is saved in the **derived > pipeline** folder, relative to the project root. To change the name of the generated pipeline file, specify the argument `GeneratedYMLFileName` for `padv.pipeline.GitLabOptions`. To change where the pipeline file generates, specify the argument `GeneratedPipelineDirectory`.

For information on how to use the pipeline generator to integrate into a GitLab CI system, see “Integrate Process into GitLab”.

See Also

`padv.pipeline.generatePipeline`

Topics

“Approaches to Running Processes in CI”

“How Pipeline Generation Works”

“Integrate Process into GitLab”

“Tips for Setting Up CI Agents”

padv.pipeline.JenkinsOptions

Options for generating pipeline configuration file for Jenkins

Description

This object requires CI/CD Automation for Simulink Check. Use the `padv.pipeline.JenkinsOptions` object to represent the desired options for generating a Jenkins pipeline configuration file. To generate a Jenkins pipeline configuration file, use `padv.pipeline.JenkinsOptions` as an input argument to the `padv.pipeline.generatePipeline` function. For information on how to use the pipeline generator to integrate into a Jenkins CI system, see "Integrate Process into Jenkins".

Note To use the pipeline generator for Jenkins, you must have at least 3 executors configured and available on your Jenkins instance.

Creation

Syntax

```
options = padv.pipeline.JenkinsOptions
options = padv.pipeline.JenkinsOptions(Name=Value)
```

Description

`options = padv.pipeline.JenkinsOptions` returns configuration options for generating a Jenkins pipeline configuration file.

`options = padv.pipeline.JenkinsOptions(Name=Value)` sets properties using one or more name-value arguments. For example, `padv.pipeline.JenkinsOptions(AgentLabel = "high_memory")` creates an object that specifies that a generated pipeline configuration file use an agent with the label `high_memory`.

Properties

AgentLabel — Which Jenkins agent executes pipeline tasks in Jenkins environment

"any" (default) | string | string array

Which Jenkins agent executes pipeline tasks in the Jenkins environment, specified as a string or string array.

Use this property to specify the Jenkins agent that executes all stages in the pipeline. Jenkins agents are typically either a machine or a container. For more information, see the "Glossary" in the Jenkins documentation: <https://www.jenkins.io/doc/book/glossary/#agent>.

Example: `options = padv.pipeline.JenkinsOptions(AgentLabel="high_memory")`

Data Types: string

EnableArtifactCollection — When to collect build artifacts

"always", 1, or true (default) | "never", 0, or false | "on_success" | "on_failure"

When to collect build artifacts, specified as:

- "never", 0, or false — Never collect artifacts
- "on_success" — Only collect artifacts when the pipeline succeeds
- "on_failure" — Only collect artifacts when the pipeline fails
- "always", 1, or true — Always collect artifacts

If you choose to collect artifacts, the child pipeline contains a job, `Collect_Artifacts`, that collects the build artifacts and attaches the artifacts to the `Collect_Artifacts` job.

This property uses the Jenkins Core Plugin to add an "archiveArtifacts" step in the generated Jenkinsfile that defines the Jenkins pipeline. Install the Jenkins Core Plugin before you specify `EnableArtifactCollection`. For more information, see the Jenkins documentation for "archiveArtifacts": <https://www.jenkins.io/doc/pipeline/steps/core/#archiveartifacts-archive-the-artifacts>.

Example: `padv.pipeline.JenkinsOptions(EnableArtifactCollection="on_failure")`

Data Types: `logical` | `string`

ArtifactZipFileName — Name of ZIP file for job artifacts

"padv_artifacts.zip" (default) | string

Name of ZIP file for job artifacts, specified as a string.

This property specifies the file name that appears next to the "artifacts" for the "archiveArtifacts" step in the generated Jenkinsfile that defines the Jenkins pipeline.

For more information, see the Jenkins documentation for "archiveArtifacts": <https://www.jenkins.io/doc/pipeline/steps/core/#archiveartifacts-archive-the-artifacts>.

Example: `padv.pipeline.JenkinsOptions(ArtifactZipFileName = "my_job_artifacts.zip")`

Data Types: `string`

SaveArtifactsOnSuccess — Setting to only archive artifacts for successful builds

1 (true) (default) | 0 (false)

Warning This property will be removed in a future release. Use the property `EnableArtifactCollection` instead.

Setting to only archive artifacts for successful builds, specified as a numeric or logical 0 (false) or 1 (true).

Use this property to specify whether Jenkins only saves build artifacts for successful builds. This property corresponds to the argument "onlyIfSuccessful" for the "artifacts" in the "archiveArtifacts" step in the Jenkinsfile that defines the pipeline.

For more information, see the Jenkins documentation for "archiveArtifacts": <https://www.jenkins.io/doc/pipeline/steps/core/#archiveartifacts-archive-the-artifacts>.

Example: `padv.pipeline.JenkinsOptions(SaveArtifactsOnSuccess = false)`

Data Types: `logical`

GeneratedJenkinsFileName — File name of generated Jenkins pipeline file

`"simulink_pipeline"` (default) | `string`

File name of generated Jenkins pipeline file, specified as a string.

By default, the generated pipeline generates into the subfolder **derived > pipeline**, relative to the project root. To change where the pipeline file generates, specify `GeneratedPipelineDirectory`.

Example: `padv.pipeline.JenkinsOptions(GeneratedJenkinsFileName = "padv_generated_pipeline_file")`

Data Types: `string`

UseMatlabPlugin — Specify whether Jenkins uses MATLAB Plugin to launch MATLAB

`1` (true) (default) | `0` (false)

Specify whether Jenkins uses MATLAB Plugin to launch MATLAB, specified as a numeric or logical `0` (false) or `1` (true).

If the property `UseMatlabPlugin` is `true`, Jenkins uses the `"runMATLABCommand"` step to launch MATLAB and the pipeline generator ignores the properties `MatlabLaunchCmd` and `MatlabStartupOptions`. For more information, see the Jenkins documentation for `"runMATLABCommand"`: <https://www.jenkins.io/doc/pipeline/steps/matlab/#runmatlabcommand-run-matlab-commands-scripts-or-functions>

If the property `UseMatlabPlugin` is `false`, Jenkins uses the specified `ShellEnvironment` to launch MATLAB and uses the options specified by the properties `MatlabLaunchCmd` and `MatlabStartupOptions`.

Using the MATLAB Plugin for Jenkins is recommended. For more information, see <https://plugins.jenkins.io/matlab/>.

Example: `padv.pipeline.JenkinsOptions(UseMatlabPlugin = false)`

Data Types: `logical`

ShellEnvironment — Shell environment Jenkins uses to launch MATLAB

`""` (default) | `string`

Shell environment Jenkins uses to launch MATLAB, specified as one of these values:

- `"bat"` — Windows® batch script
- `"sh"` — Shell script
- `"pwsh"` — PowerShell Core script
- `"powershell"` — Windows PowerShell script
- `""` — Automatically use `"bat"` or `"sh"` based on the platform where pipeline generation runs

If the property `UseMatlabPlugin` is `true`, Jenkins uses the `"runMATLABCommand"` step to launch MATLAB and the pipeline generator ignores the properties `MatlabLaunchCmd` and `MatlabStartupOptions`. For more information, see the Jenkins documentation for `"runMATLABCommand"`: <https://www.jenkins.io/doc/pipeline/steps/matlab/#runmatlabcommand-run-matlab-commands-scripts-or-functions>

If the property `UseMatlabPlugin` is `false`, Jenkins uses the specified `ShellEnvironment` to launch MATLAB and uses the options specified by the properties `MatlabLaunchCmd` and `MatlabStartupOptions`.

```
Example: padv.pipeline.JenkinsOptions(UseMatlabPlugin = false, ShellEnvironment = "bat")
```

Data Types: `string`

RunprocessCommandOptions — Options for runprocess command

```
padv.pipeline.RunProcessOptions (default) | padv.pipeline.RunProcessOptions object
```

Options for `runprocess` command, specified as a `padv.pipeline.RunProcessOptions` object. `padv.pipeline.RunProcessOptions` has properties for the name-value arguments in the `runprocess` function, except for the arguments `Tasks`, `Process`, `Subprocesses`, and `FilterArtifact`.

For example, to have the pipeline generator use a command like `runprocess(DryRun = true)` in Jenkins, you can create a `padv.pipeline.RunProcessOptions` object, specify the property values, and pass the object to `padv.pipeline.GitLabOptions`:

```
rpo = padv.pipeline.RunProcessOptions;
rpo.DryRun = true;
jo = padv.pipeline.JenkinsOptions(RunprocessCommandOptions = rpo);
```

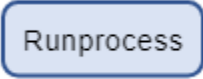
```
Example: padv.pipeline.JenkinsOptions(RunprocessCommandOptions =
padv.pipeline.RunProcessOptions)
```

PipelineArchitecture — Number of stages and grouping of tasks in CI pipeline

```
padv.pipeline.Architecture.SingleStage (default) |
padv.pipeline.Architecture.SerialStages |
padv.pipeline.Architecture.SerialStagesGroupPerTask |
padv.pipeline.Architecture.IndependentModelPipelines
```

Number of stages and grouping of tasks in CI pipeline, specified as either:

- `padv.pipeline.Architecture.SingleStage` — The pipeline has a single stage, named **Runprocess**, that runs each of the tasks in the process. This pipeline architecture requires at least 3 Jenkins executors. For information on defining Jenkins executors, see [Managing Nodes](#).



```
graph TD
    Runprocess[Runprocess]
```

- `padv.pipeline.Architecture.SerialStages` — The pipeline has one stage for each task iteration in the process. This pipeline architecture requires at least 3 Jenkins executors. For information on defining Jenkins executors, see [Managing Nodes](#).



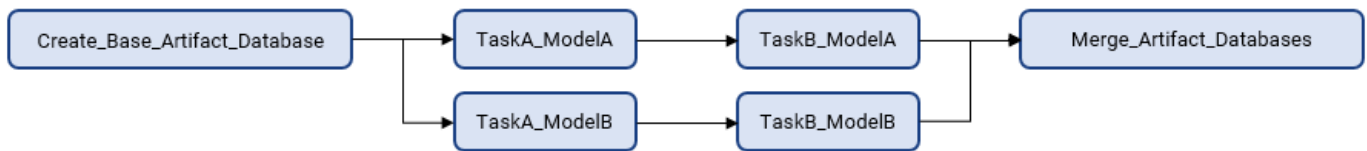
- `padv.pipeline.Architecture.SerialStagesGroupPerTask` — The pipeline has one stage for each task in the process. This pipeline architecture requires at least 3 Jenkins executors. For information on defining Jenkins executors, see [Managing Nodes](#).



- `padv.pipeline.Architecture.IndependentModelPipelines`— The pipeline contains parallel, downstream pipelines for each model. Each downstream pipeline independently runs the tasks associated with that model. The pipeline also has stages for creating and merging the artifact database files that store information about the artifacts in your project. This pipeline architecture requires at least 3 executors, plus an additional executor for each model in your project.

To make sure the jobs run in parallel, make sure that you either:

- Have multiple agents available
- Configure your agent to run multiple jobs concurrently



For more information on pipeline architectures, see “Integrate Process into Jenkins”.

Example: `padv.pipeline.JenkinsOptions(PipelineArchitecture = padv.pipeline.Architecture.SerialStages)`

ForceRunAllTasks — Pipeline runs both up to date and outdated tasks

0 (false) (default) | 1 (true)

Pipeline runs both up to date and outdated tasks, specified as a numeric or logical 1 (true) or 0 (false).

The property defines the Force argument for the `runprocess` function in the generated pipeline file.

Example: `padv.pipeline.JenkinsOptions(ForceRunAllTasks=true)`

Data Types: logical

ExitInBatchMode — Exits MATLAB if MATLAB was run with the -batch startup option

1 (true) (default) | 0 (false)

Exits MATLAB if MATLAB was run with the `-batch` startup option, specified as a numeric or logical 0 (false) or 1 (true).

This property defines the `ExitInBatchMode` argument for the `runprocess` function in the generated pipeline file.

Example: `padv.pipeline.JenkinsOptions(ExitInBatchMode=false)`

Data Types: logical

RerunFailedTasks — Treats tasks which previously failed as being outdated

0 (false) (default) | 1 (true)

Treats tasks which previously failed as being outdated, specified as a numeric or logical 1 (true) or 0 (false).

This property defines the `RerunFailedTasks` argument for the `runprocess` function in the generated pipeline file.

Example: `padv.pipeline.JenkinsOptions(RerunFailedTasks=true)`

Data Types: `logical`

RerunErroredTasks — Treats tasks which previously generated errors as outdated

0 (false) (default) | 1 (true)

Treats tasks which previously generated errors as outdated, specified as a numeric or logical 1 (true) or 0 (false).

This property defines the `RerunErroredTasks` argument for the `runprocess` function in the generated pipeline file.

Example: `padv.pipeline.JenkinsOptions(RerunErroredTasks=true)`

Data Types: `logical`

MatlabLaunchCmd — Command to start MATLAB program

"matlab" (default) | string

Command to start MATLAB program, specified as a string.

Use this property to specify how the pipeline starts the MATLAB program. This property defines how the generated pipeline file launches MATLAB.

Example: `padv.pipeline.JenkinsOptions(MatlabLaunchCmd = "matlab")`

Data Types: `string`

MatlabStartupOptions — Command-line startup options for MATLAB

"-nodesktop -logfile output.log" (default) | string

Command-line startup options for MATLAB, specified as a string.

Use this property to specify the command-line startup options that the pipeline uses when starting the MATLAB program. This property defines the command-line startup options that appear next to the `-batch` option and `MatlabLaunchCmd` value in the `script` section of the generated pipeline file. The pipeline starts MATLAB with the specified startup options.

By default, the support package launches MATLAB using the `-batch` option. If you need to run MATLAB without the `-batch` option, specify the property `AddBatchStartupOption` as `false`.

Note If you run MATLAB using the `-nodisplay` option, you should set up a virtual display server before you include the following built-in tasks in your process model:

- **Generate SDD Report**
- **Generate Simulink Web View**
- **Generate Model Comparison**

For information, see "Set Up Virtual Display Machines Without Displays".

Example: `padv.pipeline.JenkinsOptions(MatlabStartupOptions = "-nodesktop -logfile mylogfile.log")`

Data Types: `string`

AddBatchStartupOption — Specify whether to open MATLAB using -batch startup option
`1 (true) (default) | 0 (false)`

Specify whether to open MATLAB using -batch startup option, specified as a numeric or logical `0` (false) or `1` (true).

By default, the support package launches MATLAB in CI using the -batch startup option.

If you need to launch MATLAB with options that are not compatible with -batch, specify `AddBatchStartupOption` as `false`.

Example: `padv.pipeline.JenkinsOptions(AddBatchStartupOption = false)`

Data Types: `logical`

GeneratedPipelineDirectory — Specify where the generated pipeline file generates
`fullfile("derived","pipeline") (default) | string`

Specify where the generated pipeline file generates, specified as a string.

This property defines the directory where the generated pipeline file generates.

By default, the generated pipeline file is named "simulink_pipeline". To change the name of the generated pipeline file, specify `GeneratedJenkinsFileName`.

Example: `padv.pipeline.JenkinsOptions(GeneratedPipelineDirectory = fullfile("derived","pipeline","test"))`

Data Types: `string`

GenerateJUnitForProcess — Generate JUnit-style XML reports for process
`true or 1 (default) | false or 0`

Generate JUnit-style XML reports for each task in the process, specified as a numeric or logical `1` (true) or `0` (false).

JUnit reports allow you see which tests failed in CI without having to examine the job logs.

If you generate JUnit reports, Jenkins can show test failures and trends directly in the user interface. For more information on how Jenkins displays JUnit results, see the Jenkins documentation: <https://plugins.jenkins.io/junit/>.

Note You must have the JUnit plugin installed on your Jenkins controller to see JUnit results. For information, see <https://plugins.jenkins.io/junit/>.

Example: `padv.pipeline.JenkinsOptions(GenerateJUnitForProcess = false)`

Data Types: `logical`

GenerateReport — Generate Process Advisor build report
`true or 1 (default) | false or 0`

Generate Process Advisor build report, specified as a numeric or logical `1 (true)` or `0 (false)`.

Example: `padv.pipeline.JenkinsOptions(GenerateReport = false)`

Data Types: `logical`

ReportFormat — File format for generated report

`"pdf"` (default) | `"html"` | `"html-file"` | `"docx"`

File format for the generated report, specified as one of these values:

- `"pdf"` — PDF file
- `"html"` — HTML report, packaged as a zipped file that contains the HTML file, images, style sheet, and JavaScript files of the report
- `"html-file"` — HTML report
- `"docx"` — Microsoft Word document

Example: `padv.pipeline.JenkinsOptions(ReportFormat = "html-file")`

ReportPath — Name and path of generated report

`"$PROJECTROOT$/ProcessAdvisorReport"` (default) | string array

Name and path of generated report, specified as a string array.

By default, the report path uses a relative path to the project root and the pipeline generator generates a report `ProcessAdvisorReport.pdf`.

Example: `padv.pipeline.JenkinsOptions(ReportFormat = "myReport")`

Data Types: `string`

StopOnStageFailure — Stop running pipeline after stage fails

`0 (false)` (default) | `1 (true)`

Stop running pipeline after stage fails, specified as a numeric or logical `0 (false)` or `1 (true)`.

By default, the pipeline continues to run, even if a stage in the pipeline fails.

Example: `padv.pipeline.JenkinsOptions(StopOnStageFailure = true)`

Data Types: `logical`

CheckOutdatedResultsAfterMerge — Check for outdated results after merge

`1 (true)` (default) | `0 (false)`

Check for outdated results after merge, specified as a numeric or logical `1 (true)` or `0 (false)`.

When specified as `true`, the pipeline checks if task results are still up-to-date after merging artifact database files from parallel jobs. Outdated results are not expected if the merge is successful. When there are outdated results, there could be an issue with the merge.

Example: `false`

Data Types: `logical`

Examples

Specify Jenkins Configuration Options When Generating Pipeline Configuration File

Create a `padv.pipeline.JenkinsOptions` object and change the options. When you generate a pipeline configuration file, the file uses the specified options.

This example shows how to use the pipeline generator API. For information on how to use the pipeline generator to integrate into a Jenkins CI system, see “Integrate Process into Jenkins”.

Load a project. For this example, you can load a Process Advisor example project. In the MATLAB Command Window, enter:

```
processAdvisorExampleStart
```

Create a `padv.pipeline.JenkinsOptions` object for generating a Jenkins pipeline configuration file. Specify a Jenkins agent label of `high_memory`, specify that the function `runprocess` should not automatically exit MATLAB after the pipeline finishes running, and a single stage pipeline architecture.

```
JenkinsOptions = padv.pipeline.JenkinsOptions(...
AgentLabel = "high_memory",...
ExitInBatchMode = 0,...
PipelineArchitecture = padv.pipeline.Architecture.SingleStage);
```

Generate a Jenkins pipeline configuration file by using the function `padv.pipeline.generatePipeline` with the specified options.

```
padv.pipeline.generatePipeline(JenkinsOptions);
```

Note Calling `padv.pipeline.generatePipeline(JenkinsOptions)` is equivalent to calling `padv.pipeline.generateJenkinsPipeline(JenkinsOptions)`.

By default, the generated pipeline file is named `simulink_pipeline` and is saved in the **derived > pipeline** folder, relative to the project root. To change the name of the generated pipeline file, specify the argument `GeneratedJenkinsFileName` for `padv.pipeline.JenkinsOptions`. To change where the pipeline file generates, specify the argument `GeneratedPipelineDirectory`.

For information on how to use the pipeline generator to integrate into a Jenkins CI system, see “Integrate Process into Jenkins”.

See Also

`padv.pipeline.generatePipeline`

Topics

“Approaches to Running Processes in CI”

“How Pipeline Generation Works”

“Integrate Process into Jenkins”

“Tips for Setting Up CI Agents”

Report Generator API

padv.ProcessAdvisorReportGenerator Class

Namespace: padv

Superclasses: mlreportgen.report.Report

Settings for generating Process Advisor report

Description

This class requires CI/CD Automation for Simulink Check. Use the `padv.ProcessAdvisorReportGenerator` class to represent the settings for generating a Process Advisor report. After you run tasks using the Process Advisor app or `runprocess` function, you can call the `generateReport` function on a `padv.ProcessAdvisorReportGenerator` object to generate a report of the task results.

The `padv.ProcessAdvisorReportGenerator` class is a handle class.

Creation

Syntax

```
padv.ProcessAdvisorReportGenerator()  
padv.ProcessAdvisorReportGenerator(Name=Value)
```

Description

`padv.ProcessAdvisorReportGenerator()` returns settings for generating a Process Advisor report.

`padv.ProcessAdvisorReportGenerator(Name=Value)` sets properties using one or more name-value arguments. For example, `padv.ProcessAdvisorReportGenerator(Format="html-file")` creates a report settings object that specifies for the generated Process Advisor report to be an HTML file.

Input Arguments

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `padv.ProcessAdvisorReportGenerator(Format="html-file")`

Format — File format for generated report

"pdf" (default) | "html" | "html-file" | "docx"

File format for the generated report, specified as one of these values:

- "pdf" — PDF file

- "html" — HTML report, packaged as a zipped file that contains the HTML file, images, style sheet, and JavaScript files of the report
- "html-file" — HTML report
- "docx" — Microsoft Word document

This argument specifies the `Type` property for the `padv.ProcessAdvisorReportGenerator` object.

Example: "html-file"

OutputPath — Name and path of generated report

"\$PROJECTROOT\$/ProcessAdvisorReport" (default) | string

Name and path of generated report, specified as a string.

By default, the report path uses a relative path to the project root and the pipeline generator generates a report `ProcessAdvisorReport.pdf`.

Example: "tools/myReport"

Data Types: string

Process — Name of process that you want to generate report for

`padv.ProcessModel.DefaultProcessId` (default) | string

Name of process that you want to generate report for, specified by a character vector or string.

By default, the report generator generates a report for the default process. But if you define multiple processes inside your process model, you can generate a report for a different process by using the `Process` argument.

Example: "CIPipeline"

Example: "Fail-Fast"

Data Types: string

Examples


Generate Report with Task Results




Run a task and generate a report with the task results.

Open the Process Advisor example project.

```
processAdvisorExampleStart
```

This command creates a copy of the Process Advisor example project and opens Process Advisor on the model `AHRS_Voter`.

Run a task. For this example, in Process Advisor, point to the task **Generate Simulink Web View** and click the run button .

Tasks	I/O	Details
<input type="radio"/> Collect Model Maintainability Metrics		
<input type="radio"/> Generate Simulink Web View   		
<input type="radio"/> Check Modeling Standards		
<input type="radio"/> Detect Design Errors		

Run this task and outdated dependent tasks

Use the `generateReport` function to generate an HTML report with the task results.

```
generateReport(padv.ProcessAdvisorReportGenerator(Format="html-file"))
```

The report, `ProcessAdvisorReport.html`, generates in the current working folder.

Open and inspect the report. The report shows a summary of the task status, results, inputs, and outputs.

See Also

`generateReport` | `mlreportgen.report.Report` | Process Advisor

Topics

“Automate and Run Tasks with Process Advisor”

“Programmatically Run Tasks”

generateReport

Generate report for recent task results

Syntax

```
generateReport(reportSettings)
```

Description

`generateReport(reportSettings)` generates a report with task results.

After you run tasks using the Process Advisor app or `runprocess` function, you can call the `generateReport` function on a `padv.ProcessAdvisorReportGenerator` object to generate a report of the task results.

Alternatively, you can use `runprocess` with the `GenerateReport` name-value argument specified as `true`: `runprocess(GenerateReport = true)`.

Examples


Generate Report with Task Results

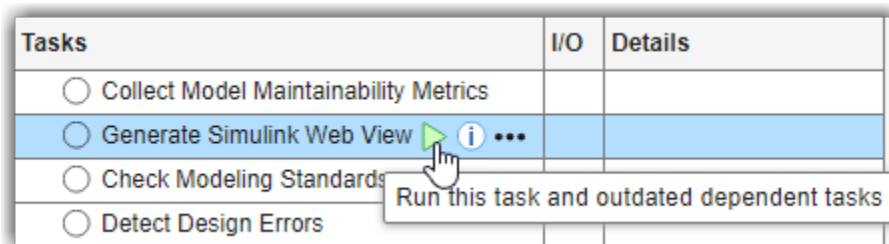
Run a task and generate a report with the task results.

Open the Process Advisor example project.

```
processAdvisorExampleStart
```

This command creates a copy of the Process Advisor example project and opens Process Advisor on the model `AHRS_Voter`.

Run a task. For this example, in Process Advisor, point to the task **Generate Simulink Web View** and click the run button .



Use the `generateReport` function to generate an HTML report with the task results.

```
generateReport(padv.ProcessAdvisorReportGenerator(Format="html-file"))
```

The report, `ProcessAdvisorReport.html`, generates in the current working folder.

Open and inspect the report. The report shows a summary of the task status, results, inputs, and outputs.

Input Arguments

reportSettings — Report generation settings

`padv.ProcessAdvisorReportGenerator` object

Report generation settings, specified as a `padv.ProcessAdvisorReportGenerator` object.

Example: `padv.ProcessAdvisorReportGenerator`

Example: `padv.ProcessAdvisorReportGenerator(Format="html-file")`

Tips

- If you want to run tasks and generate a report in batch mode, you can specify the `runprocess` argument `ExitInBatchMode` as `false` and use the `exitCode` returned by `runprocess` to exit. For example:

```
[buildResult, exitCode] = runprocess(ExitInBatchMode=false);  
rptObj = padv.ProcessAdvisorReportGenerator();  
generateReport(rptObj);  
exit(exitCode);
```

Otherwise, the `runprocess` function automatically exits MATLAB before the report can generate.

- Alternatively, you can use `runprocess` with the `GenerateReport` name-value argument specified as `true`: `runprocess(GenerateReport = true)`.

Alternative Functionality

You can export a build report that summarizes the task statuses, task results, and other information about the task execution by using the Process Advisor app. In the Process Advisor toolstrip, in the **Export** section, click **Report**. In the Export Report dialog box, you can specify options for the report and export a report for the current process by clicking **Export**.

See Also

`padv.ProcessAdvisorReportGenerator` | `runprocess` | Process Advisor

Topics

“Automate and Run Tasks with Process Advisor”

“Programmatically Run Tasks”

Utilities

padv.util.ArtifactAddress

Address for artifact in project

Description

Use the `padv.util.ArtifactAddress` object to represent the address of an artifact in your project.

Creation

Syntax

```
addressObj = padv.util.ArtifactAddress(filePath)
addressObj = padv.util.ArtifactAddress( ____,Name=Value)
```

Description

`addressObj = padv.util.ArtifactAddress(filePath)` creates an artifact address by using the file path specified by `filePath`. You can access information inside the artifact address object by using the object functions listed below.

`addressObj = padv.util.ArtifactAddress(____,Name=Value)` creates an artifact address using the settings specified by one or more name-value arguments. For example, to create an artifact address that specifies the name of the project that contains the artifact, specify `OwningProjectName=projectName`.

Input Arguments

filePath — File path

string array

File path, specified as a string array.

Example: `padv.util.ArtifactAddress(fullfile("tools","sampleChecks.json"))`

Data Types: string

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, . . . ,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `padv.util.ArtifactAddress(filePath,OwningProjectName=projectName)`

OwningProjectName — Project that contains artifact

string array

Project that contains the artifact, specified as a string array.

You can retrieve the owning project name of an artifact address object by using the `getOwningProject` object function.

Example: "ProcessAdvisorExample"

Data Types: string

Track — Setting for tracking changes to artifact

true or 1 | false or 0

Setting for tracking changes to the artifact, specified as a numeric or logical 1 (true) or 0 (false).

For more information, see “Exclude Files from Change Tracking in Process Advisor”.

Example: false

Data Types: logical

Object Functions

Function	Description
<code>getFileAddress</code>	Get address of file on disk. <code>getFileAddress(addressObj)</code>
<code>getKey</code>	Get unique address of artifact. <code>getKey(addressObj)</code>
<code>getOwningProject</code>	Get name of project that contains the artifact. <code>getOwningProject(addressObj)</code>
<code>getAbsolutePath</code>	Get the absolute path of the artifact. <code>getAbsolutePath(addressObj)</code>
<code>isFileArtifact</code>	Determine if input is file. <code>isFileArtifact(addressObj)</code>
<code>isSubFileArtifact</code>	Determine if input is subfile. A subfile is a part of a larger file. For example, a subsystem is a subfile of a model file. <code>isSubFileArtifact(addressObj)</code>

Examples

Specify Address for Artifact

Create artifact address for file in project.

```
addressObj = padv.util.ArtifactAddress(...
fullfile("tools", "sampleChecks.json"));
```

Use artifact address to create `padv.Artifact` object.

```
paArtifact = padv.Artifact("other_file",addressObj)
```

Specify Which Project Contains Artifact

Specify the name of the project that contains the artifact.

```
projectName = "My Reference Project";
```

Specify that the project contains the artifact.

```
addressObj = padv.util.ArtifactAddress(...  
fullfile("tools", "sampleChecks.json"), ...  
OwningProjectName=projectName)
```

You can view which project contains the artifact by using the `getOwningProject` function.

```
getOwningProject(addressObj)
```

```
ans =
```

```
    "My Reference Project"
```

See Also

Process Advisor | `runprocess`

Topics

“Automate and Run Tasks with Process Advisor”

“Create Custom Tasks”

“Best Practices for Process Model Authoring”

padv.util.closeModelsLoadedByTask

Close models loaded by task

Syntax

```
padv.util.closeModelsLoadedByTask(PreviouslyLoadedModels = modelList)
```

Description

`padv.util.closeModelsLoadedByTask(PreviouslyLoadedModels = modelList)` closes models that were loaded by the current task. The function determines which models the task loaded by comparing the current list of loaded models to a list of previously loaded models, `modelList`. The function uses `close_system(model, 0)` to close the models without saving.

Use this function inside the run function of a custom task to close models loaded by the task. Note that the function does not close models that are open in the Simulink Editor.

Examples

Close Models Loaded by Task

Find which models were already loaded and then use the function

```
padv.util.closeModelsLoadedByTask
```

to close only models loaded by the current task.

Inside the run function for your custom task, use the function `get_param` to find and save a list of the previously loaded models. Then, after your task performs its action and specifies the task results, close the models loaded by the task. For example, the run function in your custom task might look like:

```
function taskResult=run(obj, input)
    % Before the task loads models, save a list of the models that are already loaded.
    loadedModels = get_param(Simulink.allBlockDiagrams(), 'Name');

    % <load models for this task>
    % <specify task results>

    % Close models that were loaded by this task.
    padv.util.closeModelsLoadedByTask(PreviouslyLoadedModels=loadedModels);
end
```

Input Arguments

modelList — List of previously loaded models

List of previously loaded models, specified as an array of model names.

You can use the function `get_param` to find the currently loaded models:

```
loadedModels = get_param(Simulink.allBlockDiagrams(), 'Name');
```

Example: {'modelA'; 'modelB'; 'modelC'}

See Also

Process Advisor | `runprocess`

Topics

“Automate and Run Tasks with Process Advisor”

“Create Custom Tasks”

“Best Practices for Process Model Authoring”

padv.util.forceReanalyzeProject

Reanalyze project and log analysis events

Syntax

```
padv.util.forceReanalyzeProject()
```

Description

`padv.util.forceReanalyzeProject()` forces a reanalysis of the current project by creating backups of the existing artifact database (`artifacts.dmr`), clearing the existing project analysis, and reanalyzing the project.

The function also logs project analysis events, which can help with troubleshooting persistent project analysis issues. Note that when you run the function, the function closes and reopens the project.

The function creates backup files and detailed logs in the `derived` folder in the project and creates a ZIP file containing these artifacts for further analysis. The files include:

- `artifacts_no_update.dmr.bak` — Backup of the `artifacts.dmr` file before update
- `artifacts_update.dmr.bak` — Backup of the `artifacts.dmr` file after update
- `artifacts_new.dmr.bak` — Backup of the `artifacts.dmr` file after reanalysis
- `dt_Event_Log.txt` — Event log file
- `detailed_logs.txt` — Detailed log file
- `logs.zip` — ZIP file containing the above files

Note You should only use the function `padv.util.forceReanalyzeProject` when there are unexpected project analysis issues. When you clear the existing project analysis file, you might permanently lose important information, including the UUIDs that the digital thread assigned to artifacts in your project. Reanalyzing a project might take some time to complete. The `artifacts.dmr` file might be used by other project users and if you use other tools that use the digital thread, you might need to re-run the metrics in those tools.

For general task and result cleanup, use `runprocess` instead. The `runprocess` function has name-value arguments, `Clean` and `DeleteOutputs`, that you can use to clean task results and delete task outputs. For information, see `runprocess`.

See Also

Process Advisor | `runprocess`

Topics

“Automate and Run Tasks with Process Advisor”

“Create Custom Tasks”

“Best Practices for Process Model Authoring”

padv.util.getCurrentProject

Get current project and persist project instance

Syntax

```
cp = padv.util.getCurrentProject()
```

Description

`cp = padv.util.getCurrentProject()` gets the currently open project, and returns a project object, `cp`. You can use this function to get the current project in your code, for example, in custom queries. This function can be faster than the `currentProject` function because `cp` is a persistent variable.

Examples

Get Current Project

Get the current project, represented by a `matlab.project.Project` object.

Open the Process Advisor example project.

```
processAdvisorExampleStart
```

Get the current project.

```
cp = padv.util.getCurrentProject()
```

Output Arguments

cp — Current project

`matlab.project.Project`

Current project, returned as a `matlab.project.Project` object. `cp` is a persistent variable that can remain in memory between calls to the function.

If you do not have a project open, then the function returns an empty array.

See Also

Process Advisor | `runprocess`

Topics

“Automate and Run Tasks with Process Advisor”

“Create Custom Tasks”

“Best Practices for Process Model Authoring”

padv.util.getModelName

Namespace: padv.util

Find name of model that contains artifact

Syntax

```
modelName = padv.util.getModelName(artifact)
```

Description

`modelName = padv.util.getModelName(artifact)` returns the name of the model that contains artifact.

Input Arguments

artifact — Artifact information

`padv.Artifact` object

Artifact information, specified as a `padv.Artifact` object.

You can create a `padv.Artifact` object either by:

- Running a built-in query. When you run a built-in query, the query returns either a `padv.Artifact` object or an array of `padv.Artifact` objects.
- Using the `padv.Artifact` class.

Example:

```
padv.Artifact("sl_model_file", padv.util.ArtifactAddress(fullfile("02_Models",  
"AHRS_Voter", "specification", "AHRS_Voter.slx")))
```

Output Arguments

modelName — Name of model that contains artifact

string

Name of model that contains artifact, returned as a string.

See Also

Process Advisor | `runprocess`

Topics

“Automate and Run Tasks with Process Advisor”

“Create Custom Tasks”

“Best Practices for Process Model Authoring”

padv.util.getProjectReferences

Get list of project references

Syntax

```
prjReferences = padv.util.getProjectReferences()  
prjReferences = padv.util.getProjectReferences("reset")
```

Description

`prjReferences = padv.util.getProjectReferences()` gets a list of the project references for the current project. The function caches the list.

`prjReferences = padv.util.getProjectReferences("reset")` resets the cached list of project references.

Examples

Get List of Project References

Get a list of the project references for the current project.

Open the Process Advisor example for project references.

```
processAdvisorProjectReferenceExampleStart
```

Get the list of project references for the current project.

```
prjReferences = padv.util.getProjectReferences()
```

Output Arguments

prjReferences — Project references

ProjectReference object | array of ProjectReference objects

Project references, returned as a ProjectReference object or an array of ProjectReference objects.

See Also

Process Advisor | `runprocess`

Topics

“Automate and Run Tasks with Process Advisor”

“Create Custom Tasks”

“Best Practices for Process Model Authoring”

padv.util.getTestCaseID

Find ID for test case that contains artifact

Syntax

```
testCaseID = padv.util.getTestCaseID(artifact)
```

Description

`testCaseID = padv.util.getTestCaseID(artifact)` returns the ID for the test case that contains `artifact`.

Examples

Find Test Case ID Associated with Artifact

Find the test case ID for a test case by using `padv.util.getTestCaseID`.

Open the Process Advisor example project. In the MATLAB Command Window, enter:

```
processAdvisorExampleStart
```

Create a query that can find the test cases in the project. Since test cases are part of a larger test file, test cases are subfile artifacts and you must specify `FilterSubFileArtifacts` as `false` to stop the query from filtering out the test cases.

```
q = padv.builtin.query.FindArtifacts(ArtifactType = "sl_test_case", ...
  FilterSubFileArtifacts = false);
```

Find the test cases in the project by running the query. The query returns the as an array of `padv.Artifact` objects.

```
testCaseArtifacts = run(q);
```

Find the test case ID for one of the test cases returned by the query.

```
id = padv.util.getTestCaseID(testCaseArtifacts(1))
```

Input Arguments

artifact — Artifact information

`padv.Artifact` object

Artifact information, specified as a `padv.Artifact` object.

You can create a `padv.Artifact` object either by:

- Running a built-in query. When you run a built-in query, the query returns either a `padv.Artifact` object or an array of `padv.Artifact` objects.

- Using the `padv.Artifact` class.

Example:

```
padv.Artifact("sl_model_file", padv.util.ArtifactAddress(fullfile("02_Models",  
"AHRS_Voter", "specification", "AHRS_Voter.slx"))))
```

Output Arguments

testCaseID — ID for test case that contains artifact

string

ID for the test case that contains the artifact, returned as a string.

See Also

Process Advisor | `runprocess`

Topics

“Automate and Run Tasks with Process Advisor”

“Create Custom Tasks”

“Best Practices for Process Model Authoring”

padv.util.mergeArtifactDatabases

Merge artifact database files

Syntax

```
padv.util.mergeArtifactDatabases(Base = baseFile, Branches = filesToMerge,
Merged = mergedFile)
padv.util.mergeArtifactDatabases( ____, CheckOutdatedResults = false)
```

Description

`padv.util.mergeArtifactDatabases(Base = baseFile, Branches = filesToMerge, Merged = mergedFile)` merges the artifact database files, `filesToMerge`, with the common ancestor artifact database file, `baseFile`, to create a merged artifact database file `mergedFile`.

You can use this function to merge artifact database files from different feature branches or CI pipeline jobs. The function requires an open project.

`padv.util.mergeArtifactDatabases(____, CheckOutdatedResults = false)` merges without validating that task results are still up-to-date after the merge. Outdated results are not expected if the merge is successful. When there are outdated results, there could be an issue with the merge. By default, `CheckOutdatedResults` is `true`.

Note Only supported in R2023b Update 5 and later releases.

Examples

Merge Project Analysis from Different Feature Branches

Process Advisor and the build system are able to detect changes to project files and identify outdated tasks by using the information in the artifact database file `artifacts.dmr`. When your team works on a project with multiple feature branches, you might need to merge different versions of `artifacts.dmr` into a single file that contains the latest project analysis. To create the file, you need to save a copy of the base artifact database file and then merge the `artifacts.dmr` files from each branch.

When your team members clone the project from source control, have them download the latest derived files, including the `artifacts.dmr` file that contains the latest analysis of the project. By default, digital thread stores the artifact database file inside the `derived` folder in the project root.

You can use a database or repository management tool to handle derived files effectively.

To resolve conflicts between the artifact database files from the different feature branches, you need to create a base artifact database file. Use the most recent `artifacts.dmr` file from the derived files as the base because that file represents the latest shared state of project analysis across the feature branches.

Create a copy of the artifact database file inside the `derived` folder and name the file `base.dmr`.

```
padv.util.saveArtifactDatabase(fullfile("derived", "base.dmr"))
```

As each team member works on their separate branches, the digital thread updates the `artifacts.dmr` file in their copy of the project to reflect their changes.

After a team member makes the changes on their branch, use the function `padv.util.saveArtifactDatabase` in each branch to save a copy of the artifact database file from that branch. For example, you might have artifact database files like `featureA.dmr` and `featureB.dmr`.

Merge the artifact database files into a new `artifacts.dmr` file by using the function `padv.util.mergeArtifactDatabases`. The base artifact database file is `base.dmr` and the artifact database files from the branches are `featureA.dmr` and `featureB.dmr`.

```
padv.util.mergeArtifactDatabases(...
Base = fullfile("derived", "base.dmr"), ...
Branches = [fullfile("derived", "featureA.dmr"), fullfile("derived", "featureB.dmr")], ...
Merged = fullfile("derived", "artifacts.dmr"))
```

This section describes how to merge artifact database files from separate feature branches, but you can also use these functions to merge artifact database files from jobs in CI and tasks that you run in parallel. Starting in R2023b Update 5, GitHub and Jenkins pipelines that you generate by using the function `padv.pipeline.generatePipeline` automatically merge artifact database files.

Input Arguments

baseFile — Path and name of base artifact database file

string

Path and name of base artifact database file, specified as a string.

The base artifact database file is the common ancestor of the artifact database files that you want to merge. The path must be relative to the project root or an absolute path.

To create a common ancestor, you can save a copy of an artifact database file by using the function `padv.util.saveArtifactDatabase`.

Example: `fullfile("derived", "base.dmr")`

Data Types: string

filesToMerge — Paths and names of artifact database files to merge

string array

Paths and names of artifact database files that you want to merge, specified as a string array.

Example: `[fullfile("derived", "modelA.dmr"), fullfile("derived", "modelB.dmr")]`

Data Types: string

mergedFile — Path and name of merged artifact database file

string

Path and name of merged artifact database file, specified as a string.

The path must be relative to the project root or an absolute path.

Example: `fullfile("derived", "artifacts.dmr")`

Data Types: `string`

Version History

Introduced in R2023b

See Also

Process Advisor | `runprocess`

Topics

“Automate and Run Tasks with Process Advisor”

“Create Custom Tasks”

“Best Practices for Process Model Authoring”

padv.util.refreshProcessModel

Refresh process model data

Syntax

```
padv.util.refreshProcessModel()
```

Description

`padv.util.refreshProcessModel()` refreshes the process model. Use this function if you need to manually refresh the process model data.

Examples

Refresh Process Model

Make a change to a project and programmatically refresh the process model data.

Open the example project for Process Advisor.

```
processAdvisorExampleStart
```

The `AHRS_Voter` model opens.

Make a change to the `AHRS_Voter` model and re-save the model.

The warning banner in Process Advisor shows that the process model data needs to be refreshed.

Programmatically refresh the process model data by using `padv.util.refreshProcessModel`.

```
padv.util.refreshProcessModel
```

See Also

Process Advisor | `runprocess`

Topics

“Automate and Run Tasks with Process Advisor”

“Create Custom Tasks”

“Best Practices for Process Model Authoring”

padv.util.saveArtifactDatabase

Save copy of artifact database file

Syntax

```
padv.util.saveArtifactDatabase(destination)
```

Description

`padv.util.saveArtifactDatabase(destination)` saves a copy of the artifact database file in the destination specified by `destination`.

The artifact database file, `artifacts.dmr`, is saved in the `derived` folder in the project root. This file tracks the project artifacts and their dependencies. Manually copying this file can lead to inconsistencies or incorrect behavior due to pending artifact changes.

You can use this function to create base artifact database files and save copies of artifact database files from different feature branches or CI pipeline jobs.

The function requires an open project and requires CI/CD Automation for Simulink Check.

Note Only supported in R2023b Update 5 and later releases.

Examples

Merge Project Analysis from Different Feature Branches

Process Advisor and the build system are able to detect changes to project files and identify outdated tasks by using the information in the artifact database file `artifacts.dmr`. When your team works on a project with multiple feature branches, you might need to merge different versions of `artifacts.dmr` into a single file that contains the latest project analysis. To create the file, you need to save a copy of the base artifact database file and then merge the `artifacts.dmr` files from each branch.

When your team members clone the project from source control, have them download the latest derived files, including the `artifacts.dmr` file that contains the latest analysis of the project. By default, digital thread stores the artifact database file inside the `derived` folder in the project root.

You can use a database or repository management tool to handle derived files effectively.

To resolve conflicts between the artifact database files from the different feature branches, you need to create a base artifact database file. Use the most recent `artifacts.dmr` file from the derived files as the base because that file represents the latest shared state of project analysis across the feature branches.

Create a copy of the artifact database file inside the `derived` folder and name the file `base.dmr`.

```
padv.util.saveArtifactDatabase(fullfile("derived", "base.dmr"))
```

As each team member works on their separate branches, the digital thread updates the `artifacts.dmr` file in their copy of the project to reflect their changes.

After a team member makes the changes on their branch, use the function `padv.util.saveArtifactDatabase` in each branch to save a copy of the artifact database file from that branch. For example, you might have artifact database files like `featureA.dmr` and `featureB.dmr`.

Merge the artifact database files into a new `artifacts.dmr` file by using the function `padv.util.mergeArtifactDatabases`. The base artifact database file is `base.dmr` and the artifact database files from the branches are `featureA.dmr` and `featureB.dmr`.

```
padv.util.mergeArtifactDatabases(...  
Base = fullfile("derived", "base.dmr"), ...  
Branches = [fullfile("derived", "featureA.dmr"), fullfile("derived", "featureB.dmr")], ...  
Merged = fullfile("derived", "artifacts.dmr"))
```

This section describes how to merge artifact database files from separate feature branches, but you can also use these functions to merge artifact database files from jobs in CI and tasks that you run in parallel. Starting in R2023b Update 5, GitHub and Jenkins pipelines that you generate by using the function `padv.pipeline.generatePipeline` automatically merge artifact database files.

Input Arguments

destination — File destination

string

File destination for copied artifact database file, specified as a string.

The path must be relative to the project root or an absolute path and must include the `.dmr` extension.

Example: `fullfile("derived", "base.dmr")`

Data Types: string

Version History

Introduced in R2023b

See Also

Process Advisor | `runprocess`

Topics

“Automate and Run Tasks with Process Advisor”

“Create Custom Tasks”

“Best Practices for Process Model Authoring”

padv.util.unpackExternalCodeCache

Unpack code generation target from Simulink cache files

Syntax

```
padv.util.unpackExternalCodeCache(cacheFiles)
```

Description

`padv.util.unpackExternalCodeCache(cacheFiles)` unpacks the code generation target from the Simulink cache files, `cacheFiles`.

An external code cache allows your team to generate code in parallel while maintaining up-to-date task results. For information on parallel code generation, see the property `GenerateExternalCodeCache` for the built-in task `padv.builtin.task.GenerateCode`.

If your team generates code in parallel by generating an external code cache, downstream tasks that depend on the generated code need to unpack the generated code target before running the task action. Built-in tasks that depend on generated code, like `padv.builtin.task.AnalyzeModelCode`, unpack the code generation target by using the utility function `padv.util.unpackExternalCodeCache`.

Examples

Unpack Code Generation Target

Generate and unpack code generation target.

Open the parallel code generation example.

```
processAdvisorParallelExampleStart
```

Generate code by running a code generation task iteration. For example, run the code generation task on the reference model `OuterLoop_Control`.

```
runprocess(Tasks = "padv.builtin.task.GenerateCode", ...
    FilterArtifact = fullfile("02_Models","OuterLoop_Control", ...
    "specification","OuterLoop_Control.slx"));
```

Find the external code cache file by using the built-in query.

```
q = padv.builtin.query.FindExternalCodeCache;
artifactsArray = run(q);
```

Unpack the cache file.

```
padv.util.unpackExternalCodeCache(artifactsArray);
```

Input Arguments

cacheFiles — Address for external code cache files

array of `padv.Artifact` objects | cell array of character vectors | string array

Absolute or relative address for external code cache files, specified as either an array of `padv.Artifact` objects, a cell array of character vectors, or a string array.

The built-in code generation task, `padv.builtin.task.GenerateCode`, generates these cache files when you specify the task property `GenerateExternalCodeCache` as `true`.

The files must be:

- `.slxc.bk` files
- compatible with the `slxcunpack` function
- inside the project root folder

See Also

Process Advisor | `runprocess`

Topics

“Automate and Run Tasks with Process Advisor”

“Create Custom Tasks”

“Best Practices for Process Model Authoring”

Built-In Task Library

padv.builtin.task.AnalyzeModelCode Class

Namespace: padv.builtin.task padv.builtin.task padv.builtin.task

Superclasses: padv.Task

Task for checking coding standards or proving code quality with Polyspace

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.task.AnalyzeModelCode` class provides a task that can check coding standards or prove code quality. By default, the task quickly analyzes generated model code for many types of run-time defects, coding standards, and code metrics by using Polyspace® Bug Finder™. But you can use the property `VerificationMode` to reconfigure the task to check *every* operation in your code for a set of possible run-time errors and try to prove the absence of the error for all execution paths by using Polyspace Code Prover™. For more information, see `VerificationMode`.

This task runs on the generated model code, iterating over either each model in the project or the project itself. If a model does not have generated code, the task skips the model and displays a warning message. You can generate code using the built-in task `padv.builtin.task.GenerateCode` and then analyze the generated code using the `AnalyzeModelCode` task. You can add these tasks to your process model by using the method `addTask`. After you add the tasks to your process model, you can run the tasks from the Process Advisor app or by using the function `runprocess`.

To view the source code for this built-in task, in the MATLAB Command Window, enter:

```
open padv.builtin.task.AnalyzeModelCode
```

The `padv.builtin.task.AnalyzeModelCode` class is a `handle` class.

Note Starting in R2023b, this task is not supported on macOS (Apple silicon).

Creation

Description

`task = padv.builtin.task.AnalyzeModelCode()` creates a task for checking coding standards with Polyspace Bug Finder.

`task = padv.builtin.task.AnalyzeModelCode(Name=Value)` sets certain properties using one or more name-value arguments. For example, `task = padv.builtin.task.AnalyzeModelCode(Name = "MyAnalyzeModelCode")` creates a task with the specified name.

You can use this syntax to set property values for `TreatAsRefModel`, `Name`, `Title`, `DescriptionText`, `DescriptionCSH`, `IterationQuery`, `InputQueries`, `InputDependencyQuery`, `LaunchToolAction`, and `LaunchToolText`.

The class also has other properties, but you cannot set those properties during task creation.

Properties

The `AnalyzeModelCode` class inherits properties from `padv.Task`. The properties listed in “Specialized Inherited Properties” on page 7-0 are `padv.Task` properties that the `AnalyzeModelCode` task overrides.

The task also has properties for specifying:

- “General Polyspace Options” on page 7-0 for specifying the verification mode, result directory, and generated reports
- “Advanced Polyspace Analysis Options” on page 7-0 for batch and scheduler options
- “Advanced Polyspace Project Options” on page 7-0 for managing project files
- “Advanced Polyspace Access Configuration Options” on page 7-0 for uploading results to Polyspace Access™

Specialized Inherited Properties

Name — Unique identifier for task in process

"padv.builtin.task.AnalyzeModelCode" (default) | string

Unique identifier for task in process, specified as a string.

Example: "MyAnalyzeModelCode"

Data Types: string

Title — Human-readable name that appears in Process Advisor app

"Check Coding Standards" (default) | string

Human-readable name that appears in Process Advisor app, specified as a string.

Example: "My Analyze Model Code Task"

Data Types: string

DescriptionText — Task description

"This task uses Polyspace Bug Finder to analyze generated model code for run-time defects, coding standards, and code metrics." (default) | string

Task description, specified as a string.

When you point to a task in Process Advisor and click the information icon, the tooltip shows the task description.

Example: "This task analyzes generated model code."

Data Types: string

DescriptionCSH — Path to task documentation

path to `AnalyzeModelCode` documentation (default) | string

Path to task documentation, specified as a string.

When you point to a task in Process Advisor, click the ellipsis (...), and click **Help**, Process Advisor opens the task documentation.

Example: `fullfile(pwd,"taskHelpFiles","myTaskDocumentation.pdf")`

Data Types: string

RequiredIterationArtifactType – Artifact type that task can run on

"sl_model_file" (default) | ...

Type of artifact, specified as one or more of the values listed in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file

Category	Artifact Type	Description
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
"sl_test_suite_result"	Simulink Test suite result	

Example: "sl_model_file"

Example: ["sl_model_file" "zc_file"]

IterationQuery – Find artifacts that task iterates over

padv.builtin.query.FindModels (default) | padv.Query object | name of padv.Query object

Query that finds the artifacts that the task iterates over, specified as a `padv.Query` object or the name of a `padv.Query` object. When you specify `IterationQuery`, the task runs one time *for each* artifact returned by the query. In the Process Advisor app, the artifacts returned by `IterationQuery` appear under task title.

By default, this task runs on the generated model code, iterating over either each model in the project or the project itself. If a model does not have generated code, the task skips the model and displays a warning message.

For more information about task iterations, see "Overview of Process Model".

Example: `padv.builtin.query.FindProjectFile`

Licenses — List of licenses that task requires

`["matlab_coder" "real-time_workshop" "rtw_embedded_coder"]` (default) | string

List of licenses that the task requires, specified as a string.

Data Types: string

LaunchToolAction — Function that launches tool

`@launchToolAction` (default) | function handle

Function that launches a tool, specified as the function handle.

When you point to a task in the Process Advisor app, you can click the ellipsis (...) to see more options. For built-in tasks, you have the option to launch a tool associated with the task.

For the task `AnalyzeModelCode`, you can launch the Polyspace Code Verifier app.

Data Types: `function_handle`

LaunchToolText — Description of action that LaunchToolAction property performs

`"Open Polyspace Code Verifier"` (default) | string

Description of the action that the `LaunchToolAction` property performs, specified as a string.

Data Types: string

InputQueries — Inputs to task

`padv.builtin.query.GetOutputsOfDependentTask(Task="padv.builtin.task.GenerateCode")` (default) | `padv.Query` object | name of `padv.Query` object | array of `padv.Query` objects

Inputs to the task, specified as:

- a `padv.Query` object
- the name of `padv.Query` object
- an array of `padv.Query` objects
- an array of names of `padv.Query` objects

By default, the task `AnalyzeModelCode` gets the model and generated code inputs by using the built-in queries:

- `padv.builtin.query.GetIterationArtifact`
- `padv.builtin.query.GetOutputsOfDependentTask` on the task `padv.builtin.task.GenerateCode`

InputDependencyQuery — Finds artifact dependencies for task inputs

`padv.builtin.query.GetDependentArtifacts` (default) | `padv.Query` object | name of `padv.Query` object

Query that finds artifact dependencies for task inputs, specified as a `padv.Query` object or the name of a `padv.Query` object.

The build system runs the query specified by `InputDependencyQuery` to find the dependencies for the task inputs, since those dependencies can impact if task results are up-to-date.

For more information, see “Overview of Process Model”.

Example: `padv.builtin.query.GetDependentArtifacts`

General Polyspace Options

TreatAsRefModel — Treat code as code generated by reference model

"" (default) | true | false

Treat code as code generated by reference model, specified as either "", true, or false.

By default, the task automatically identifies whether a model is a top model or a reference model before analyzing the model code. But you can specify `TreatAsRefModel` as true or false if you want to override the behavior and only analyze reference model code or top model code.

Example: true

Data Types: logical

ResultDir — Directory where build system stores results from analyzing model code

fullfile('\$DEFAULTOUTPUTDIR\$', '\$ITERATIONARTIFACT\$', 'ps_results') (default) | string

Directory where build system stores results from analyzing model code, specified as a string.

Data Types: string

Reports — Reports output by task

["BugFinderSummary" "CodingStandards"] (default) | string

Reports output by task, specified as a string.

Data Types: string

ReportPath — Path to reports output by task

string(fullfile('\$DEFAULTOUTPUTDIR\$', '\$ITERATIONARTIFACT\$', 'ps_results')) (default) | string

Path to reports output by task, specified as a string.

Data Types: string

ReportFormat — Format of output reports

"PDF" (default) | "HTML" | "WORD" | comma-separated list of formats

Format of output reports, specified as either:

- "PDF" — PDF file.
- "HTML" — HTML report.
- "WORD" — Microsoft Word document.
- Combination of these formats, specified as a comma-separated list. For example, "PDF,HTML" generates multiple reports. One in PDF format and one in HTML format.

Example: "HTML"

Data Types: string

ReportNames — Names of output reports

["\$ITERATIONARTIFACT\$_BugFinderSummary" "\$ITERATIONARTIFACT\$_CodingStandards"] (default) | string

Names of output reports, specified as a string.

Data Types: string

VerificationMode — Polyspace mode

"BugFinder" (default) | "CodeProver"

Polyspace mode, specified as either:

- "BugFinder" — Quickly analyze generated model code for many types of run-time defects, coding standards, and code metrics by using Polyspace Bug Finder.
- "CodeProver" — Check *every* operation in your code for a set of possible run-time errors and try to prove the absence of the error for all execution paths by using Polyspace Code Prover.

You can use both Bug Finder and Code Prover in your software development process, but each task instance must have a unique Name and you need to configure the tasks to prevent the tasks from overwriting each other.

For information on the differences between Bug Finder and Code Prover, see "Differences Between Polyspace Bug Finder and Polyspace Code Prover" (Polyspace Bug Finder).

Example: "CodeProver"

Advanced Polyspace Analysis Options**Batch — Option to run analysis on server (-batch)**

false or 0 (default) | true or 1

Option to run analysis on server (-batch), specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

Scheduler — Specify cluster or job scheduler (-scheduler)

string.empty (default) | string

Specify cluster or job scheduler (-scheduler), specified as a string.

Example: "NodeHost"

Data Types: string

Advanced Polyspace Project Options**SavePsPrjFileAfterAnalysis — Save Polyspace project file after analyzing model code**

true or 1 (default) | false or 0

Save Polyspace project file after analyzing model code, specified as a numeric or logical 1 (true) or 0 (false).

Example: false

Data Types: logical

PsPrjFileName — Polyspace project file path

"\$ITERATIONARTIFACT\$_BugFinder" (default) | string

Polyspace project file path, specified as a string.

The built-in tasks use tokens, like \$ITERATIONARTIFACT\$, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Data Types: string

Advanced Polyspace Access Configuration Options**PsAccessEnable — Upload results to Polyspace Access**

false or 0 (default) | true or 1

Enable task to upload analysis results to Polyspace Access, specified as a numeric or logical 1 (true) or 0 (false).

Note If you specify PsAccessEnable as true, you must also specify values for the other Polyspace Access Configuration Options. For information, see “Upload Results to Polyspace Access” on page 7-15.

Example: true

Data Types: logical

PsAccessHostName — Polyspace Access machine host name

"" (default) | string

Polyspace Access machine host name, specified as a string. You can find the host name in the URL of the Polyspace Access interface, for example, `https://hostname:port/metrics/index.html`.

Example: my-company-server

Data Types: string

PsAccessPortNumber — Polyspace Access port

"9443" (default) | string

Polyspace Access port, specified as a string. You can find the port number in the URL of the Polyspace Access interface, for example, `https://hostname:portNumber/metrics/index.html`.

Example: "9999"

Data Types: string

PsAccessProtocol — HTTP protocol used to access Polyspace Access

"https" (default) | "http"

HTTP protocol used to access Polyspace Access, specified as either "http" or "https".

Example: "http"

PsAccessCredentialsFile — Full path to text file where you store your login credentials for Polyspace Access

string.empty (default) | string

Full path to text file where you store your login credentials for Polyspace Access, specified as a string.

A credentials file is useful if you do not want to store your credentials in your process model. For information on how to create a credentials file, see “Encrypt Password and Store Login Options in a Variable” (Polyspace Bug Finder).

Alternatively, you can specify an API key (`PsAccessApiKey`) or a username and password (`PsAccessUserName` and `PsAccessEncryptedPassword`) to pass your credentials to Polyspace Access.

Example: `"C:\Users\username\myCredentials.txt"`

Data Types: `string`

PsAccessApiKey — API key for Polyspace Access

`string.empty (default) | string`

API key for Polyspace Access, specified as a string.

For information on API keys and how to assign an API key to a user, see `login options`.

Alternatively, you can use a credentials file (`PsAccessCredentialsFile`) or a username and password (`PsAccessUserName` and `PsAccessEncryptedPassword`) to pass your credentials to Polyspace Access.

Example: `"keyValue123"`

Data Types: `string`

PsAccessUserName — Username for Polyspace Access

`" " (default) | string`

Username for Polyspace Access, specified as a string.

For information on login credentials, see `login options`.

Alternatively, you can use a credentials file (`PsAccessCredentialsFile`) or an API key (`PsAccessApiKey`) to pass your credentials to Polyspace Access.

Data Types: `string`

PsAccessEncryptedPassword — Password for Polyspace Access

`" " (default) | string`

Password for Polyspace Access, specified as a string.

For information on login credentials, see `login options`.

Alternatively, you can use a credentials file (`PsAccessCredentialsFile`) or an API key (`PsAccessApiKey`) to pass your credentials to Polyspace Access.

Data Types: `string`

PsAccessParentFolder — Path of parent folder in Polyspace Access explorer

`" " (default) | string`

Path of parent folder in Polyspace Access explorer under which you store uploaded results, specified as a string.

For more information, see `upload options`.

Example: "public/myProject"

Data Types: string

PsAccessResultsName — Name of uploaded results in Polyspace Access explorer

"" (default) | string

Name of uploaded results in Polyspace Access explorer, specified as a string.

The built-in tasks use tokens, like `$ITERATIONARTIFACT$`, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

For more information, see `upload options`.

Example: "\$ITERATIONARTIFACT\$_CodingStandards"

Data Types: string

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Run Polyspace analysis. Depending on the <code>VerificationMode</code> property value, the task runs either Polyspace Bug Finder or Polyspace Code Prover.</p> <hr/> <p>Note You do not need to manually invoke this method. When you run a task using the Process Advisor app or the <code>runprocess</code> function, the build system automatically invokes the <code>run</code> method for the task.</p> <hr/> <p>The <code>run</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this built-in task, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = run(obj, input) ... end</pre>
-----	---

dryRun	<p>Dry run the task to validate task inputs and generate representative task outputs without actually running the task. The <code>dryRun</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this class, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = dryRun(obj, input) ... end</pre> <p>For more information, see “Dry Run Tasks to Test Process Model”.</p>
launchToolAction	<p>Launch the Polyspace Code Verifier app. Process Advisor uses this method when you open the tool associated with a task.</p>

Examples

Add Polyspace Bug Finder Task to Process

Add a task that can quickly analyze generated model code for many types of run-time defects, coding standards, and code metrics by using Polyspace Bug Finder.

Open the process model for your project. If you do not have a process model, open the Process Advisor app to automatically create a process model.

In the process model file, add the `AnalyzeModelCode` task to your process model by using the `addTask` method. By default, the `AnalyzeModelCode` task performs Bug Finder analysis.

```
psbfTask = pm.addTask(padv.builtin.task.AnalyzeModelCode);
```

You can reconfigure the task behavior by using the task properties. For example, to use a custom configuration from a Polyspace project (`.psprj`) file and customize the task outputs:

```
psbfTask.addInputQueries(padv.builtin.query.FindFileWithAddress( ...
    Type='ps_prj_file',Path=fullfile('tools','CodingRulesOnly_config.psprj')));
psbfTask.ResultDir = string(fullfile('$DEFAULTOUTPUTDIR$', ...
    '$ITERATIONARTIFACT$', 'coding_standards'));
psbfTask.Reports = "CodingStandards";
psbfTask.ReportPath = string(fullfile('$DEFAULTOUTPUTDIR$', ...
    '$ITERATIONARTIFACT$', 'coding_standards'));
psbfTask.ReportNames = "$ITERATIONARTIFACT$_CodingStandards";
psbfTask.ReportFormat = "PDF";
```

The `AnalyzeModelCode` task requires outputs from the `GenerateCode` task. Specify this dependency in your process model by using the `dependsOn` method.

To make sure that you run your tasks using the built-in task `GenerateCode` before you add the task, you can use conditional logic in your process model. For example:


```

includeGenerateCodeTask = true;
includeAnalyzeModelCode = true && exist('polyspaceroot','file');

%% Generate Code
% Tools required: Embedded Coder
% By default, generating code for each model in the project
if includeGenerateCodeTask
    codegenTask = pm.addTask(padv.builtin.task.GenerateCode);
    % ... Optionally specify task property values
end

%% Check coding standards
% Tools required: Polyspace Bug Finder
if includeGenerateCodeTask && includeAnalyzeModelCode
    psbfTask = pm.addTask(padv.builtin.task.AnalyzeModelCode);
    % ... Optionally specify task property values
end

% Set task dependencies
if includeGenerateCodeTask && includeAnalyzeModelCode
    psbfTask.dependsOn(codegenTask);
end

```

This code also makes sure that Polyspace is installed and integrated by checking if the Polyspace installation folder (polyspaceroot) exists before adding the task to the process model.

Add Polyspace Code Prover Task to Process

Add a task that can check *every* operation in your code for a set of possible run-time errors and try to prove the absence of the error for all execution paths by using Polyspace Code Prover.

Open the process model for your project.

In the process model file, add the `AnalyzeModelCode` task to your process model by using the `addTask` function. To have the task use Polyspace Code Prover, specify the `VerificationMode` as `"CodeProver"`.

```

pscpTask = pm.addTask(padv.builtin.task.AnalyzeModelCode(Name="ProveCodeQuality"));
pscpTask.Title = "Prove Code Quality";
pscpTask.VerificationMode = "CodeProver";

```

This code also specifies values for the `Name` and `Title` properties since the default task name and title refer to Bug Finder analysis. You can use the other task properties to specify the report templates and other task settings.

You can reconfigure the task behavior by using the task properties. For example:

```

pscpTask.ResultDir = string(fullfile('$DEFAULTOUTPUTDIR$', ...
    '$ITERATIONARTIFACT$', 'code_quality'));
pscpTask.Reports = ["Developer", "CallHierarchy", "VariableAccess"];
pscpTask.ReportPath = string(fullfile('$DEFAULTOUTPUTDIR$', ...
    '$ITERATIONARTIFACT$', 'code_quality'));
pscpTask.ReportNames = [...
    '$ITERATIONARTIFACT$_Developer', ...
    '$ITERATIONARTIFACT$_CallHierarchy', ...

```

```
    "$ITERATIONARTIFACT$ _VariableAccess"];
    pscpTask.ReportFormat = "PDF";
```

The `AnalyzeModelCode` task requires outputs from the `GenerateCode` task. Specify this dependency in your process model by using the `dependsOn` method.

To make sure that you run your tasks using the built-in task `GenerateCode` before you add the task, you can use conditional logic in your process model. For example:

```
includeGenerateCodeTask = true;
includeProveCodeQuality = true && (~isempty(ver('pscodeprover')) || ~isempty(ver('pscodeproverse

%% Generate Code
% Tools required: Embedded Coder
% By default, generating code for each model in the project
if includeGenerateCodeTask
    codegenTask = pm.addTask(padv.builtin.task.GenerateCode);
    % ... Optionally specify task property values
end

%% Prove Code Quality
% Tools required: Polyspace Code Prover
if includeGenerateCodeTask && includeProveCodeQuality
    pscpTask = pm.addTask(padv.builtin.task.AnalyzeModelCode(Name="ProveCodeQuality"));
    pscpTask.Title = "Prove Code Quality";
    pscpTask.VerificationMode = "CodeProver";
    % ... Optionally specify task property values
end

% Set task dependencies
if includeGenerateCodeTask && includeProveCodeQuality
    pscpTask.dependsOn(codegenTask);
end
```

This code also makes sure that Polyspace Code Prover is available before adding the task to the process model.

Add Both Bug Finder and Code Prover Tasks to Process

You can use both Bug Finder and Code Prover in your software development workflow. Both Polyspace Bug Finder and Polyspace Code Prover detect run-time errors through static analysis.

- Bug Finder quickly analyzes your code and detects many types of defects.
- Code Prover checks every operation in your code for a set of possible run-time errors and try to prove the absence of the error for all execution paths.

To include both a Bug Finder task and a Code Prover task in your process model, you must add two separate instances of the `AnalyzeModelCode` task to the process model. Each instance needs a unique value for the `Name` property. Use the `VerificationMode` property to specify whether the task uses Bug Finder (default) or Code Prover ("CodeProver"). You can use the other task properties to specify the report templates and other task settings. For example, in your process model:

```
%% Check Coding Standards with Polyspace Bug Finder
psbfTask = pm.addTask(padv.builtin.task.AnalyzeModelCode());
```

```

% Report Options
psbfTask.ResultDir = fullfile(defaultResultPath,"bug_finder");
psbfTask.ReportPath = fullfile(defaultResultPath,"bug_finder");

%% Prove Code Quality with Polyspace Code Prover
pscpTask = pm.addTask(padv.builtin.task.AnalyzeModelCode(Name="ProveCodeQuality"));
pscpTask.Title = "Prove Code Quality";
pscpTask.VerificationMode = "CodeProver";
% Report Options
pscpTask.ResultDir = string(fullfile(defaultResultPath,"code_prover"));
pscpTask.Reports = ["Developer", "CallHierarchy", "VariableAccess"];
pscpTask.ReportPath = string(fullfile(defaultResultPath,"code_prover"));
pscpTask.ReportNames = [...
    "$ITERATIONARTIFACT$ _Developer", ...
    "$ITERATIONARTIFACT$ _CallHierarchy", ...
    "$ITERATIONARTIFACT$ _VariableAccess"];

```

Note that this code specifies different result directories and report paths for each task to prevent the task outputs from overwriting each other.

For information on:

- Bug Finder and Code Prover, see “Differences Between Bug Finder and Code Prover” (Polyspace Bug Finder).
- How Bug Finder and Code Prover fit into a software development workflow, see “Workflow Using Both Polyspace Bug Finder and Polyspace Code Prover” (Polyspace Bug Finder).

Upload Results to Polyspace Access

If you have a Polyspace Access license, you can reconfigure the `AnalyzeModelCode` task to automatically upload results to Polyspace Access for your team to review.

Before you reconfigure the task, make sure that you have performed this one-time setup “Prerequisites” (Polyspace Bug Finder).

To reconfigure the task, update your process model to specify the property `PsAccessEnable` as `true` and to specify values for these properties:

- `PsAccessHostName`
- `PsAccessPortNumber`
- `PsAccessProtocol`
- `PsAccessParentFolder`
- And one of the following sets of credentials:
 - `PsAccessCredentialsFile`
 - `PsAccessApiKey`
 - `PsAccessUserName` and `PsAccessEncryptedPassword`

For example, in your process model:

```

%% Check coding standards
psTask = pm.addTask(padv.builtin.task.AnalyzeModelCode());

```

```

psTask.addInputQueries(padv.builtin.query.FindFileWithAddress( ...
    Type = "ps_prj_file",...
    Path = fullfile("tools","CodingRulesOnly_config.psprj"));
psTask.ResultDir = string(fullfile("$DEFAULTOUTPUTDIR", ...
    "$ITERATIONARTIFACT$", "coding_standards"));
psTask.Reports = "CodingStandards";
psTask.ReportPath = string(fullfile("$DEFAULTOUTPUTDIR$", ...
    "$ITERATIONARTIFACT$", "coding_standards"));
psTask.ReportNames = "$ITERATIONARTIFACT$_CodingStandards";
psTask.ReportFormat = "PDF";

% Polyspace Access configuration options
psTask.PsAccessEnable = true;
psTask.PsAccessHostName = "my-polyspace-access";
psTask.PsAccessPortNumber = "9443";
psTask.PsAccessProtocol = "https";
psTask.PsAccessCredentialsFile = "C:\Users\username\myCredentials.txt";
psTask.PsAccessParentFolder = "public/myProject";
psTask.PsAccessResultsName = "$ITERATIONARTIFACT$_CodingStandards";

```

This code uses a credentials file, `myCredentials.txt`, to pass credentials to Polyspace Access, but you can also use an API key or a username and password. For information on how to generate and maintain credentials for Polyspace Access, see [login options](#).

For information on Polyspace Access, see “Send Bug Finder Analysis from Desktop to Locally Hosted Server” (Polyspace Bug Finder) and “Run Polyspace Bug Finder on Server and Upload Results to Web Interface” (Polyspace Bug Finder).

Tips

- This task requires that your Polyspace installation is integrated with MATLAB and Simulink. If you have not already integrated your installation, use the function `polyspacesetup`. For information, see “Integrate Polyspace with MATLAB and Simulink” (Polyspace Bug Finder).
- If you start MATLAB with the `-batch` option, the task requires a Polyspace server product. The required server product depends on the task configuration:
 - **Check Coding Standards** (default) — Requires the Polyspace Bug Finder Server™ product.
 - **Prove Code Quality** — Requires the Polyspace Code Prover Server product.
- You can use both Bug Finder and Code Prover in your software development workflow. For information on how to include both a Bug Finder task and a Code Prover task in your process model, see “Add Both Bug Finder and Code Prover Tasks to Process” on page 7-14.

For information on the differences between Bug Finder and Code Prover, see “Differences Between Polyspace Bug Finder and Polyspace Code Prover” (Polyspace Bug Finder).

See Also

`addTask` | `padv.builtin.task.RunCodeInspection` | `padv.builtin.task.GenerateCode` | `padv.builtin.query.FindCodeForModel` | `padv.ProcessModel` | Process Advisor | `runprocess`

Topics

“Add Tasks to Process”

“Modify Default Process Model to Fit Your Process”

“Overview of Process Model”

“Integrate Polyspace with MATLAB and Simulink” (Polyspace Bug Finder)

padv.builtin.task.CollectMetrics Class

Namespace: padv.builtin.task padv.builtin.task padv.builtin.task

Superclasses: padv.Task

Task for collecting model design and testing metrics

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.task.CollectMetrics` class provides a task that can collect model design and testing metrics using the `metric.Engine` API for the Model Design and Model Testing Dashboards. By default, the task collects model maintainability metrics that can help you monitor the size, architecture, and complexity of the software units and components in your project. But you can reconfigure the task to collect model testing, SIL code testing, or PIL code testing metrics by using the `Dashboard` property to specify which dashboard you want to collect metrics for. You can add these tasks to your process model by using the method `addTask`. After you add the tasks to your process model, you can run the tasks from the Process Advisor app or by using the function `runprocess`.

To view the source code for this built-in task, in the MATLAB Command Window, enter:

```
open padv.builtin.task.CollectMetrics
```

The `padv.builtin.task.CollectMetrics` class is a `handle` class.

Creation

Description

`task = padv.builtin.task.CollectMetrics()` creates a task for collecting model maintainability metrics like size, architecture, and complexity. These are the same metric results that the Model Maintainability Dashboard uses.

`task = padv.builtin.task.CollectMetrics(Name=Value)` sets certain properties using one or more name-value arguments. For example, `task = padv.builtin.task.CollectMetrics(Name = "MyCollectMetricsTask")` creates a task with the specified name.

You can use this syntax to set property values for `InputQueries`, `Name`, `IterationQuery`, `InputDependencyQuery`, `Licenses`, `LaunchToolAction`, and `LaunchToolText`.

The `padv.builtin.task.CollectMetrics` class also has other properties, but you cannot set those properties during task creation.

Properties

The `CollectMetrics` class inherits properties from `padv.Task`. The properties listed in “Specialized Inherited Properties” on page 7-0 are `padv.Task` properties that the `CollectMetrics` task overrides.

The task also has properties for specifying “Metric Collection Options” on page 7-0 . The task uses these properties to specify input arguments for the `getAvailableMetricIds`, `execute`, and `generateReport` functions of the `metric.Engine` API.

Specialized Inherited Properties

Name — Unique identifier for task in process

`"padv.builtin.task.CollectMetrics"` (default) | string

Unique identifier for task in process, specified as a string.

Example: `"MyCollectMetricsTask"`

Data Types: string

Title — Human-readable name that appears in Process Advisor app

`"Collect Model Maintainability Metrics"` (default) | string

Human-readable name that appears in Process Advisor app, specified as a string.

Example: `"My Metric Collection Task"`

Data Types: string

DescriptionText — Task description

`"This task collects and reports metric data used by the model design and testing dashboards."` (default) | string

Task description, specified as a string.

When you point to a task in Process Advisor and click the information icon, the tooltip shows the task description.

Example: `"This task collects and reports metric data used by the model design and testing dashboards."`

Data Types: string

DescriptionCSH — Path to task documentation

`path to CollectMetrics documentation` (default) | string

Path to task documentation, specified as a string.

When you point to a task in Process Advisor, click the ellipsis (...), and click **Help**, Process Advisor opens the task documentation.

Example: `fullfile(pwd, "taskHelpFiles", "myTaskDocumentation.pdf")`

Data Types: string

RequiredIterationArtifactType — Artifact type that task can run on

`"sl_model_file"` (default) | ...

Type of artifact, specified as one or more of the values listed in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture

Category	Artifact Type	Description
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
"sl_test_suite_result"	Simulink Test suite result	

Example: "sl_model_file"

Example: ["sl_model_file" "zc_file"]

IterationQuery – Find artifacts that task iterates over

`padv.builtin.query.FindDesignModels` (default) | `padv.Query` object | name of `padv.Query` object

Query that finds the artifacts that the task iterates over, specified as a `padv.Query` object or the name of a `padv.Query` object. When you specify `IterationQuery`, the task runs one time *for each* artifact returned by the query. In the Process Advisor app, the artifacts returned by `IterationQuery` appear under task title.

For more information about task iterations, see “Overview of Process Model”.

Example: `padv.builtin.query.FindUnits`

InputDependencyQuery – Finds artifact dependencies for task inputs

`padv.Query` object | name of `padv.Query` object

Query that finds artifact dependencies for task inputs, specified as a `padv.Query` object or the name of a `padv.Query` object.

The build system runs the query specified by `InputDependencyQuery` to find the dependencies for the task inputs, since those dependencies can impact if task results are up-to-date.

For more information about task inputs, see “Overview of Process Model”.

Example: `padv.builtin.query.GetDependentArtifacts`

LaunchToolAction — Function that launches tool

`@launchToolAction` (default) | function handle

Function that launches a tool, specified as the function handle.

When you point to a task in the Process Advisor app, you can click the ellipsis (...) to see more options. For built-in tasks, you have the option to launch a tool associated with the task.

By default, the task `CollectMetrics` can launch the Model Maintainability Dashboard. If you specify the `Dashboard` property as a value other than `"ModelMaintainability"`, the task can launch the Model Testing Dashboard instead.

Data Types: `function_handle`

LaunchToolText — Description of action that LaunchToolAction property performs

`"Open Dashboard"` (default) | string

Description of the action that the `LaunchToolAction` property performs, specified as a string.

Data Types: `string`

InputQueries — Inputs to task

`padv.Query` object | name of `padv.Query` object | array of `padv.Query` objects

Inputs to the task, specified as:

- a `padv.Query` object
- the name of `padv.Query` object
- an array of `padv.Query` objects
- an array of names of `padv.Query` objects

By default, the task `CollectMetrics` gets the current model that the task is iterating over by using the built-in query `padv.builtin.query.GetIterationArtifact`.

OutputDirectory — Location for standard task outputs

`string(fullfile('$DEFAULTOUTPUTDIR$', '$ITERATIONARTIFACT$', 'metrics'))` (default)
| string

Location for standard task outputs, specified as a string.

The built-in tasks use tokens, like `$DEFAULTOUTPUTDIR$`, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Data Types: `string`

Metric Collection Options

Dashboard — Dashboard metrics to collect

`"ModelMaintainability"` (default) | `"ModelUnitPILTesting"` | `"ModelUnitSILTesting"` | `"ModelUnitTesting"`

Dashboard metrics to collect, specified as one of these values:

- "ModelMaintainability" — Analyze the size, architecture, and complexity of the MATLAB, Simulink, and Stateflow artifacts in your project by using the "Model Maintainability Metrics".
- "ModelUnitPILTesting" — Assess the quality and completeness of processor-in-the-loop (PIL) code testing by using the "Code Testing Metrics". Collecting these metrics requires a Simulink Test license.
- "ModelUnitSILTesting" — Assess the quality and completeness of software-in-the-loop (SIL) code testing by using the "Code Testing Metrics". Collecting these metrics requires a Simulink Test license.
- "ModelUnitTesting" — Assess the quality, traceability, and completeness of your models, requirements, tests, and test results by using the "Model Testing Metrics". By default, collecting these metrics requires a Requirements Toolbox™ license and Simulink Test license. If you do not want to collect requirements metrics, you can specify the property `IncludeRequirements` as `false`. When `IncludeRequirements` is `false`, the task does not require a Requirements Toolbox license.

The task uses this property to get the available metrics using the function `getAvailableMetricIds`.

Note If you specify a value other than "ModelMaintainability", make sure to specify the task iteration query as `padv.builtin.query.FindUnits` since you can only collect model testing and code testing metrics on units and not components.

Example: "ModelUnitTesting"

Installed — Filter metrics based on MathWorks® product installation status

1 (true) (default) | 0 (false)

Filter metrics based on whether the associated MathWorks product is installed, specified as either:

- 1 (true) — Only collect metrics associated with MathWorks products installed on the current machine.
- 0 (false) — Try to collect metrics for each of the available metrics, even if the associated MathWorks products are not installed on the current machine.

Example: `false`

Data Types: `logical`

IncludeRequirements — Include requirements metrics

1 (true) (default) | 0 (false)

Include requirements metrics in model testing metric results, specified as either:

- 1 (true) — If you specified the property `Dashboard` as "ModelUnitTesting", the task includes requirements metrics in the model testing metric results. Collecting requirements metrics requires a Requirements Toolbox license.
- 0 (false) — The task does not collect requirements metrics. The task excludes metrics where the metric ID contains the word `requirement` (case insensitive).

Example: `false`

Data Types: `logical`

ReportPath — Path to report output by task

`string(fullfile('$DEFAULTOUTPUTDIR$', '$ITERATIONARTIFACT$', 'metrics'))` (default)
| string

Path to report output by task, specified as a string.

The task generates the report by using the function `generateReport`.

Data Types: string

ReportName — Name of output report

`"$ITERATIONARTIFACT$_ModelMaintainability"` (default) | string

Name of output report, specified as a string.

Data Types: string

ReportFormat — Format of output report

`"pdf"` (default) | `"html-file"`

Format of output report, specified as either:

- `"pdf"` — PDF file.
- `"html-file"` — HTML report.

Example: `"html-file"`

Data Types: string

FilteredMetrics — List of metrics to filter out

`string.empty` (default) | string

List of metrics to filter out, specified as a string.

For example, if you are collecting model maintainability metrics (Dashboard property specified as `"ModelMaintainability"`), you can skip metric collection for a metric by specifying the value of `FilteredMetrics` as the metric ID for the metric.

Example: `"slcomp.ComponentInterfaceSignals"`

Data Types: string

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Collect metrics. Depending on the <code>Dashboard</code> property value, the task can collect either model maintainability metrics, model testing metrics, SIL code testing metrics, or PIL code testing metrics.</p> <hr/> <p>Note You do not need to manually invoke this method. When you run a task using the Process Advisor app or the <code>runprocess</code> function, the build system automatically invokes the <code>run</code> method for the task.</p> <hr/> <p>The <code>run</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this built-in task, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = run(obj, input) ... end</pre>
dryRun	<p>Dry run the task to validate task inputs and generate representative task outputs without actually running the task. The <code>dryRun</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this class, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = dryRun(obj, input) ... end</pre> <p>For more information, see “Dry Run Tasks to Test Process Model”.</p>
launchToolAction	<p>By default, the task launches the Model Maintainability Dashboard. If you specified the <code>Dashboard</code> property as a value other than “<code>ModelMaintainability</code>”, the task launches the Model Testing Dashboard instead. Process Advisor uses this method when you open the tool associated with a task.</p>

Examples

Collect Model Maintainability Metrics During Process

Add a task that can collect model maintainability metrics using the `metric.Engine` API for the Model Maintainability Dashboard.

Open the process model for your project. If you do not have a process model, open the Process Advisor app to automatically create a process model.

In the process model file, add the `CollectMetrics` task to your process model by using the `addTask` method. By default, the `CollectMetrics` task collects model maintainability metrics.

```
mmMetricTask = pm.addTask(padv.builtin.task.CollectMetrics());
```

You can reconfigure the task behavior by using the task properties. For example, to have the task return the generated metric results report as an HTML file instead of a PDF:

```
mmMetricTask.ReportFormat = "html-file";
```

Collect Model Testing and Code Testing Metrics During Process

By default, the `CollectMetrics` task collects model maintainability metrics. To collect different types of metrics, you can add multiple instances of the `CollectMetrics` to the process and reconfigure those instances to collect different metrics. For example, you can add tasks for model testing, SIL code testing, and PIL code testing metrics.

Each task instance needs a unique value for the `Name` property. To specify which metrics you want the task to collect, use the `Dashboard` property of the task. Since the dashboards collect model testing and code testing metrics for units, and not components, you need to specify the `IterationQuery` as `padv.builtin.query.FindUnits`. The other changes to the task property values give the task instances unique titles in Process Advisor and unique names for the reports that the task generates.

```
%% Collect Model Testing Metrics
mtMetricTask = pm.addTask(padv.builtin.task.CollectMetrics(...
    Name="ModelTestingMetrics",...
    IterationQuery=padv.builtin.query.FindUnits));
mtMetricTask.Title = "Collect Model Testing Metrics";
mtMetricTask.Dashboard = "ModelUnitTesting";
mtMetricTask.ReportName = "$ITERATIONARTIFACT$_ModelTesting";

%% Collect SIL Code Testing Metrics
stMetricTask = pm.addTask(padv.builtin.task.CollectMetrics(...
    Name="SILTestingMetrics",...
    IterationQuery=padv.builtin.query.FindUnits));
stMetricTask.Title = "Collect SIL Code Testing Metrics";
stMetricTask.Dashboard = "ModelUnitSILTesting";
stMetricTask.ReportName = "$ITERATIONARTIFACT$_SILTesting";

%% Collect PIL Code Testing Metrics
ptMetricTask = pm.addTask(padv.builtin.task.CollectMetrics(...
    Name="PILTestingMetrics",...
    IterationQuery=padv.builtin.query.FindUnits));
```

```
ptMetricTask.Title = "Collect PIL Code Testing Metrics";  
ptMetricTask.Dashboard = "ModelUnitPILTesting";  
ptMetricTask.ReportName = "$ITERATIONARTIFACT$_PILTesting";
```

You can use the other task properties to specify other task options.

To specify a preferred execution order for your tasks, you can use `runsAfter`. For example, if you want your process to merge test results before collecting model testing, SIL code testing, and PIL code testing metrics:

```
mtMetricTask.runsAfter(mergeTestTask);  
stMetricTask.runsAfter(mtMetricTask);  
ptMetricTask.runsAfter(stMetricTask);
```

See Also

`padv.builtin.task.GenerateCode` | `padv.builtin.task.MergeTestResults` |
`padv.builtin.task.RunModelStandards` | `padv.builtin.query.FindDesignModels` |
`padv.builtin.query.FindUnits` | Process Advisor | `runprocess`

Topics

[“Add Tasks to Process”](#)

[“Modify Default Process Model to Fit Your Process”](#)

[“Overview of Process Model”](#)

[“Explore Status and Quality of Testing Activities Using Model Testing Dashboard”](#)

[“Monitor Design Complexity Using Model Maintainability Dashboard”](#)

padv.builtin.task.DetectDesignErrors Class

Namespace: padv.builtin.task padv.builtin.task padv.builtin.task

Superclasses: padv.Task

Task for detecting design errors with Simulink Design Verifier

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.task.DetectDesignErrors` class provides a task that can detect design errors in your models by using Simulink Design Verifier™. Design error detection can identify dead logic, integer overflow, division by zero, and violations of design properties and assertions. By default, the `DetectDesignErrors` task outputs a design error detection report and data file.

You can add the task to your process model by using the method `addTask`. After you add the task to your process model, you can run the task from the Process Advisor app or by using the function `runprocess`.

To view the source code for this built-in task, in the MATLAB Command Window, enter:

```
open padv.builtin.task.DetectDesignErrors
```

The `padv.builtin.task.DetectDesignErrors` class is a `handle` class.

Creation

Description

`task = padv.builtin.task.DetectDesignErrors()` creates a task for detecting design errors with Simulink Design Verifier.

`task = padv.builtin.task.DetectDesignErrors(Name=Value)` sets certain properties using one or more name-value arguments. For example, `task = padv.builtin.task.DetectDesignErrors(Name = "MyDetectDesignErrors")` creates a task with the specified name.

You can use this syntax to set property values for `InputQueries`, `Name`, `IterationQuery`, `InputDependencyQuery`, `Licenses`, `LaunchToolAction`, and `LaunchToolText`.

The `padv.builtin.task.DetectDesignErrors` class also has other properties, but you cannot set those properties during task creation.

Properties

The `DetectDesignErrors` class inherits properties from `padv.Task`. The properties listed in “Specialized Inherited Properties” on page 7-0 are `padv.Task` properties that the `DetectDesignErrors` task overrides.

The task also has properties for specifying “Simulink Design Verifier Options” on page 7-0 for creating a design verification options object by using `sldvoptions` and generating a report by using `sldvreport`.

Specialized Inherited Properties

Name — Unique identifier for task in process

"padv.builtin.task.DetectDesignErrors" (default) | string

Unique identifier for task in process, specified as a string.

Example: "MyDetectDesignErrors"

Data Types: string

Title — Human-readable name that appears in Process Advisor app

"Detect Design Errors" (default) | string

Human-readable name that appears in Process Advisor app, specified as a string.

Example: "Detect Run-Time Errors"

Data Types: string

DescriptionText — Task description

"This task uses Simulink Design Verifier to detect design errors or dead logic for your models." (default) | string

Task description, specified as a string.

When you point to a task in Process Advisor and click the information icon, the tooltip shows the task description.

Example: "This task uses Simulink Design Verifier to detect design errors or dead logic for your models."

Data Types: string

DescriptionCSH — Path to task documentation

path to DetectDesignErrors documentation (default) | string

Path to task documentation, specified as a string.

When you point to a task in Process Advisor, click the ellipsis (...), and click **Help**, Process Advisor opens the task documentation.

Example: `fullfile(pwd,"taskHelpFiles","myTaskDocumentation.pdf")`

Data Types: string

RequiredIterationArtifactType — Artifact type that task can run on

"sl_model_file" (default) | ...

Type of artifact, specified as one or more of the values listed in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class

Category	Artifact Type	Description
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component

Category	Artifact Type	Description
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "sl_model_file"

Example: ["sl_model_file" "zc_file"]

IterationQuery – Find artifacts that task iterates over

padv.builtin.query.FindModels (default) | padv.Query object | name of padv.Query object

Query that finds the artifacts that the task iterates over, specified as a padv.Query object or the name of a padv.Query object. When you specify IterationQuery, the task runs one time *for each* artifact returned by the query. In the Process Advisor app, the artifacts returned by IterationQuery appear under task title.

For more information about task iterations, see “Overview of Process Model”.

InputDependencyQuery – Finds artifact dependencies for task inputs

padv.Query object | name of padv.Query object

Query that finds artifact dependencies for task inputs, specified as a padv.Query object or the name of a padv.Query object.

The build system runs the query specified by InputDependencyQuery to find the dependencies for the task inputs, since those dependencies can impact if task results are up-to-date.

For more information about task inputs, see “Overview of Process Model”.

Example: padv.builtin.query.GetDependentArtifacts

Licenses – List of licenses that task requires

["simulink_coverage" "simulink_design_verifier"] (default) | string

List of licenses that the task requires, specified as a string.

Data Types: `string`

LaunchToolAction — Function that launches tool

@launchToolAction (default) | function handle

Function that launches a tool, specified as the function handle.

When you point to a task in the Process Advisor app, you can click the ellipsis (...) to see more options. For built-in tasks, you have the option to launch a tool associated with the task.

For the task `DetectDesignErrors`, you can launch the Design Verifier app.

Data Types: `function_handle`

LaunchToolText — Description of action that LaunchToolAction property performs

"Open Design Verifier" (default) | string

Description of the action that the `LaunchToolAction` property performs, specified as a string.

Data Types: `string`

InputQueries — Inputs to task

padv.Query object | name of padv.Query object | array of padv.Query objects

Inputs to the task, specified as:

- a `padv.Query` object
- the name of `padv.Query` object
- an array of `padv.Query` objects
- an array of names of `padv.Query` objects

By default, the task `DetectDesignErrors` gets the current model that the task is iterating over by using the built-in query `padv.builtin.query.GetIterationArtifact`.

OutputDirectory — Location for standard task outputs

"\$DEFAULTOUTPUTDIR\$\\$ITERATIONARTIFACT\$\design_error_detections" (default) | string

Location for standard task outputs, specified as a string.

The built-in tasks use tokens, like `$DEFAULTOUTPUTDIR$`, as placeholders for dynamic path resolution during run-time. For more information, see "Dynamically Resolve Paths with Tokens".

Data Types: `string`

Simulink Design Verifier Options

DataFileName — Folder and or file name for analysis data

"\$ITERATIONARTIFACT\$_sldvdata" (default) | string

Folder and or file name for the MAT-file that contains the data generated during the analysis, specified as a string. The data is stored in an `sldvData` structure.

The built-in tasks use tokens, like `$ITERATIONARTIFACT$`, as placeholders for dynamic path resolution during run-time. For more information, see "Dynamically Resolve Paths with Tokens".

Example: "myModel_sldvdata"

Data Types: string

DesignMinMaxCheck — Check that intermediate and output signals in models are within range of specified minimum and maximum constraints

"off" (default) | "on"

Check that the intermediate and output signals in models are within the range of specified minimum and maximum constraints, specified as "on" or "off".

Example: "on"

DetectActiveLogic — Analyze models for active logic

"off" (default) | "on"

Analyze models for active logic, specified as "on" or "off". Note that this parameter is enabled only if DetectDeadLogic is "on".

Example: "on"

DetectBlockInputRangeViolations — Analyze models for block input range violations

"off" (default) | "on"

Analyze models for block input range violations, specified as "on" or "off".

Example: "on"

DetectDeadLogic — Analyze models for dead logic

"off" (default) | "on"

Analyze models for dead logic, specified as "on" or "off".

Example: "on"

DetectDivisionByZero — Analyze models for division-by-zero errors

"on" (default) | "off"

Analyze models for division-by-zero errors, specified as "on" or "off".

Example: "off"

DetectDSMAccessViolations — Analyze models for data store access violations

"off" (default) | "on"

Analyze models for data store access violations, specified as "on" or "off".

Example: "on"

DetectHISMViolationsHisl_0002 — Check usage of rem and reciprocal operations that cause non-finite results

"on" (default) | "off"

Check the usage of rem and reciprocal operations that cause non-finite results, specified as "on" or "off".

Example: "off"

DetectHISMViolationsHisl_0003 — Check usage of Square Root (Sqrt) operations with inputs that can be negative

"on" (default) | "off"

Check the usage of Square Root (Sqrt) operations with inputs that can be negative, specified as "on" or "off".

Example: "off"

DetectHISMViolationsHisl_0004 — Check usage of log and log10 operations that cause non-finite results

"on" (default) | "off"

Check the usage of log and log10 operations that cause non-finite results, specified as "on" or "off".

Example: "off"

DetectHISMViolationsHisl_0028 — Check usage of Reciprocal Square Root (rSqrt) blocks with inputs that can go zero or negative

"on" (default) | "off"

Check the usage of Reciprocal Square Root (rSqrt) blocks with inputs that can go zero or negative, specified as "on" or "off".

Example: "off"

DetectInfNaN — Analyze models for non-finite and NaN floating-point values

"off" (default) | "on"

Analyze models for non-finite and NaN floating-point values, specified as "on" or "off".

Example: "on"

DetectIntegerOverflow — Analyze models for integer and fixed-point data overflow errors

"on" (default) | "off"

Analyze models for integer and fixed-point data overflow errors, specified as "on" or "off".

Example: "off"

DetectOutOfBounds — Analyze models for out of bounds array access errors

"on" (default) | "off"

Analyze models for out of bounds array access errors, specified as "on" or "off".

Example: "off"

DetectSubnormal — Analyze models for subnormal floating-point values

"off" (default) | "on"

Analyze models for subnormal floating-point values, specified as "on" or "off".

Example: "on"

DisplayReport — Display report that Simulink Design Verifier generates

"off" (default) | "on"

After analysis, display the report that Simulink Design Verifier generates, specified as "on" or "off".

Example: "on"

GenerateReport — Generate design error detection report

true or 1 (default) | false or 0

Generate design error detection report, specified as a numeric or logical 1 (true) or 0 (false).

Example: false

Data Types: logical

MaxProcessTime — Maximum time (in seconds) that Simulink Design Verifier spends analyzing model

300 (default) | double

Maximum time (in seconds) that Simulink Design Verifier spends analyzing a model, specified as a double.

Example: 120

Data Types: double

Options — Options for generated report

["summary" "objectives"] (default) | "summary" | "objectives" | "objects"

Options for the generated report, specified as "summary", "objectives", "objects", or a combination of these options in an array.

Example: "summary"

ReportFormat — Format of generated report

"HTML" (default) | "PDF"

Format of the generated report, specified as either:

- "HTML" — HTML format
- "PDF" — PDF format

Example: "PDF"

ReportFilePath — Folder and or file name for analysis report

fullfile('\$DEFAULTOUTPUTDIR\$', '\$ITERATIONARTIFACT\$', 'design_error_detections', '\$ITERATIONARTIFACT\$_Design_Error_Detection_Report') (default) | string

Folder and or file name for the analysis report, specified as a string.

The built-in tasks use tokens, like \$DEFAULTOUTPUTDIR\$, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Example: "myModel_report"

Data Types: string

ShowUI — Display messages in log window

false or 0 (default) | true or 1

Display messages in the log window, specified as a numeric or logical 1 (true) or 0 (false). When ShowUI is specified as 0, messages appear in the MATLAB Command Window.

Example: true

Data Types: logical

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Detect design errors using Simulink Design Verifier</p> <hr/> <p>Note You do not need to manually invoke this method. When you run a task using the Process Advisor app or the <code>runprocess</code> function, the build system automatically invokes the <code>run</code> method for the task.</p> <hr/> <p>The <code>run</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this built-in task, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = run(obj, input) ... end</pre>
dryRun	<p>Dry run the task to validate task inputs and generate representative task outputs without actually running the task. The <code>dryRun</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this class, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = dryRun(obj, input) ... end</pre> <p>For more information, see “Dry Run Tasks to Test Process Model”.</p>
launchToolAction	<p>Launch the Design Verifier app. Process Advisor uses this method when you open the tool associated with a task.</p>

Examples

Add Design Verifier Task to Process

Add a task that can detect design errors by using Simulink Design Verifier.

Open the process model for your project. If you do not have a process model, open the Process Advisor app to automatically create a process model.

In the process model file, add the `DetectDesignErrors` task to your process model by using the `addTask` method.

```
dedObj = pm.addTask(padv.builtin.task.DetectDesignErrors);
```

You can reconfigure the task behavior by using the task properties. For example, to detect dead logic:

```
dedObj.DetectDeadLogic = "on";
```

See Also

Process Advisor | `runprocess`

Topics

“Add Tasks to Process”

“Modify Default Process Model to Fit Your Process”

“Overview of Process Model”

padv.builtin.task.FindClones Class

Namespace: padv.builtin.task padv.builtin.task padv.builtin.task

Superclasses: padv.Task

Task for detecting identical modeling patterns using Clone Detector

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.task.FindClones` class provides tasks that can identify clones in a model. Clones are modeling patterns with identical block types and connections. By default, it creates separate tasks to find clones across model and to find clones using the library in the Process Advisor app. When task provided by the class detects clones in the model, it generates a clone detection report in HTML format and saves the clone detection results in MAT files for further analysis.

You can add the task to your process model by using the method `addTask`. After you add the task to your process model, you can run the task from the Process Advisor app or by using the function `runprocess`.

To view the source code for this built-in task, in the MATLAB Command Window, enter:

```
open padv.builtin.task.FindClones
```

Note Generation of the clone detection report is supported in Simulink versions R2023b or later.

The `padv.builtin.task.FindClones` class is a handle class.

Creation

Description

`task = padv.builtin.task.FindClones` creates tasks for detecting identical modeling patterns using Clone Detector.

`task = padv.builtin.task.FindClones(Name=Value)` sets certain properties using one or more name-value arguments. For example, `task = padv.builtin.task.FindClones(Name="MyFindClonesTask")` creates a task with the specified name.

You can use this syntax to set property values for `Name`, `Title`, `InputQueries`, `InputDependencyQuery`, `IterationQuery`, `LaunchToolAction`, and `LaunchToolText`.

The `padv.builtin.task.FindClones` class also has other properties, but you cannot set those properties during task creation.

Properties

The FindClones class inherits properties from padv.Task. The properties listed in “Specialized Inherited Properties” on page 7-0 are padv.Task properties that the FindClones task overrides.

The task also has properties for specifying “Find Clone Options” on page 7-0 for specifying the clone detection task settings.

Specialized Inherited Properties

Name — Unique identifier for task in process

"padv.builtin.task.FindClones" (default) | string

Unique identifier for the task in the process, specified as a string.

Example: "MyFindClonesTask"

Data Types: string

Title — Human-readable name that appears in Process Advisor app

"Find Clones" (default) | "Find Clones Across Model" | "Find Library Clones" | string

Human-readable name that appears in the Process Advisor app, specified as a string.

Example: "My Find Clones Task"

Data Types: string

DescriptionText — Task description

"This task uses Clone Detector to detect clone patterns in your models."
(default) | string

Task description, specified as a string.

When you point to a task in Process Advisor and click the information icon, the tooltip shows the task description.

Example: "This task identifies clones in your models."

Data Types: string

DescriptionCSH — Path to task documentation

path to FindClones documentation (default) | string

Path to task documentation, specified as a string.

When you point to a task in Process Advisor, click the ellipsis (...), and click **Help**, Process Advisor opens the task documentation.

Example: fullfile(pwd, "taskHelpFiles", "myTaskDocumentation.pdf")

Data Types: string

InputQueries — Inputs to task

padv.builtin.query.FindArtifacts('ArtifactType', 'sl_library_file') (default) |
padv.Query object | name of padv.Query object | array of padv.Query objects

Inputs to the task, specified as:

- a `padv.Query` object
- the name of a `padv.Query` object
- an array of `padv.Query` objects
- an array of names of `padv.Query` objects

By default, the task `FindClones` gets the current model that the task is iterating over by using these built-in queries:

- `padv.builtin.query.GetIterationArtifact`
- `padv.builtin.query.FindArtifacts` on the task `padv.builtin.task.FindClones`

InputDependencyQuery — Query that finds artifact dependencies for task inputs

`padv.builtin.query.FindArtifacts` (default) | `padv.Query` object | name of `padv.Query` object

Query that finds artifact dependencies for task inputs, specified as a `padv.Query` object or the name of a `padv.Query` object.

The build system runs the query specified by `InputDependencyQuery` to find the dependencies for the task inputs, since those dependencies can impact if task results are up to date.

For more information about task inputs, see “Overview of Process Model”.

Example: `padv.builtin.query.FindArtifacts`

IterationQuery — Query that find artifacts that task iterates over

`padv.builtin.query.FindModels` (default) | `padv.Query` object | name of `padv.Query` object

Query that finds the artifacts that the task iterates over, specified as a `padv.Query` object or the name of a `padv.Query` object. When you specify `IterationQuery`, the task runs one time *for each* artifact returned by the query. In the Process Advisor app, the artifacts returned by `IterationQuery` appear under the task title.

For more information about task iterations, see “Overview of Process Model”.

Example: `padv.builtin.query.FindProjectFile`

LaunchToolAction — Function that launches tool

`@launchToolAction` (default) | function handle

Function that launches a tool, specified as the function handle.

When you point to a task in the Process Advisor app, you can click the ellipsis (...) to see more options. For built-in tasks, you have the option to launch a tool associated with the task.

By default, the task `FindClones` launches the Clone Detector app.

Data Types: `function_handle`

LaunchToolText — Description of action that LaunchToolAction property performs

"Open Find Clones Options" (default) | string

Description of the action that the `LaunchToolAction` property performs, specified as a string.

Data Types: string

RequiredIterationArtifactType – Artifact type that task can run on

"sl_model_file" (default) | ...

Type of artifact, specified as one or more of the values listed in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file

Category	Artifact Type	Description
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "sl_model_file"

Example: ["sl_model_file", "zc_file"]

OutputDirectory — Location for standard task outputs

fullfile('\$DEFAULTOUTPUTDIR\$', '\$ITERATIONARTIFACT\$', 'find_clones') (default) | string

Location for standard task outputs, specified as a string.

The built-in tasks use tokens, such as \$DEFAULTOUTPUTDIR\$, as placeholders for dynamic path resolution during run time. For more information, see "Dynamically Resolve Paths with Tokens".

Data Types: string

Find Clone Options

ReportPath — Path to generated report

fullfile('\$DEFAULTOUTPUTDIR\$', '\$ITERATIONARTIFACT\$', 'find_clones') (default) | string

Path to the generated report, specified as a string.

The built-in tasks use tokens, such as `$DEFAULTOUTPUTDIR`, as placeholders for dynamic path resolution during run time. For more information, see “Dynamically Resolve Paths with Tokens”.

Data Types: string

DetectLibraryClones — Option to detect clones from library

true or 1 (default) | false or 0

Option to detect clones from all the libraries in the project, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

TreatLibraryClonesAs — Outcome of library clone detection task

"Failure" (default) | "Warning"

Outcome when library clone detection task finds a clone, specified as "Failure" or "Warning".

Example: "Warning"

LibraryReportName — File name for generated library clone detection report

"\$ITERATIONARTIFACT\$_LibraryClonesReport" (default) | string

File name for the generated library clone detection report, specified as a string.

The built-in tasks use tokens, such as `$ITERATIONARTIFACT`, as placeholders for dynamic path resolution during run time. For more information, see “Dynamically Resolve Paths with Tokens”.

Example: "myLibraryCloneDetectionReport"

Data Types: string

LibraryResultFilename — MATLAB file name for generated library clone detection results

"\$ITERATIONARTIFACT\$_LibraryResults.mat" (default) | string

MATLAB file name for the generated library clone detection results, specified as a string.

The built-in tasks use tokens, such as `$ITERATIONARTIFACT`, as placeholders for dynamic path resolution during run time. For more information, see “Dynamically Resolve Paths with Tokens”.

Example: "myLibraryCloneDetectionResults.mat"

Data Types: string

DetectClonesAcrossModel — Option to detect clones across model

true or 1 (default) | false or 0

Option to detect clones across model, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

TreatAcrossModelClonesAs — Outcome of detection of clones across model task

"Warning" (default) | "Failure"

Outcome when detection of clones across model task finds a clone, specified as "Failure" or "Warning".

The built-in tasks use tokens, such as \$ITERATIONARTIFACT\$, as placeholders for dynamic path resolution during run time. For more information, see “Dynamically Resolve Paths with Tokens”.

Example: "Warning"

AcrossModelReportName — File name for generated detection of clones across model report

"\$ITERATIONARTIFACT\$_AcrossModelClonesReport" (default) | string

File name for the generated detection of clones across model report, specified as a string.

The built-in tasks use tokens, such as \$ITERATIONARTIFACT\$, as placeholders for dynamic path resolution during run time. For more information, see “Dynamically Resolve Paths with Tokens”.

Example: "myAcrossModelClonesReport"

Data Types: string

AcrossModelResultFilename — MATLAB file name for generated detection of clones across model results

"\$ITERATIONARTIFACT\$_AcrossModelResults.mat" (default) | string

MATLAB file name for the generated detection of clones across model results, specified as a string.

Example: "myAcrossModelClonesResults.mat"

Data Types: string

MinimumRegionSize — Minimum number of blocks

4 (default) | uint32

Minimum number of blocks in a region required to consider a pattern as a clone, specified as a uint32 integer.

This property applies only when DetectClonesAcrossModel is true.

Example: 10

Data Types: uint32

MinimumCloneGroupSize — Minimum number of clone pattern occurrences

3 (default) | uint32

Minimum number of clone pattern occurrences required to consider a pattern as a clone group, specified as a uint32 integer.

This property applies only when DetectClonesAcrossModel is true.

Example: 5

Data Types: uint32

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Find clones using Clone Detector app.</p> <hr/> <p>Note You do not need to manually invoke this method. When you run a task using the Process Advisor app or the <code>runprocess</code> function, the build system automatically invokes the <code>run</code> method for the task.</p> <hr/> <p>The <code>run</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this built-in task, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = run(obj, input) ... end</pre>
dryRun	<p>Dry run the task to validate task inputs and generate representative task outputs without actually running the task. The <code>dryRun</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this class, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = dryRun(obj, input) ... end</pre> <p>For more information, see “Dry Run Tasks to Test Process Model”.</p>
launchToolAction	<p>Launch the Clone Detector app. Process Advisor uses this method when you open the tool associated with a task.</p>

Examples

Add Find Clones Task to Process

Add a task that can identify clones for the models in your project.

Open the process model for your project. If you do not have a process model, open the Process Advisor app to automatically create a process model.

In the process model file, add the `FindClones` task to your process model by calling the `addTask` method on the `padv.ProcessModel` object `pm`.

```
findclonesTask = pm.addTask(padv.builtin.task.FindClones);
```

You can reconfigure the task behavior by using the task properties. For example, to change the size of minimum number of blocks in a region required to consider a pattern as a clone, set the task property `MinimumRegionSize`.

```
findclonesTask.MinimumRegionSize = "7";
```

When the task identifies clones in the model, it saves the clone data in the MAT files `ITERATIONARTIFACT_LibraryResults.mat` and `ITERATIONARTIFACT_AcrossModelResults.mat`, where `ITERATIONARTIFACT` is the input artifact for the task, inside the folder `OutputDirectory`. The clone information in these MAT files is similar to the `Simulink.CloneDetection.Results` object, which you can use to analyze the results of the `FindClones` task.

The task also creates another MAT file in the folder `m2m_ITERATIONARTIFACT` in the folder `OutputDirectory`. To highlight the clones in the model, navigate to the `OutputDirectory` folder that contains the folder `m2m_ITERATIONARTIFACT` and then open the Clone Detector app from the Process Advisor app. Clone Detector app uses the MAT file from this folder to highlight the clones. You can highlight a particular clone in the model by clicking on the hyperlinks in the Clone Detector app.

See Also

`addTask` | `padv.ProcessModel` | Process Advisor | `runprocess`

Topics

[“Add Tasks to Process”](#)

[“Modify Default Process Model to Fit Your Process”](#)

[“Overview of Process Model”](#)

[“Enable Component Reuse by Using Clone Detection”](#)

[“Find Clones Across the Model”](#)

padv.builtin.task.GenerateCode Class

Namespace: padv.builtin.task padv.builtin.task padv.builtin.task

Superclasses: padv.Task

Task for generating code with Embedded Coder

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.task.GenerateCode` class provides a task that can generate code by using Embedded Coder.

You can add the task to your process model by using the method `addTask`. After you add the task to your process model, you can run the task from the Process Advisor app or by using the function `runprocess`.

To view the source code for this built-in task, in the MATLAB Command Window, enter:

```
open padv.builtin.task.GenerateCode
```

The `padv.builtin.task.GenerateCode` class is a handle class.

Creation

Description

`task = padv.builtin.task.GenerateCode()` creates a task for generating code with Embedded Coder.

`task = padv.builtin.task.GenerateCode(Name=Value)` sets certain properties using one or more name-value arguments. For example, `task = padv.builtin.task.GenerateCode(Name = "MyCodeGenTask")` creates a task with the specified name.

You can use this syntax to set property values for `TreatAsRefModel`, `InputQueries`, `Name`, `Title`, `IterationQuery`, `InputDependencyQuery`, `DescriptionText`, `DescriptionCSH`, `Licenses`, `LaunchToolAction`, and `LaunchToolText`.

The `padv.builtin.task.GenerateCode` class also has other properties, but you cannot set those properties during task creation.

Properties

The `GenerateCode` class inherits properties from `padv.Task`. The properties listed in “Specialized Inherited Properties” on page 7-0 are `padv.Task` properties that the `GenerateCode` task overrides.

The task also has properties for specifying:

- “General Embedded Coder Options” on page 7-0 for generating code using the function `slbuild`. For more information on the `slbuild` arguments, see `slbuild`.
- “Advanced Caching and Parallel Code Generation Options” on page 7-0 .

Specialized Inherited Properties

Name — Unique identifier for task in process

`"padv.builtin.task.GenerateCode"` (default) | string

Unique identifier for task in process, specified as a string.

Example: `"MyCodeGenerationTask"`

Data Types: string

Title — Human-readable name that appears in Process Advisor app

`"Generate Code"` (default) | string

Human-readable name that appears in Process Advisor app, specified as a string.

Example: `"My Code Generation Task"`

Data Types: string

DescriptionText — Task description

`"This task uses Embedded Coder to generate code for models. By default, this task runs on all models in the project."` (default) | string

Task description, specified as a string.

When you point to a task in Process Advisor and click the information icon, the tooltip shows the task description.

Example: `"This task uses Embedded Coder to generate code for models. By default, this task runs on all models in the project."`

Data Types: string

DescriptionCSH — Path to task documentation

`path to GenerateCode documentation` (default) | string

Path to task documentation, specified as a string.

When you point to a task in Process Advisor, click the ellipsis (...), and click **Help**, Process Advisor opens the task documentation.

Example: `fullfile(pwd, "taskHelpFiles", "myTaskDocumentation.pdf")`

Data Types: string

RequiredIterationArtifactType — Artifact type that task can run on

`"sl_model_file"` (default) | ...

Type of artifact, specified as one or more of the values listed in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component

Category	Artifact Type	Description
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "sl_model_file"

Example: ["sl_model_file "zc_file"]

IterationQuery – Find artifacts that task iterates over

`padv.builtin.query.FindModels (default) | padv.Query object | name of padv.Query object`

Query that finds the artifacts that the task iterates over, specified as a `padv.Query` object or the name of a `padv.Query` object. When you specify `IterationQuery`, the task runs one time *for each* artifact returned by the query. In the Process Advisor app, the artifacts returned by `IterationQuery` appear under task title.

For more information about task iterations, see “Overview of Process Model”.

Example: `padv.builtin.query.FindProjectFile`

InputDependencyQuery – Finds artifact dependencies for task inputs

`padv.Query object | name of padv.Query object`

Query that finds artifact dependencies for task inputs, specified as a `padv.Query` object or the name of a `padv.Query` object.

The build system runs the query specified by `InputDependencyQuery` to find the dependencies for the task inputs, since those dependencies can impact if task results are up-to-date.

For more information about task inputs, see “Overview of Process Model”.

Example: `padv.builtin.query.GetDependentArtifacts`

Licenses — List of licenses that task requires

["matlab_coder" "real-time_workshop"] (default) | string

List of licenses that the task requires, specified as a string.

Data Types: string

LaunchToolAction — Function that launches tool

@launchToolAction (default) | function handle

Function that launches a tool, specified as the function handle.

When you point to a task in the Process Advisor app, you can click the ellipsis (...) to see more options. For built-in tasks, you have the option to launch a tool associated with the task.

For the task `GenerateCode`, you can launch the Embedded Coder app.

Data Types: function_handle

LaunchToolText — Description of action that LaunchToolAction property performs

"Open Embedded Coder" (default) | string

Description of the action that the `LaunchToolAction` property performs, specified as a string.

Data Types: string

InputQueries — Inputs to task

padv.Query object | name of padv.Query object | array of padv.Query objects

Inputs to the task, specified as:

- a `padv.Query` object
- the name of `padv.Query` object
- an array of `padv.Query` objects
- an array of names of `padv.Query` objects

By default, the task `GenerateCode` gets the current model that the task is iterating over by using the built-in query `padv.builtin.query.GetIterationArtifact`.

OutputDirectory — Location for standard task outputs

path to code generation folder (default) | string

Location for standard task outputs, specified as a string.

By default, the task `padv.builtin.task.GenerateCode` uses the path to the code generation folder specified by the parameter **CodeGenFolder**.

Data Types: string

CacheDirectory — Location for additional cache files

path to simulation cache folder (default) | string

Location for additional cache files that the task generates, specified as a string. The cache directory can contain temporary files that do not need to be either saved in the task results or archived by a CI system.

By default, the task `padv.builtin.task.GenerateCode` uses the path to the code generation folder specified by the parameter **CacheFolder**.

Data Types: `string`

General Embedded Coder Options

TreatAsRefModel — Setting that controls whether task generates reference model code or top model code

`[]` (default) | `true` or `1` | `false` or `0`

Setting that controls whether the task generates reference model code or top model code, specified as either:

- `[]` — Allow the task to automatically identify whether a model is a top model or a reference model before generating code.
- `1` (`true`) — Only generate reference model code.
- `0` (`false`) — Only generate top model code.

By default, the task automatically identifies whether a model is a top model or a reference model before generating code. But you can specify `TreatAsRefModel` as `true` or `false` if you want to override that behavior and only generate reference model code or top model code.

Example: `true`

Data Types: `logical`

GenerateCodeOnly — Generate code versus an executable file

`false` or `0` (default) | `true` or `1`

Generate code versus an executable file, specified as a numeric or logical `1` (`true`) or `0` (`false`).

By default, the task generates code only and does not build an executable file.

Example: `true`

Data Types: `logical`

ObfuscateCode — Generate obfuscated C code

`false` or `0` (default) | `true` or `1`

Generate obfuscated C code, specified as a numeric or logical `1` (`true`) or `0` (`false`).

Example: `true`

Data Types: `logical`

UpdateThisModelReferenceTarget — Conditional rebuild option for model reference build

`"IfOutOfDateOrStructuralChange"` (default) | `"Force"` | `"IfOutOfDate"`

Conditional rebuild option for model reference build, specified as either:

- `"Force"`
- `"IfOutOfDateOrStructuralChange"`
- `"IfOutOfDate"`

Example: `"IfOutOfDate"`

ForceTopModelBuild — Force top model of model hierarchy to build

false or 0 (default) | true or 1

Force top model of model hierarchy to build, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

IncludeModelReferenceSimulationTargets — Build model reference simulation targets

false or 0 (default) | true or 1

Build model reference simulation targets, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

Advanced Caching and Parallel Code Generation Options**GenerateExternalCodeCache — Setting to collect only SLXC files as task outputs**

false or 0 (default) | true or 1

Setting to collect only SLXC files as task outputs, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

ExternalCodeCacheDirectory — Location to save SLXC filefullfile('\$DEFAULTOUTPUTDIR\$', '\$ITERATIONARTIFACT\$', "external_code_cache")
(default) | string

Location to save SLXC file, specified as a string.

The built-in tasks use tokens, like \$DEFAULTOUTPUTDIR\$, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Data Types: string

TrackAllGeneratedCode — Track all code files

false or 0 (default) | true or 1

Track all code files, not just model.c and model.h files, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

Methods**Specialized Public Methods**

This class overrides the following inherited methods.

run	<p>Generate code using Embedded Coder</p> <hr/> <p>Note You do not need to manually invoke this method. When you run a task using the Process Advisor app or the <code>runprocess</code> function, the build system automatically invokes the <code>run</code> method for the task.</p> <hr/> <p>The <code>run</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this built-in task, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = run(obj, input) ... end</pre>
dryRun	<p>Dry run the task to validate task inputs and generate representative task outputs without actually running the task. The <code>dryRun</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this class, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = dryRun(obj, input) ... end</pre> <p>For more information, see “Dry Run Tasks to Test Process Model”.</p>
launchToolAction	<p>Launch the Embedded Coder app. Process Advisor uses this method when you open the tool associated with a task.</p>

Examples

Add Code Generation Task to Process

Add a task that can generate code for each model in the project using Embedded Coder.

Open the process model for your project. If you do not have a process model, open the Process Advisor app to automatically create a process model.

In the process model file, add the `GenerateCode` task to your process model by using the `addTask` method.

```
codegenTask = pm.addTask(padv.builtin.task.GenerateCode);
```

You can reconfigure the task behavior by using the task properties. For example, to rebuild models if the build process detects changes in known dependencies of the model:

```
codegenTask.UpdateThisModelReferenceTarget = 'IfOutOfDate';
```

See Also

`addTask` | `padv.builtin.task.AnalyzeModelCode` |
`padv.builtin.task.RunCodeInspection` | `padv.builtin.query.FindCodeForModel` |
`padv.ProcessModel` | `Process Advisor` | `runprocess`

Topics

“Add Tasks to Process”

padv.builtin.task.GenerateModelComparison Class

Namespace: padv.builtin.task padv.builtin.task padv.builtin.task

Superclasses: padv.Task

Task for generating model comparison report

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.task.GenerateModelComparison` class provides a task that can compare models in the project to their ancestors in Git and publish a comparison report using the Comparison Tool. The task compares your version of the model to either the latest or previous version on the main branch in Git:

- If you make a change to a model and run the task, the task compares your version of the model to either the head of the current branch or latest version on the main branch in Git.
- If you do not make changes to a model and then run the task, the task compares the model to the previous version available on the main branch in Git.

You can use the task properties to specify different report options, filtering options, and the name of the Git branch used for the comparison.

You can add the task to your process model by using the method `addTask`. After you add the task to your process model, you can run the task from the Process Advisor app or by using the function `runprocess`.

To view the source code for this built-in task, in the MATLAB Command Window, enter:

```
open padv.builtin.task.GenerateModelComparison
```

The `padv.builtin.task.GenerateModelComparison` class is a `handle` class.

Creation

Description

`task = padv.builtin.task.GenerateModelComparison()` creates a task for comparing models using the Comparison Tool.

`task = padv.builtin.task.GenerateModelComparison(Name=Value)` sets certain properties using one or more name-value arguments. For example, `task = padv.builtin.task.GenerateModelComparison(Name = "MyModelComparisonTask")` creates a task with the specified name.

You can use this syntax to set property values for `InputQueries`, `Name`, `IterationQuery`, `InputDependencyQuery`, `Licenses`, `LaunchToolAction`, and `LaunchToolText`.

The `padv.builtin.task.GenerateModelComparison` class also has other properties, but you cannot set those properties during task creation.

Properties

The `GenerateModelComparison` class inherits properties from `padv.Task`. The properties listed in “General Task Properties” on page 7-0 are `padv.Task` properties that the `GenerateModelComparison` task overrides.

The task also has properties for specifying:

- “Comparison Options” on page 7-0 for comparing models by using `visdiff`
- “Report Options” on page 7-0

General Task Properties

Name — Unique identifier for task in process

`"padv.builtin.task.GenerateModelComparison"` (default) | string

Unique identifier for task in process, specified as a string.

Example: `"MyModelDiffTask"`

Data Types: string

Title — Human-readable name that appears in Process Advisor app

`"Generate Model Comparison"` (default) | string

Human-readable name that appears in Process Advisor app, specified as a string.

Example: `"My Generate Model Comparison Task"`

Data Types: string

DescriptionText — Task description

`"This task uses the Model Comparison tool to generate a difference report with the previous version of the model."` (default) | string

Task description, specified as a string.

When you point to a task in Process Advisor and click the information icon, the tooltip shows the task description.

Example: `"This task uses the Model Comparison tool to generate a difference report with the previous version of the model."`

Data Types: string

DescriptionCSH — Path to task documentation

`path to GenerateModelComparison documentation` (default) | string

Path to task documentation, specified as a string.

When you point to a task in Process Advisor, click the ellipsis (...), and click **Help**, Process Advisor opens the task documentation.

Example: `fullfile(pwd, "taskHelpFiles", "myTaskDocumentation.pdf")`

Data Types: string

RequiredIterationArtifactType – Artifact type that task can run on

"sl_model_file" (default) | ...

Type of artifact, specified as one or more of the values listed in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file

Category	Artifact Type	Description
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "sl_model_file"

Example: ["sl_model_file" "zc_file"]

IterationQuery – Find artifacts that task iterates over

padv.builtin.query.FindModels (default) | padv.Query object | name of padv.Query object

Query that finds the artifacts that the task iterates over, specified as a padv.Query object or the name of a padv.Query object. When you specify IterationQuery, the task runs one time *for each* artifact returned by the query. In the Process Advisor app, the artifacts returned by IterationQuery appear under task title.

For more information about task iterations, see “Overview of Process Model”.

InputDependencyQuery – Finds artifact dependencies for task inputs

padv.Query object | name of padv.Query object

Query that finds artifact dependencies for task inputs, specified as a padv.Query object or the name of a padv.Query object.

The build system runs the query specified by `InputDependencyQuery` to find the dependencies for the task inputs, since those dependencies can impact if task results are up-to-date.

For more information about task inputs, see “Overview of Process Model”.

Example: `padv.builtin.query.GetDependentArtifacts`

LaunchToolAction — Function that launches tool

`@launchToolAction (default) | function handle`

Function that launches a tool, specified as the function handle.

When you point to a task in the Process Advisor app, you can click the ellipsis (...) to see more options. For built-in tasks, you have the option to launch a tool associated with the task.

For the task `GenerateModelComparison`, you can launch the Model Comparison tool.

Data Types: `function_handle`

LaunchToolText — Description of action that LaunchToolAction property performs

`"Compare to Ancestor" (default) | string`

Description of the action that the `LaunchToolAction` property performs, specified as a string.

Data Types: `string`

InputQueries — Inputs to task

`padv.Query object | name of padv.Query object | array of padv.Query objects`

Inputs to the task, specified as:

- a `padv.Query` object
- the name of `padv.Query` object
- an array of `padv.Query` objects
- an array of names of `padv.Query` objects

By default, the task `GenerateModelComparison` gets the current model that the task is iterating over by using the built-in query `padv.builtin.query.GetIterationArtifact`.

OutputDirectory — Location for standard task outputs

`fullfile('$DEFAULTOUTPUTDIR$', '$ITERATIONARTIFACT$', 'model_comparison') (default) | string`

Location for standard task outputs, specified as a string.

The built-in tasks use tokens, like `$DEFAULTOUTPUTDIR$`, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Data Types: `string`

Comparison Options

Filter — Setting for filtering model comparison report

`"default" (default) | "unfiltered"`

Setting for filtering model comparison report, specified as either:

- "unfiltered" — Removes all filtering from the comparison.
- "default" — Default filtering for the comparison, which hides non-functional changes.

Example: "unfiltered"

MainBranch — Name of Git branch used for comparison

"main" (default) | string

Name of Git branch used for comparison, specified as a string.

Example: "taskBranch"

Data Types: string

Report Options

ReportName — Name of generated comparison report

"\$ITERATIONARTIFACT\$_Model_Comparison" (default) | string

Name of generated comparison report, specified as a string.

The built-in tasks use tokens, like \$ITERATIONARTIFACT\$, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Example: "myModel_Model_Comparison"

Data Types: string

ReportPath — Path to generated comparison report

fullfile('\$DEFAULTOUTPUTDIR\$', '\$ITERATIONARTIFACT\$', 'model_comparison')
(default) | string

Path to generated comparison report, specified as a string.

The built-in tasks use tokens, like \$DEFAULTOUTPUTDIR\$, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Data Types: string

ReportFormat — Format of generated comparison report

"HTML" (default) | "DOCX" | "PDF"

Format of generated comparison report, specified as either "DOCX", "HTML", or "PDF".

Example: "PDF"

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Generate a model comparison report using the Comparison Tool</p> <hr/> <p>Note You do not need to manually invoke this method. When you run a task using the Process Advisor app or the <code>runprocess</code> function, the build system automatically invokes the <code>run</code> method for the task.</p> <hr/> <p>The <code>run</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this built-in task, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = run(obj, input) ... end</pre>
dryRun	<p>Dry run the task to validate task inputs and generate representative task outputs without actually running the task. The <code>dryRun</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this class, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = dryRun(obj, input) ... end</pre> <p>For more information, see “Dry Run Tasks to Test Process Model”.</p>
launchToolAction	<p>Launch the Comparison Tool. Process Advisor uses this method when you open the tool associated with a task.</p>

Examples

Add Model Comparison Task to Process

Add a task that can generate a model comparison report using the Comparison Tool.

Open the process model for your project. If you do not have a process model, open the Process Advisor app to automatically create a process model.

In the process model file, add the `GenerateModelComparison` task to your process model by using the `addTask` method.

```
mdlCompTask = pm.addTask(padv.builtin.task.GenerateModelComparison);
```

You can reconfigure the task behavior by using the task properties. For example, to use a different branch for the comparison:

```
mdlCompTask.MainBranch = "branchname";
```

Tips

- To find and compare model ancestors, this task requires that you use Git source control for your project. For information on how to add a project to Git source control, see “Use Source Control with MATLAB Projects”.
- If you run MATLAB using the `-nodisplay` option or you use a machine that does not have a display (like many CI runners and Docker containers), you should set up a virtual display server before you include this task in your process model. For information, see “Set Up Virtual Display for No-Display Machine” in the User's Guide.

See Also

`addTask` | `padv.ProcessModel` | Process Advisor | `runprocess` | `visdiff`

Topics

“Add Tasks to Process”

“Modify Default Process Model to Fit Your Process”

“Overview of Process Model”

“Review Changes in Simulink Models”

padv.builtin.task.GenerateSDDReport Class

Namespace: padv.builtin.task padv.builtin.task padv.builtin.task

Superclasses: padv.Task

Task for generating a System Design Description (SDD) report

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.task.GenerateSDDReport` class provides a task that can generate a System Design Description (SDD) report from a predefined template using Simulink Report Generator™. The System Design Description report provides a summary or detailed information about a system design represented by a model.

You can add the task to your process model by using the method `addTask`. After you add the task to your process model, you can run the task from the Process Advisor app or by using the function `runprocess`.

To view the source code for this built-in task, in the MATLAB Command Window, enter:

```
open padv.builtin.task.GenerateSDDReport
```

The `padv.builtin.task.GenerateSDDReport` class is a `handle` class.

Note If you are using the pipeline generator, `padv.pipeline.generatePipeline`, this task is not supported for the parallel pipeline architecture `IndependentModelPipelines`.

Creation

Description

`task = padv.builtin.task.GenerateSDDReport()` creates a task for generating a System Design Description (SDD) report using Simulink Report Generator.

`task = padv.builtin.task.GenerateSDDReport(Name=Value)` sets certain properties using one or more name-value arguments. For example, `task = padv.builtin.task.GenerateSDDReport(Name = "MySDDReportTask")` creates a task with the specified name.

You can use this syntax to set property values for `InputQueries`, `Name`, `IterationQuery`, `InputDependencyQuery`, `Licenses`, `LaunchToolAction`, and `LaunchToolText`.

The `padv.builtin.task.GenerateSDDReport` class also has other properties, but you cannot set those properties during task creation.

Properties

The `GenerateSDDReport` class inherits properties from `padv.Task`. The properties listed in “Specialized Inherited Properties” on page 7-0 are `padv.Task` properties that the `GenerateSDDReport` task overrides.

The task also has properties for specifying “SDD Report Options” on page 7-0 for specifying the report options for an SDD object.

Specialized Inherited Properties

Name — Unique identifier for task in process

`"padv.builtin.task.GenerateSDDReport"` (default) | string

Unique identifier for task in process, specified as a string.

Example: `"MyGenerateSDDReportTask"`

Data Types: string

Title — Human-readable name that appears in Process Advisor app

`"Generate SDD Report"` (default) | string

Human-readable name that appears in Process Advisor app, specified as a string.

Example: `"My Generate SDD Report Task"`

Data Types: string

DescriptionText — Task description

`"This task uses Simulink Report Generator to create a System Design Description report for your models."` (default) | string

Task description, specified as a string.

When you point to a task in Process Advisor and click the information icon, the tooltip shows the task description.

Example: `"This task uses Simulink Report Generator to create a System Design Description report for your models."`

Data Types: string

DescriptionCSH — Path to task documentation

`path to GenerateSDDReport documentation` (default) | string

Path to task documentation, specified as a string.

When you point to a task in Process Advisor, click the ellipsis (...), and click **Help**, Process Advisor opens the task documentation.

Example: `fullfile(pwd, "taskHelpFiles", "myTaskDocumentation.pdf")`

Data Types: string

RequiredIterationArtifactType — Artifact type that task can run on

`"sl_model_file"` (default) | ...

Type of artifact, specified as one or more of the values listed in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture

Category	Artifact Type	Description
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
"sl_test_suite_result"	Simulink Test suite result	

Example: "sl_model_file"

Example: ["sl_model_file "zc_file"]

IterationQuery – Find artifacts that task iterates over

`padv.builtin.query.FindModels (default) | padv.Query object | name of padv.Query object`

Query that finds the artifacts that the task iterates over, specified as a `padv.Query` object or the name of a `padv.Query` object. When you specify `IterationQuery`, the task runs one time *for each* artifact returned by the query. In the Process Advisor app, the artifacts returned by `IterationQuery` appear under task title.

For more information about task iterations, see “Overview of Process Model”.

InputDependencyQuery – Finds artifact dependencies for task inputs

`padv.Query object | name of padv.Query object`

Query that finds artifact dependencies for task inputs, specified as a `padv.Query` object or the name of a `padv.Query` object.

The build system runs the query specified by `InputDependencyQuery` to find the dependencies for the task inputs, since those dependencies can impact if task results are up-to-date.

For more information about task inputs, see “Overview of Process Model”.

Example: `padv.builtin.query.GetDependentArtifacts`

Licenses — List of licenses that task requires

["matlab_report_gen" "simulink_report_gen"] (default) | string

List of licenses that the task requires, specified as a string.

Data Types: string

LaunchToolAction — Function that launches tool

@launchToolAction (default) | function handle

Function that launches a tool, specified as the function handle.

When you point to a task in the Process Advisor app, you can click the ellipsis (...) to see more options. For built-in tasks, you have the option to launch a tool associated with the task.

For the task `GenerateSDDReport`, you can launch a Report Options dialog.

Data Types: function_handle

LaunchToolText — Description of action that LaunchToolAction property performs

"Open SDD Report Options" (default) | string

Description of the action that the `LaunchToolAction` property performs, specified as a string.

Data Types: string

InputQueries — Inputs to task

padv.Query object | name of padv.Query object | array of padv.Query objects

Inputs to the task, specified as:

- a `padv.Query` object
- the name of `padv.Query` object
- an array of `padv.Query` objects
- an array of names of `padv.Query` objects

By default, the task `GenerateSDDReport` gets the current model that the task is iterating over by using the built-in query `padv.builtin.query.GetIterationArtifact`.

OutputDirectory — Location for standard task outputsfullfile('\$DEFAULTOUTPUTDIR\$', '\$ITERATIONARTIFACT\$',
'system_design_description') (default) | string

Location for standard task outputs, specified as a string.

The built-in tasks use tokens, like `$DEFAULTOUTPUTDIR$`, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Data Types: string

SDD Report Options**DisplayReport — Open generated report**

false or 0 (default) | true or 1

Open the generated report, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

IncludeCustomLibraries — Include custom libraries in design description

false or 0 (default) | true or 1

Include custom libraries in the design description, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

IncludeDetails — Include design details in design description

true or 1 (default) | false or 0

Include design details, like block parameters, in the design description, specified as a numeric or logical 1 (true) or 0 (false).

Example: false

Data Types: logical

IncludeGlossary — Include glossary in design description

true or 1 (default) | false or 0

Include a glossary in the design description, specified as a numeric or logical 1 (true) or 0 (false).

Example: false

Data Types: logical

IncludeLookupTables — Include lookup tables in design description

true or 1 (default) | false or 0

Include lookup tables in the design description, specified as a numeric or logical 1 (true) or 0 (false).

Example: false

Data Types: logical

IncludeModelRefs — Include model references in design description

false or 0 (default) | true or 1

Include model references in the design description, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

IncludeRequirementsLinks — Include requirement links in design description

true or 1 (default) | false or 0

Include requirement links in the design description, specified as a numeric or logical 1 (true) or 0 (false).

Example: false

Data Types: `logical`

IncrOutputName — Increment report name to avoid overwriting existing report

`false` or `0` (default)

Do not specify this property.

Data Types: `logical`

LegalNotice — Legal notice that appears on report

`"For Internal Distribution Only"` (default) | `string`

Legal notice that appears on the report, specified as a string.

Example: `"Confidential"`

Data Types: `string`

PackageType — File packaging type for HTML reports

`1` (default)

The SDD report does not use this property.

ReportFormat — Output format for generated report

`"html"` (default) | `"pdf"` | `"docx"`

Output format for the generated report, specified as either:

- `"html"` — HTML format. You can use the property `PackageType` to specify whether report files are zipped, unzipped, or produce both zipped and unzipped files.
- `"pdf"` — PDF format
- `"docx"` — Microsoft Word document format

Example: `"pdf"`

ReportName — File name for generated report

`"$ITERATIONARTIFACT$_SDD"` (default) | `string`

File name for the generated report, specified as a string.

The built-in tasks use tokens, like `$ITERATIONARTIFACT$`, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Example: `"mySDDReport"`

Data Types: `string`

ReportPath — Path to generated report

`fullfile('$DEFAULTOUTPUTDIR$', '$ITERATIONARTIFACT$', 'system_design_description')` (default) | `string`

Path to the generated report, specified as a string.

The built-in tasks use tokens, like `$DEFAULTOUTPUTDIR$`, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Data Types: `string`

ReportTitle – Title of report

"" (default) | string

Title of the report, specified as a string.

Data Types: string

TitleImagePath – Path of image that appears on report title page

"" (default) | string

Path of image that appears on report title page, specified as a string.

Data Types: string

Subtitle – Subtitle of report

"Design Description" (default) | string

Subtitle of the report, specified as a string.

Data Types: string

TimeFormat – Date and time format for report creation date

"" (default) | "dd-mmm-yyyy" | "HH:MM:SS"

Date and time format for report creation date, specified as one of the values listed in this table.

Value	Example Output
"dd-mmm-yyyy HH:MM:SS"	26-Jul-2021 17:10:21
"dd-mmm-yyyy"	26-Jul-2021
"mm/dd/yy"	07/26/21
"mmm"	Jul
"m"	J
"mm"	07
"mm/dd"	07/26
"dd"	26
"ddd"	Mon
"d"	M
"yyyy"	2021
"yy"	21
"mmyy"	Jul21
"HH:MM:SS"	17:10:21
"HH:MM:SS PM"	5:10:21 PM
"HH:MM"	17:10
"HH:MM PM"	5:10 PM
"QQ-YY"	Q3-21
"QQ"	Q3

By default, if you do not specify a value for `TimeFormat`, the SDD report uses "dd-mmm-yyyy HH:MM:SS". The date and time appear next to **Published** on the title page of the report.

Example: "dd-mmm-yyyy"

Data Types: `string`

UseStatusWindow — Display report generation status messages in separate window

`false` or `0` (default) | `true` or `1`

Display report generation status messages in separate window, specified as a numeric or logical 1 (`true`) or 0 (`false`).

Example: `true`

Data Types: `logical`

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Generate a System Design Description (SDD) report using Simulink Report Generator</p> <hr/> <p>Note You do not need to manually invoke this method. When you run a task using the Process Advisor app or the <code>runprocess</code> function, the build system automatically invokes the <code>run</code> method for the task.</p> <hr/> <p>The <code>run</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this built-in task, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = run(obj, input) ... end</pre>
-----	--

dryRun	<p>Dry run the task to validate task inputs and generate representative task outputs without actually running the task. The <code>dryRun</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this class, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = dryRun(obj, input) ... end</pre> <p>For more information, see “Dry Run Tasks to Test Process Model”.</p>
launchToolAction	<p>Launch Report Options dialog. Process Advisor uses this method when you open the tool associated with a task.</p>

Examples

Add SDD Report Generation Task to Process

Add a task that can generate an SDD report for the models in your project.

Open the process model for your project. If you do not have a process model, open the Process Advisor app to automatically create a process model.

In the process model file, add the `GenerateSDDReport` task to your process model by calling the `addTask` method on the `padv.ProcessModel` object `pm`.

```
sddTask = pm.addTask(padv.builtin.task.GenerateSDDReport);
```

You can reconfigure the task behavior by using the task properties. For example, to generate a PDF instead of HTML, set the task property `ReportFormat` to "pdf".

```
sddTask.ReportFormat = "pdf";
```

See Also

`addTask` | `padv.ProcessModel` | Process Advisor | `runprocess`

Topics

“Add Tasks to Process”

“Modify Default Process Model to Fit Your Process”

“Overview of Process Model”

“System Design Description” (Simulink Report Generator)

padv.builtin.task.GenerateSimulinkWebView Class

Namespace: padv.builtin.task padv.builtin.task padv.builtin.task

Superclasses: padv.Task

Task for creating web views for models

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.task.GenerateSimulinkWebView` class provides a task that can create web views for your models using Simulink Report Generator. You can view, navigate, and share a web view without a Simulink license.

You can add the task to your process model by using the method `addTask`. After you add the task to your process model, you can run the task from the Process Advisor app or by using the function `runprocess`.

To view the source code for this built-in task, in the MATLAB Command Window, enter:

```
open padv.builtin.task.GenerateSimulinkWebView
```

The `padv.builtin.task.GenerateSimulinkWebView` class is a `handle` class.

Creation

Description

`task = padv.builtin.task.GenerateSimulinkWebView()` creates a task for creating web views using Simulink Report Generator.

`task = padv.builtin.task.GenerateSimulinkWebView(Name=Value)` sets certain properties using one or more name-value arguments. For example, `task = padv.builtin.task.GenerateSimulinkWebView(Name = "MyWebViewTask")` creates a task with the specified name.

You can use this syntax to set property values for `InputQueries`, `Name`, `IterationQuery`, `InputDependencyQuery`, `Licenses`, `LaunchToolAction`, and `LaunchToolText`.

The `padv.builtin.task.GenerateSimulinkWebView` class also has other properties, but you cannot set those properties during task creation.

Properties

The `GenerateSimulinkWebView` class inherits properties from `padv.Task`. The properties listed in “Specialized Inherited Properties” on page 7-0 are `padv.Task` properties that the `GenerateSimulinkWebView` task overrides.

The task also has properties for specifying “Web View Options” on page 7-0 for exporting Simulink models to web views by using `slwebview`.

Specialized Inherited Properties

Name — Unique identifier for task in process

"padv.builtin.task.GenerateSimulinkWebView" (default) | string

Unique identifier for task in process, specified as a string.

Example: "MyWebViewTask"

Data Types: string

Title — Human-readable name that appears in Process Advisor app

"Generate Simulink Web View" (default) | string

Human-readable name that appears in Process Advisor app, specified as a string.

Example: "My Web View Task"

Data Types: string

DescriptionText — Task description

"This task uses Simulink Report Generator to create a web view for your models." (default) | string

Task description, specified as a string.

When you point to a task in Process Advisor and click the information icon, the tooltip shows the task description.

Example: "This task uses Simulink Report Generator to create a web view for your models."

Data Types: string

DescriptionCSH — Path to task documentation

path to GenerateSimulinkWebView documentation (default) | string

Path to task documentation, specified as a string.

When you point to a task in Process Advisor, click the ellipsis (...), and click **Help**, Process Advisor opens the task documentation.

Example: `fullfile(pwd, "taskHelpFiles", "myTaskDocumentation.pdf")`

Data Types: string

RequiredIterationArtifactType — Artifact type that task can run on

"sl_model_file" (default) | ...

Type of artifact, specified as one or more of the values listed in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class

Category	Artifact Type	Description
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component

Category	Artifact Type	Description
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "sl_model_file"

Example: ["sl_model_file" "zc_file"]

IterationQuery – Find artifacts that task iterates over

`padv.builtin.query.FindModels` (default) | `padv.Query` object | name of `padv.Query` object

Query that finds the artifacts that the task iterates over, specified as a `padv.Query` object or the name of a `padv.Query` object. When you specify `IterationQuery`, the task runs one time *for each* artifact returned by the query. In the Process Advisor app, the artifacts returned by `IterationQuery` appear under task title.

For more information about task iterations, see “Overview of Process Model”.

InputDependencyQuery – Finds artifact dependencies for task inputs

`padv.Query` object | name of `padv.Query` object

Query that finds artifact dependencies for task inputs, specified as a `padv.Query` object or the name of a `padv.Query` object.

The build system runs the query specified by `InputDependencyQuery` to find the dependencies for the task inputs, since those dependencies can impact if task results are up-to-date.

For more information about task inputs, see “Overview of Process Model”.

Example: `padv.builtin.query.GetDependentArtifacts`

Licenses – List of licenses that task requires

["matlab_report_gen" "simulink_report_gen"] (default) | string

List of licenses that the task requires, specified as a string.

Data Types: `string`

LaunchToolAction — Function that launches tool

`@launchToolAction (default) | function handle`

Function that launches a tool, specified as the function handle.

When you point to a task in the Process Advisor app, you can click the ellipsis (...) to see more options. For built-in tasks, you have the option to launch a tool associated with the task.

For the task `GenerateSimulinkWebView`, you can launch a web view options dialog.

Data Types: `function_handle`

LaunchToolText — Description of action that LaunchToolAction property performs

`"Open Web View Options" (default) | string`

Description of the action that the `LaunchToolAction` property performs, specified as a string.

Data Types: `string`

InputQueries — Inputs to task

`padv.Query object | name of padv.Query object | array of padv.Query objects`

Inputs to the task, specified as:

- a `padv.Query` object
- the name of `padv.Query` object
- an array of `padv.Query` objects
- an array of names of `padv.Query` objects

By default, the task `GenerateSimulinkWebView` gets the current model that the task is iterating over by using the built-in query `padv.builtin.query.GetIterationArtifact`.

OutputDirectory — Location for standard task outputs

`fullfile('$DEFAULTOUTPUTDIR$', '$ITERATIONARTIFACT$', 'webview') (default) | string`

Location for standard task outputs, specified as a string.

The built-in tasks use tokens, like `$DEFAULTOUTPUTDIR$`, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Data Types: `string`

Web View Options

FollowLinks — Follow links into library blocks

`1 (default) | 0`

Follow links into library blocks, specified as either:

- `0` — Does not allow you to follow links into library blocks in a web view
- `1` — Allows you to follow links into library blocks in a web view

Example: `0`

FollowModelReference — Access referenced models in web view

1 (default) | 0

Access referenced models in a web view, specified as either:

- 0 — Does not allow you to access referenced models in a web view
- 1 — Allows you to access referenced models in a web view

Example: 0

IncludeNotes — Include user notes in web view

true or 1 (default) | false or 0

Include user notes in web view, specified as a numeric or logical 1 (true) or 0 (false).

Example: false

Data Types: logical

IncrementalExport — Export models incrementally

false or 0 (default) | true or 1

Export models incrementally, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

LookUnderMasks — Export ability to interact with masked blocks

"All" (default) | "None"

Export the ability to interact with masked blocks, specified as either "None" or "All".

Example: "None"

PackagingType — Type of web view output package

"unzipped" (default) | "zipped" | "both"

Type of web view output package, specified as "zipped", "unzipped", or "both".

Example: "zipped"

RecurseFolder — Export models in subfolders

false or 0 (default) | true or 1

Export models in subfolders, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

ReportName — File name for generated report

"\$ITERATIONARTIFACT\$_webview" (default) | string

File name for the generated report, specified as a string.

The built-in tasks use tokens, like \$ITERATIONARTIFACT\$, as placeholders for dynamic path resolution during run-time. For more information, see "Dynamically Resolve Paths with Tokens".

Example: "myModel_webview"

Data Types: string

ReportPath — Path to generated report

fullfile('\$DEFAULTOUTPUTDIR\$', '\$ITERATIONARTIFACT\$', 'webview') (default) | string

Path to the generated report, specified as a string.

Data Types: string

SearchScope — Systems to export

"All" (default) | "CurrentAndBelow" | "CurrentAndAbove" | "Current"

Systems to export, relative to the system_name system, specified as "All", "CurrentAndBelow", "CurrentAndAbove", or "Current".

Example: "Current"

ShowProgressBar — Display status bar when exporting web view

false or 0 (default) | true or 1

Display the status bar when you export a web view, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

ViewFile — Display exported web view in web browser

false or 0 (default) | true or 1

Display the web view in a web browser when you export the web view, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

Methods**Specialized Public Methods**

This class overrides the following inherited methods.

run	<p>Create web views using Simulink Report Generator</p> <hr/> <p>Note You do not need to manually invoke this method. When you run a task using the Process Advisor app or the <code>runprocess</code> function, the build system automatically invokes the <code>run</code> method for the task.</p> <hr/> <p>The <code>run</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this built-in task, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = run(obj, input) ... end</pre>
dryRun	<p>Dry run the task to validate task inputs and generate representative task outputs without actually running the task. The <code>dryRun</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this class, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = dryRun(obj, input) ... end</pre> <p>For more information, see “Dry Run Tasks to Test Process Model”.</p>
launchToolAction	<p>Launch web view options dialog. Process Advisor uses this method when you open the tool associated with a task.</p>

Examples

Add Web View Task to Process

Add a task that can create web views for the models in your project.

Open the process model for your project. If you do not have a process model, open the Process Advisor app to automatically create a process model.

In the process model file, add the `GenerateSimulinkWebView` task to your process model by using the `addTask` method.

```
slwebTask = pm.addTask(padv.builtin.task.GenerateSimulinkWebView);
```

You can reconfigure the task behavior by using the task properties. For example, to have the task not follow links into library blocks:

```
slwebTask.FollowLinks = false;
```

Tips

- If you run MATLAB using the `-nodisplay` option or you use a machine that does not have a display (like many CI runners and Docker containers), you should set up a virtual display server before you include this task in your process model. For information, see "Set Up Virtual Display for No-Display Machine" in the User's Guide.

See Also

Process Advisor | `runprocess`

Topics

"Add Tasks to Process"

"Modify Default Process Model to Fit Your Process"

"Overview of Process Model"

padv.builtin.task.MergeTestResults Class

Namespace: padv.builtin.task padv.builtin.task padv.builtin.task

Superclasses: padv.Task

Task for generating consolidated test results report and merged coverage reports

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.task.MergeTestResults` class provides a task that can generate a consolidated test results report and merged coverage reports using Simulink Test and Simulink Coverage™. The task can generate the following artifacts for a model:

- a consolidated test results report
- a merged model coverage report for normal mode simulation results
- a merged code coverage report for software-in-the-loop (SIL) mode results
- a merged code coverage report for processor-in-the-loop (PIL) mode results

You can run your tests using the built-in task `padv.builtin.task.RunTestsPerTestCase` and then generate the reports using the `MergeTestResults` task. You can add these tasks to your process model by using the method `addTask`. After you add the tasks to your process model, you can run the tasks from the Process Advisor app or by using the function `runprocess`.

Alternatively, you can run your tests using the built-in task `padv.builtin.task.RunTestsPerModel`, but to generate the consolidated test results report and merged coverage report you need to reconfigure the `MergeTestResults` task.

To view the source code for this built-in task, in the MATLAB Command Window, enter:

```
open padv.builtin.task.MergeTestResults
```

The `padv.builtin.task.MergeTestResults` class is a `handle` class.

Creation

Description

`task = padv.builtin.task.MergeTestResults()` creates a task for generating a consolidated test results report and merged coverage reports using Simulink Test and Simulink Coverage.

`task = padv.builtin.task.MergeTestResults(Name=Value)` sets certain properties using one or more name-value arguments. For example, `task = padv.builtin.task.MergeTestResults(Name = "MyTestAndCoverageReportsTask")` creates a task with the specified name.

You can use this syntax to set property values for `Name`, `Title`, `IterationQuery`, `InputQueries`, `InputDependencyQuery`, or `Licenses`.

The `padv.builtin.task.MergeTestResults` class also has other properties, but you cannot set those properties during task creation.

Properties

The `MergeTestResults` class inherits properties from `padv.Task`. The properties listed in “Specialized Inherited Properties” on page 7-0 are `padv.Task` properties that the `MergeTestResults` task overrides.

The task also has properties for specifying:

- “Test Result Options” on page 7-0
- “Coverage Report Options” on page 7-0

Specialized Inherited Properties

Name — Unique identifier for task in process

`"padv.builtin.task.MergeTestResults"` (default) | string

Unique identifier for task in process, specified as a string.

Example: `"MyTestAndCoverageReportsTask"`

Data Types: string

Title — Human-readable name that appears in Process Advisor app

`"Merge Test Results"` (default) | string

Human-readable name that appears in Process Advisor app, specified as a string.

Example: `"My Test And Coverage Reports Task"`

Data Types: string

DescriptionText — Task description

`"This task uses Simulink Test and Simulink Coverage to generate a consolidated test results report and a merged coverage report for a model."` (default) | string

Task description, specified as a string.

When you point to a task in Process Advisor and click the information icon, the tooltip shows the task description.

Example: `"This task uses Simulink Test and Simulink Coverage to generate a consolidated test results report and a merged coverage report for a model."`

Data Types: string

DescriptionCSH — Path to task documentation

`path to MergeTestResults documentation` (default) | string

Path to task documentation, specified as a string.

When you point to a task in Process Advisor, click the ellipsis (...), and click **Help**, Process Advisor opens the task documentation.

Example: `fullfile(pwd, "taskHelpFiles", "myTaskDocumentation.pdf")`

Data Types: string

RequiredIterationArtifactType – Artifact type that task can run on

"sl_model_file" (default) | ...

Type of artifact, specified as one or more of the values listed in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file

Category	Artifact Type	Description
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "sl_model_file"

Example: ["sl_model_file "zc_file"]

IterationQuery – Find artifacts that task iterates over

`padv.builtin.query.FindModelsWithTestCases` (default) | `padv.Query` object | name of `padv.Query` object

Query that finds the artifacts that the task iterates over, specified as a `padv.Query` object or the name of a `padv.Query` object. When you specify `IterationQuery`, the task runs one time *for each* artifact returned by the query. In the Process Advisor app, the artifacts returned by `IterationQuery` appear under task title.

For more information about task iterations, see “Overview of Process Model”.

InputDependencyQuery – Finds artifact dependencies for task inputs

`padv.builtin.query.GetDependentArtifacts` (default) | `padv.Query` object | name of `padv.Query` object

Query that finds artifact dependencies for task inputs, specified as a `padv.Query` object or the name of a `padv.Query` object.

The build system runs the query specified by `InputDependencyQuery` to find the dependencies for the task inputs, since those dependencies can impact if task results are up-to-date. For more information, see “Overview of Process Model”.

Example: `padv.builtin.query.GetDependentArtifacts`

Licenses — List of licenses that task requires

"simulink_test" (default) | string

List of licenses that the task requires, specified as a string.

Data Types: string

CISupportOutputsByTask — Type of CI-compatible result files that task generates when run

"JUnit" (default)

Type of CI-compatible result files that the task itself generates when run, specified as either:

- "JUnit" — JUnit-style XML report for task results.
- "" — None. The build system generates a JUnit-style XML report for the task instead.

InputQueries — Inputs to task

[`padv.builtin.query.GetIterationArtifact`, `padv.builtin.query.GetOutputsOfDependentTask(Task="padv.builtin.task.RunTestsPerTestCase")`] (default) | `padv.Query` object | name of `padv.Query` object | array of `padv.Query` objects

Inputs to the task, specified as:

- a `padv.Query` object
- the name of `padv.Query` object
- an array of `padv.Query` objects
- an array of names of `padv.Query` objects

By default, the task `MergeTestResults` finds the models with test cases and the associated test results by using the built-in queries:

- `padv.builtin.query.GetIterationArtifact`
- `padv.builtin.query.GetOutputsOfDependentTask` on the task `padv.builtin.task.RunTestsPerTestCase`

OutputDirectory — Location for standard task outputs

`fullfile('$DEFAULTOUTPUTDIR$', '$ITERATIONARTIFACT$', 'test_results')` (default) | string

Location for standard task outputs, specified as a string.

The built-in tasks use tokens, like `$DEFAULTOUTPUTDIR$`, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Data Types: string

Test Result Options**Author — Name of report author**`"Process Advisor" (default) | string`

Name of the report author, specified as a string.

Example: "My Team Name"

Data Types: `string`

IncludeComparisonSignalPlots — Include signal comparison plots`false or 0 (default) | true or 1`

Include the signal comparison plots defined under baseline criteria, equivalence criteria, or assessments using the verify operator in the test case, specified as a numeric or logical 1 (`true`) or 0 (`false`).

Example: `true`

Data Types: `logical`

IncludeCoverageResult — Include coverage metrics collected at test execution`true or 1 (default) | false or 0`

Include coverage metrics that are collected at test execution, specified as a numeric or logical 1 (`true`) or 0 (`false`).

Example: `false`

Data Types: `logical`

IncludeErrorMessages — Include error messages from test case simulations`true or 1 (default) | false or 0`

Include error messages from the test case simulations, specified as a numeric or logical 1 (`true`) or 0 (`false`).

Example: `false`

Data Types: `logical`

IncludeMATLABFigures — Include figures in report`false or 0 (default) | true or 1`

Include the figures opened from a callback script, custom criteria, or by the model in the report, specified as a numeric or logical 1 (`true`) or 0 (`false`).

Example: `true`

Data Types: `logical`

IncludeMLVersion — Include MATLAB version that ran test cases`true or 1 (default) | false or 0`

Include the version of MATLAB used to run the test cases, specified as a numeric or logical 1 (`true`) or 0 (`false`).

Example: `false`

Data Types:

IncludeSimulationMetadata — Include simulation metadata for each test case or iteration

false or 0 (default) | true or 1

Include simulation metadata for each test case or iteration, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

IncludeSimulationSignalPlots — Include simulation output plots of each signal

false or 0 (default) | true or 1

Include the simulation output plots of each signal, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

IncludeTestRequirement — Include test requirement link

true or 1 (default) | false or 0

Include the test requirement link defined under Requirements in the test case, specified as a numeric or logical 1 (true) or 0 (false).

Example: false

Data Types: logical

IncludeTestResults — Include all or subset of test results in report

0 (default) | 1 | 2

Include all or a subset of test results in the report, specified as either:

- 0 — Passed and failed results
- 1 — Only passed results
- 2 — Only failed results

Example: 2

LaunchReport — Open generated report

false or 0 (default) | true or 1

Open the generated report, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

LoadSimulationSignalData — Task loads simulation signal data when loading test results

false or 0 (default) | true or 1

Task loads simulation signal data when loading test results, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

NumPlotColumnsPerPage — Number of columns of plots to include on report pages

1 (default) | 2 | 3 | 4

Number of columns of plots to include on report pages, specified as an integer 1, 2, 3, or 4.

Example: 2

NumPlotRowsPerPage — Number of rows of plots to include on report pages

2 (default) | 1 | 3 | 4

Number of rows of plots to include on report pages, specified as an integer 1, 2, 3, or 4.

Example: 1

ReportFormat — Output format for generated report

"pdf" (default) | "docx" | "zip"

Output format for the generated report, specified as either:

- "pdf" — PDF format
- "docx" — Microsoft Word document format
- "zip" — Zipped file

Example: "zip"

ReportPath — Path to generated reportfullfile('\$DEFAULTOUTPUTDIR\$', '\$ITERATIONARTIFACT\$', 'test_results') (default)
| string

Path to the generated report, specified as a string.

The built-in tasks use tokens, like \$DEFAULTOUTPUTDIR\$, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Data Types: string

ReportName — File name for generated report

"\$ITERATIONARTIFACT\$_Test_Report" (default) | string

File name for the generated report, specified as a string.

The built-in tasks use tokens, like \$ITERATIONARTIFACT\$, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Data Types: string

ReportTitle — Title of report

"\$ITERATIONARTIFACT\$ TEST REPORT" (default) | string

Title of the report, specified as a string.

The built-in tasks use tokens, like \$ITERATIONARTIFACT\$, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Data Types: string

Coverage Report Options

CovAllTestInMdlSumm — Include each test in model summary

false or 0 (default) | true or 1

Include each test in the model summary, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

CovBarGrInMdlSumm — Produce bar graphs in model summary

true or 1 (default) | false or 0

Produce bar graphs in the model summary, specified as a numeric or logical 1 (true) or 0 (false).

Example: false

Data Types: logical

CovComplexInBlkTable — Include cyclomatic complexity numbers in block details

true or 1 (default) | false or 0

Include cyclomatic complexity numbers in block details, specified as a numeric or logical 1 (true) or 0 (false).

Example: false

Data Types: logical

CovComplexInSumm — Include cyclomatic complexity numbers in summary

true or 1 (default) | false or 0

Include cyclomatic complexity numbers in summary, specified as a numeric or logical 1 (true) or 0 (false).

Example: false

Data Types: logical

CovElimFullCov — Exclude fully covered model objects from report

false or 0 (default) | true or 1

Exclude fully covered model objects from report, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

CovElimFullCovDetails — Exclude fully covered model object details from report

true or 1 (default) | false or 0

Exclude fully covered model object details from report, specified as a numeric or logical 1 (true) or 0 (false).

Example: false

Data Types: logical

CovFiltExecMetric — Filter Execution metric from report

false or 0 (default) | true or 1

Filter Execution metric from report, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

CovFiltSFEvent — Filter Stateflow events from report

false or 0 (default) | true or 1

Filter Stateflow events from report, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

CovGenerateWebViewReport — Generate web view report

false or 0 (default) | true or 1

Generate web view report, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

CovHitCntInMdlSumm — Display hit/count ratio in model summary

false or 0 (default) | true or 1

Display hit/count ratio in the model summary, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

CovReportName — Name of generated model coverage report

"\$ITERATIONARTIFACT\$_ModelCoverage_Report.html" (default) | string

Name of generated model coverage report, specified as a string. The report is an aggregated coverage report for normal simulation mode results.

The built-in tasks use tokens, like \$ITERATIONARTIFACT\$, as placeholders for dynamic path resolution during run-time. For more information, see "Dynamically Resolve Paths with Tokens".

Example: "myModel_NormalModelCoverage_Report.html"

Data Types: string

CovReportNameSIL — Name of generated SIL code coverage report

"\$ITERATIONARTIFACT\$_SIL_CodeCoverage_Report.html" (default) | string

Name of generated software-in-the-loop (SIL) code coverage report, specified as a string. The report is an aggregated coverage report for SIL mode results.

The built-in tasks use tokens, like \$ITERATIONARTIFACT\$, as placeholders for dynamic path resolution during run-time. For more information, see "Dynamically Resolve Paths with Tokens".

Example: "myModel_SIL_CodeCoverage_Report.html"

Data Types: string

CovReportNamePIL — Name of generated (PIL) code coverage report

"\$ITERATIONARTIFACT\$_PIL_CodeCoverage_Report.html" (default) | string

Name of generated processor-in-the-loop (PIL) code coverage report, specified as a string. The report is an aggregated coverage report for PIL mode results.

The built-in tasks use tokens, like \$ITERATIONARTIFACT\$, as placeholders for dynamic path resolution during run-time. For more information, see "Dynamically Resolve Paths with Tokens".

Example: "myModel_PIL_CodeCoverage_Report.html"

Data Types: string

CovShowReport — Show coverage report

false or 0 (default) | true or 1

Show coverage report, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

CovTwoColorBarGraphs — Use two-color bar graphs

true or 1 (default) | false or 0

Use two-color bar graphs, specified as a numeric or logical 1 (true) or 0 (false).

Example: false

Data Types: logical

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Generate a consolidated test results report and merged coverage reports using Simulink Test and Simulink Coverage</p> <hr/> <p>Note You do not need to manually invoke this method. When you run a task using the Process Advisor app or the <code>runprocess</code> function, the build system automatically invokes the <code>run</code> method for the task.</p> <hr/> <p>The <code>run</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this built-in task, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = run(obj, input) ... end</pre>
dryRun	<p>Dry run the task to validate task inputs and generate representative task outputs without actually running the task. The <code>dryRun</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this class, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = dryRun(obj, input) ... end</pre> <p>For more information, see “Dry Run Tasks to Test Process Model”.</p>

Examples

Add Merge Test Results Task to Process

Add a task that can generate a consolidated test results report and merged coverage reports for models in your project.

Open the process model for your project. If you do not have a process model, open the Process Advisor app to automatically create a process model.

In the process model file, add the `GenerateSimulinkWebView` task to your process model by using the `addTask` method.

```
mergeTestTask = pm.addTask(padv.builtin.task.MergeTestResults);
```

You can reconfigure the task behavior by using the task properties. For example, to change where the consolidated test results report and merged model coverage report generate:

```
defaultTestResultPath = fullfile('$DEFAULTOUTPUTDIR$', 'test_results');
mergeTestTask.ReportPath = defaultTestResultPath;
mergeTestTask.CovReportPath = defaultTestResultPath;
```

The `MergeTestResults` task requires outputs from the `RunTestsPerTestCase` task. Specify this dependency in your process model by using the `dependsOn` method.

To make sure that you run your tasks using the built-in task `RunTestsPerTestCase` before you add the `MergeTestResults` task to the process model, you can use conditional logic in your process model. For example:

```
includeTestsPerTestCaseTask = true;
includeMergeTestResultsTask = true;

%% Run tests per test case
% Tools required: Simulink Test
if includeTestsPerTestCaseTask
    milTask = pm.addTask(padv.builtin.task.RunTestsPerTestCase);
    % ... Optionally specify task property values
end

%% Merge test results
% Tools required: Simulink Test (and optionally Simulink Coverage)
if includeTestsPerTestCaseTask && includeMergeTestResultsTask
    mergeTestTask = pm.addTask(padv.builtin.task.MergeTestResults);
    % ... Optionally specify task property values
end

% Set task dependencies
if includeTestsPerTestCaseTask && includeMergeTestResultsTask
    mergeTestTask.dependsOn(milTask, WhenStatus=["Pass", "Fail"]);
end
```

This code specifies that the `MergeTestResults` task runs when the status of the `RunTestsPerTestCase` task is either a passing or failing result. If you only want to run the `MergeTestResults` when the `RunTestsPerTestCase` task passes, you can specify `WhenStatus` as "Pass" instead. For more information, see `dependsOn`.

Tips

Run your tests using the built-in task `padv.builtin.task.RunTestsPerTestCase`.

See Also

Process Advisor | `runprocess`

Topics

"Add Tasks to Process"

“Modify Default Process Model to Fit Your Process”
“Overview of Process Model”

padv.builtin.task.RunCodeInspection Class

Namespace: padv.builtin.task padv.builtin.task padv.builtin.task

Superclasses: padv.Task

Task for inspecting generated model code using Simulink Code Inspector

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.task.RunCodeInspection` class provides a task that can detect unintended functionality in your models by establishing model-to-code and code-to-model traceability using Simulink Code Inspector. The results of this task can help you to satisfy code-review objectives in DO-178 and other high-integrity standards.

You can add the task to your process model by using the method `addTask`. After you add the task to your process model, you can run the task from the Process Advisor app or by using the function `runprocess`.

To view the source code for this built-in task, in the MATLAB Command Window, enter:

```
open padv.builtin.task.RunCodeInspection
```

The `padv.builtin.task.RunCodeInspection` class is a `handle` class.

Note This task is not supported on macOS.

Creation

Description

`task = padv.builtin.task.RunCodeInspection()` creates a task for detecting unintended functionality in models using Simulink Code Inspector.

`task = padv.builtin.task.RunCodeInspection(Name=Value)` sets certain properties using one or more name-value arguments. For example, `task = padv.builtin.task.RunCodeInspection(Name = "MyCodeInspectionTask")` creates a task with the specified name.

You can use this syntax to set property values for `IsTopModel`, `ReportFolder`, `Name`, `Title`, `IterationQuery`, `InputDependencyQuery`, `InputQueries`, `LaunchToolAction`, and `LaunchToolText`.

The `padv.builtin.task.RunCodeInspection` class also has other properties, but you cannot set those properties during task creation.

Properties

The `RunCodeInspection` class inherits properties from `padv.Task`. The properties listed in “Specialized Inherited Properties” on page 7-0 are `padv.Task` properties that the `RunCodeInspection` task overrides.

The task also has properties for specifying “Code Inspection Options” on page 7-0 for creating a code inspection object `slci.Configuration`.

Specialized Inherited Properties

Name — Unique identifier for task in process

`"padv.builtin.task.RunCodeInspection"` (default) | string

Unique identifier for task in process, specified as a string.

Example: `"MyCodeInspectionTask"`

Data Types: string

Title — Human-readable name that appears in Process Advisor app

`"Inspect Code"` (default) | string

Human-readable name that appears in Process Advisor app, specified as a string.

Example: `"My Code Inspection Task"`

Data Types: string

DescriptionText — Task description

`"This task uses Simulink Code Inspector to detect unintended functionality in your models by establishing model-to-code and code-to-model traceability."` (default) | string

Task description, specified as a string.

When you point to a task in Process Advisor and click the information icon, the tooltip shows the task description.

Example: `"This task uses Simulink Code Inspector to detect unintended functionality in your models by establishing model-to-code and code-to-model traceability."`

Data Types: string

DescriptionCSH — Path to task documentation

`path to RunCodeInspection documentation` (default) | string

Path to task documentation, specified as a string.

When you point to a task in Process Advisor, click the ellipsis (...), and click **Help**, Process Advisor opens the task documentation.

Example: `fullfile(pwd, "taskHelpFiles", "myTaskDocumentation.pdf")`

Data Types: string

RequiredIterationArtifactType — Artifact type that task can run on

`"sl_model_file"` (default) | ...

Type of artifact, specified as one or more of the values listed in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture

Category	Artifact Type	Description
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
"sl_test_suite_result"	Simulink Test suite result	

Example: "sl_model_file"

Example: ["sl_model_file" "zc_file"]

IterationQuery – Find artifacts that task iterates over

`padv.builtin.query.FindModels (default) | padv.Query object | name of padv.Query object`

Query that finds the artifacts that the task iterates over, specified as a `padv.Query` object or the name of a `padv.Query` object. When you specify `IterationQuery`, the task runs one time *for each* artifact returned by the query. In the Process Advisor app, the artifacts returned by `IterationQuery` appear under task title.

For more information about task iterations, see “Overview of Process Model”.

Example: `padv.builtin.query.FindProjectFile`

InputDependencyQuery – Finds artifact dependencies for task inputs

`padv.Query object | name of padv.Query object`

Query that finds artifact dependencies for task inputs, specified as a `padv.Query` object or the name of a `padv.Query` object.

The build system runs the query specified by `InputDependencyQuery` to find the dependencies for the task inputs, since those dependencies can impact if task results are up-to-date.

For more information about task inputs, see “Overview of Process Model”.

Example: `padv.builtin.query.GetDependentArtifacts`

Licenses — List of licenses that task requires

`"simulink_code_inspector"` (default) | string

List of licenses that the task requires, specified as a string.

Data Types: string

LaunchToolAction — Function that launches tool

`@launchToolAction` (default) | function handle

Function that launches a tool, specified as the function handle.

When you point to a task in the Process Advisor app, you can click the ellipsis (...) to see more options. For built-in tasks, you have the option to launch a tool associated with the task.

For the task `RunCodeInspection`, you can launch the Code Inspector app.

Data Types: `function_handle`

LaunchToolText — Description of action that LaunchToolAction property performs

`"Open Code Inspector"` (default) | string

Description of the action that the `LaunchToolAction` property performs, specified as a string.

Data Types: string

InputQueries — Inputs to task

`padv.Query` object | name of `padv.Query` object | array of `padv.Query` objects

Inputs to the task, specified as:

- a `padv.Query` object
- the name of `padv.Query` object
- an array of `padv.Query` objects
- an array of names of `padv.Query` objects

By default, the task `RunCodeInspection` gets the model and generated code inputs by using the built-in queries:

- `padv.builtin.query.GetIterationArtifact`
- `padv.builtin.query.GetOutputsOfDependentTask` on the task
`padv.builtin.task.GenerateCode`

OutputDirectory — Location for standard task outputs

`fullfile('$DEFAULTOUTPUTDIR$', '$ITERATIONARTIFACT$', 'code_inspection')` (default) | string

Location for standard task outputs, specified as a string.

The built-in tasks use tokens, like `$DEFAULTOUTPUTDIR$`, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Data Types: string

Code Inspection Options

IsTopModel — Setting for specifying if current model is top model

`[]` (default) | `true` or `1` | `false` or `0`

Setting for specifying if current model is top model, specified as an empty logical array `[]` or as a numeric or logical `0` (`false`) or `1` (`true`).

By default, the task automatically identifies whether a model is a top model or a reference model. But you can specify `IsTopModel` as `true` or `false` if you want to override that behavior and only generate top model code or reference model code.

Example: `true`

Data Types: `logical`

ReportFolder — Path to generated report

`fullfile('$DEFAULTOUTPUTDIR$', '$ITERATIONARTIFACT$', 'code_inspection')`
(default) | `string`

Path to generated report, specified as a string.

The task uses this property to specify the report folder for code inspection.

The built-in tasks use tokens, like `$DEFAULTOUTPUTDIR$`, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Data Types: `string`

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Detect unintended functionality in your models by establishing model-to-code and code-to-model traceability using Simulink Code Inspector</p> <hr/> <p>Note You do not need to manually invoke this method. When you run a task using the Process Advisor app or the <code>runprocess</code> function, the build system automatically invokes the <code>run</code> method for the task.</p> <hr/> <p>The <code>run</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this built-in task, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = run(obj, input) ... end</pre>
dryRun	<p>Dry run the task to validate task inputs and generate representative task outputs without actually running the task. The <code>dryRun</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this class, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = dryRun(obj, input) ... end</pre> <p>For more information, see “Dry Run Tasks to Test Process Model”.</p>
launchToolAction	<p>Launch the Code Inspector app. Process Advisor uses this method when you open the tool associated with a task.</p>

Examples

Add Code Inspection Task to Process

Add a task that can detect unintended functionality in your models by establishing model-to-code and code-to-model traceability using Simulink Code Inspector.

Open the process model for your project. If you do not have a process model, open the Process Advisor app to automatically create a process model.

In the process model file, add the `RunCodeInspection` task to your process model by using the `addTask` method. By default, the `RunCodeInspection` task automatically identifies whether a model is a top model or a reference model.

```
slciTask = pm.addTask(padv.builtin.task.RunCodeInspection);
```

You can reconfigure the task behavior by using the task properties. For example, to specify a different location for the code inspection report:

```
slciTask.ReportFolder = fullfile("reports","code_inspection");
```

The `RunCodeInspection` task requires outputs from the `GenerateCode` task. Specify this dependency in your process model by using the `dependsOn` method.

To make sure that you run your tasks using the built-in task `GenerateCode` before you add the task, you can use conditional logic in your process model. For example:

```
includeGenerateCodeTask = true;
includeCodeInspection = true;

%% Generate Code
% Tools required: Embedded Coder
% By default, generating code for each model in the project
if includeGenerateCodeTask
    codegenTask = pm.addTask(padv.builtin.task.GenerateCode);
    % ... Optionally specify task property values
end

%% Inspect Code
% Tools required: Simulink Code Inspector
if includeGenerateCodeTask && includeCodeInspection
    slciTask = pm.addTask(padv.builtin.task.RunCodeInspection(IterationQuery=findModels));
    % ... Optionally specify task property values
end

%% Set Task Dependencies
if includeGenerateCodeTask && includeCodeInspection
    slciTask.dependsOn(codegenTask);
end
```

See Also

`addTask` | `padv.builtin.task.AnalyzeModelCode` | `padv.builtin.task.GenerateCode` | `padv.builtin.query.FindCodeForModel` | `padv.ProcessModel` | Process Advisor | `runprocess`

Topics

“Add Tasks to Process”

“Modify Default Process Model to Fit Your Process”

“Overview of Process Model”

padv.builtin.task.RunModelStandards Class

Namespace: padv.builtin.task padv.builtin.task padv.builtin.task

Superclasses: padv.Task

Task for checking modeling standards with Model Advisor

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.task.RunModelStandards` class provides a task that can check your models for modeling conditions and configuration settings that cause inaccurate or inefficient simulation of the system that the model represents by using the Model Advisor app. Running model standards checking can also help you verify compliance with industry standards and guidelines.

You can add the task to your process model by using the method `addTask`. After you add the task to your process model, you can run the task from the Process Advisor app or by using the function `runprocess`. You can configure this task to specify which model standards the task runs. For example, you can specify a Model Advisor configuration file or list of check identifiers to include in the Model Advisor analysis. If you do not specify which model standards to run, the task runs a subset of high-integrity systems checks by default.

To view the source code for this built-in task, in the MATLAB Command Window, enter:

```
open padv.builtin.task.RunModelStandards
```

The `padv.builtin.task.RunModelStandards` class is a `handle` class.

Creation

Description

`task = padv.builtin.task.RunModelStandards()` creates a task for checking modeling standards using Model Advisor.

`task = padv.builtin.task.RunModelStandards(Name=Value)` sets certain properties using one or more name-value arguments. For example, `task = padv.builtin.task.RunModelStandards(Name = "MyModelAdvisorTask")` creates a task with the specified name.

You can use this syntax to set property values for `InputQueries`, `Name`, `IterationQuery`, `InputDependencyQuery`, `Licenses`, `LaunchToolAction`, and `LaunchToolText`.

The `padv.builtin.task.RunModelStandards` class also has other properties, but you cannot set those properties during task creation.

Properties

The `RunModelStandards` class inherits properties from `padv.Task`. The properties listed in “Specialized Inherited Properties” on page 7-0 are `padv.Task` properties that the `RunModelStandards` task overrides.

The task also has properties for specifying “Model Advisor Options” on page 7-0 for running Model Advisor checks by using `ModelAdvisor.run`.

Specialized Inherited Properties

Name — Unique identifier for task in process

`"padv.builtin.task.RunModelStandards"` (default) | string

Unique identifier for task in process, specified as a string.

Example: `"MyModelAdvisorTask"`

Data Types: string

Title — Human-readable name that appears in Process Advisor app

`"Check Modeling Standards"` (default) | string

Human-readable name that appears in Process Advisor app, specified as a string.

Example: `"My Model Advisor Task"`

Data Types: string

DescriptionText — Task description

`"This task uses the Model Advisor to check your models for modeling conditions and configuration settings."` (default) | string

Task description, specified as a string.

When you point to a task in Process Advisor and click the information icon, the tooltip shows the task description.

Example: `"This task uses the Model Advisor to check your models for modeling conditions and configuration settings."`

Data Types: string

DescriptionCSH — Path to task documentation

`path to RunModelStandards documentation` (default) | string

Path to task documentation, specified as a string.

When you point to a task in Process Advisor, click the ellipsis (...), and click **Help**, Process Advisor opens the task documentation.

Example: `fullfile(pwd,"taskHelpFiles","myTaskDocumentation.pdf")`

Data Types: string

RequiredIterationArtifactType — Artifact type that task can run on

`"sl_model_file"` (default) | ...

Type of artifact, specified as one or more of the values listed in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture

Category	Artifact Type	Description
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
"sl_test_suite_result"	Simulink Test suite result	

Example: "sl_model_file"

Example: ["sl_model_file" "zc_file"]

IterationQuery – Find artifacts that task iterates over

`padv.builtin.query.FindModels (default) | padv.Query object | name of padv.Query object`

Query that finds the artifacts that the task iterates over, specified as a `padv.Query` object or the name of a `padv.Query` object. When you specify `IterationQuery`, the task runs one time *for each* artifact returned by the query. In the Process Advisor app, the artifacts returned by `IterationQuery` appear under task title.

For more information about task iterations, see “Overview of Process Model”.

InputDependencyQuery – Finds artifact dependencies for task inputs

`padv.Query object | name of padv.Query object`

Query that finds artifact dependencies for task inputs, specified as a `padv.Query` object or the name of a `padv.Query` object.

The build system runs the query specified by `InputDependencyQuery` to find the dependencies for the task inputs, since those dependencies can impact if task results are up-to-date.

For more information about task inputs, see “Overview of Process Model”.

Example: `padv.builtin.query.GetDependentArtifacts`

LaunchToolAction — Function that launches tool`@launchToolAction (default) | function handle`

Function that launches a tool, specified as the function handle.

When you point to a task in the Process Advisor app, you can click the ellipsis (...) to see more options. For built-in tasks, you have the option to launch a tool associated with the task.

For the task `RunModelStandards`, you can launch the Model Advisor app.

Data Types: `function_handle`

LaunchToolText — Description of action that LaunchToolAction property performs`"Open Model Advisor" (default) | string`

Description of the action that the `LaunchToolAction` property performs, specified as a string.

Data Types: `string`

InputQueries — Inputs to task`padv.Query object | name of padv.Query object | array of padv.Query objects`

Inputs to the task, specified as:

- a `padv.Query` object
- the name of `padv.Query` object
- an array of `padv.Query` objects
- an array of names of `padv.Query` objects

By default, the task `RunModelStandards` gets the current model that the task is iterating over by using the built-in query `padv.builtin.query.GetIterationArtifact`.

OutputDirectory — Location for standard task outputs`fullfile('$DEFAULTOUTPUTDIR','$ITERATIONARTIFACT$', 'model_standards') (default) | string`

Location for standard task outputs, specified as a string.

The built-in tasks use tokens, like `$DEFAULTOUTPUTDIR`, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Data Types: `string`

Model Advisor Options**CheckIDList — List of unique identifiers for Model Advisor checks**`missing (default) | string`

List of unique identifiers for Model Advisor checks, specified as string.

Note If you specify `CheckIDList` and add a Model Advisor configuration file as an input for the task, the task runs Model Advisor using the Model Advisor configuration file and ignores the list of check IDs.

Example: `["mathworks.jmaab.db_0032", "mathworks.jmaab.jc_0281"]`

Data Types: `string`

DisplayResults — Report display setting for Model Advisor

`"Summary"` (default) | `"Details"` | `"None"`

Report display setting for Model Advisor, specified as either:

- `"Summary"` — Display summary of the system results in the Command Window
- `"Details"` — Display a summary of the system results and the pass and fail results for each check on each system
- `"None"` — Do not display information in the Command Window

Example: `"Details"`

ExtensiveAnalysis — Extensive analysis setting for Model Advisor

`"on"` (default) | `"off"`

Extensive analysis setting for Model Advisor, specified as either:

- `"off"` — Model Advisor only runs checks in your configuration that do not trigger extensive analysis
- `"on"` — Model Advisor runs each check in your Model Advisor configuration file, including checks that trigger extensive analysis with tools like Simulink Design Verifier

Example: `"off"`

Force — Delete existing Model Advisor reports without prompts

`"on"` (default) | `"off"`

Delete existing Model Advisor reports without prompts, specified as either:

- `"off"` — Prompt you before removing existing `modeladvisor/system` folders.
- `"on"` — Automatically removes existing `modeladvisor/system` folders

Example: `"off"`

ParallelMode — Parallel execution setting for Model Advisor

`"off"` (default) | `"on"`

Parallel execution setting for Model Advisor, specified as either:

- `"off"` — Model Advisor does not run in parallel.
- `"on"` — If you have a Parallel Computing Toolbox™ license and a multicore machine, Model Advisor can run on multiple systems in parallel. When you specify `ParallelMode` as `"on"`, MATLAB automatically creates a parallel pool.

Example: `"on"`

ReportFormat — Format of generated report

`"html"` (default) | `"docx"`

Format of the generated report, specified as either:

- `"html"` — HTML format

- "docx" — Microsoft Word document format

Example: "docx"

ReportName — Prefix for Model Advisor report file name

"\$ITERATIONARTIFACT\$_ModelAdvisor" (default) | string

Prefix for the Model Advisor report file name, specified as a string. By default, an underscore and the model name, "*_modelName*", are appended to the report file name.

The built-in tasks use tokens, like \$ITERATIONARTIFACT\$, as placeholders for dynamic path resolution during run-time. For more information, see "Dynamically Resolve Paths with Tokens".

Data Types: string

ReportPath — Folder for generated report

fullfile('\$DEFAULTOUTPUTDIR\$', '\$ITERATIONARTIFACT\$', 'model_standards') (default) | string

Folder for the generated report, specified as a string.

Data Types: string

ShowExclusions — Exclusion display setting for report

"on" (default) | "off"

Exclusion display setting for the report, specified as either:

- "off" — Report does not list Model Advisor check exclusions
- "on" — Report lists Model Advisor check exclusions

Example: "off"

TempDir — Temporary working folder for Model Advisor

"off" (default) | "on"

Temporary working folder for Model Advisor, specified as either:

- "off" — Run Model Advisor in the current working folder
- "on" — Run Model Advisor from a temporary working folder. You can use this to avoid concurrency issues when running using a parallel pool.

Example: "on"

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Check modeling standards using Model Advisor</p> <hr/> <p>Note You do not need to manually invoke this method. When you run a task using the Process Advisor app or the <code>runprocess</code> function, the build system automatically invokes the <code>run</code> method for the task.</p> <hr/> <p>The <code>run</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this built-in task, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = run(obj, input) ... end</pre>
dryRun	<p>Dry run the task to validate task inputs and generate representative task outputs without actually running the task. The <code>dryRun</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this class, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = dryRun(obj, input) ... end</pre> <p>For more information, see “Dry Run Tasks to Test Process Model”.</p>
launchToolAction	<p>Launch the Model Advisor app. Process Advisor uses this method when you open the tool associated with a task.</p>

Examples

Add Model Advisor Task to Process

Add a task that can check modeling standards using Model Advisor.

Open the process model for your project. If you do not have a process model, open the Process Advisor app to automatically create a process model.

In the process model file, add the `RunModelStandards` task to your process model by using the `addTask` method. By default, the `RunModelStandards` task runs a subset of high-integrity systems checks.

```
maTask = pm.addTask(padv.builtin.task.RunModelStandards);
```

You can reconfigure the task behavior by using the task properties. For example, to specify a different location for the Model Advisor report:

```
% Change Report path
maTask.ReportPath = fullfile(...
    "$DEFAULTOUTPUTDIR$", "$ITERATIONARTIFACT$", "model_standards_results");
```

Specify Model Advisor Configuration File

By default, the `RunModelStandards` task runs a subset of high-integrity checks. If you want the task to run the Model Advisor checks specified by the Model Advisor configuration file, you can add the configuration file as an input to the task.

In the process model, you find the Model Advisor configuration file by using the built-in query `padv.builtin.query.FindFileWithAddress` and then specify that query as the input query by using the `addInputQueries` function. For example, suppose you have a Model Advisor configuration file called `sampleChecks.json` in a folder called `tools`:

```
maTask = pm.addTask(padv.builtin.task.RunModelStandards);

% Specify which Model Advisor configuration to run
maTask.addInputQueries(padv.builtin.query.FindFileWithAddress( ...
    Type = "ma_config_file",...
    Path = fullfile("tools","sampleChecks.json")));
```

In this code for `addInputQueries`:

- The first argument, `"ma_config_file"`, specifies that the file is a Model Advisor configuration file.
- The second argument specifies the path to the Model Advisor configuration file. In this example, the configuration file is a file, `sampleChecks.json`, in the `tools` folder in the project.

Note If you provide both a list of check IDs (`CheckIDList`) and a Model Advisor configuration file for the task, the task runs Model Advisor using the Model Advisor configuration file and ignores the list of check IDs.

Specify Model Advisor Justification File

Starting in R2023a, if you want the `RunModelStandards` task to use your Model Advisor justification files when checking modeling standards, you can reconfigure the task to add the justification files as inputs.

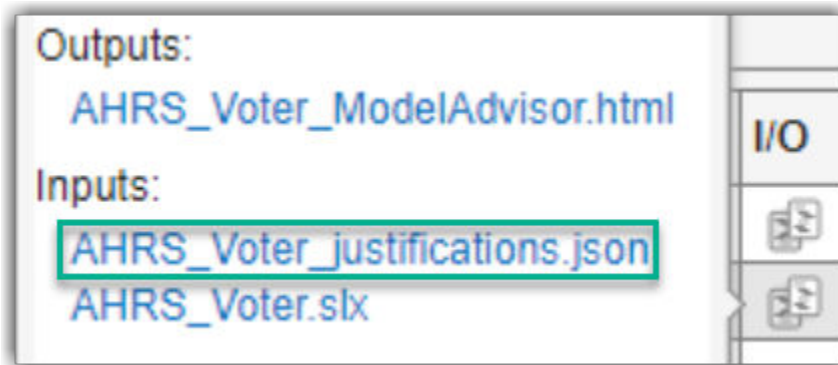
In your process model, add the built-in query `padv.builtin.query.FindMAJustificationFileForModel` as an input query for the task and

specify the folder, `JustificationFolder`, that contains the justification files. For example, if your justification files are in the directory `Justifications/ModelAdvisor` relative to your project root, add those justification files as inputs to the task by using the function `addInputQueries`.

```
maTask = pm.addTask(padv.builtin.task.RunModelStandards());

% Find and use justification files
maTask.addInputQueries(...
    padv.builtin.query.FindMAJustificationFileForModel(...
        JustificationFolder=fullfile("Justifications","ModelAdvisor")));
```

The justification file appears as an input in the **I/O** column in Process Advisor.



Create and Configure Multiple Instances of Model Advisor Task

You can add multiple instances of a task to your process model to run different task configurations. For example, you can have one instance of the built-in task `padv.builtin.task.RunModelStandards` that runs a specific Model Advisor configuration for a subset of models and another Model Advisor configuration for other models.

In your process model, to create multiple instances of a task, you need to specify unique values for the `Name` properties of each task instance. By default, the task name is the name of the task class.

```
% Tasks need unique names
maTaskA = pm.addTask(padv.builtin.task.RunModelStandards(...
    Name = "maTaskA"));
maTaskB = pm.addTask(padv.builtin.task.RunModelStandards(...
    Name = "maTaskB"));
```

If you need to reconfigure the tasks, you can specify values for the other task properties. For example, you can specify which models the task runs on, which Model Advisor configuration file the task uses, and where the reports generate.

```
% Can specify unique title for task that appears in Process Advisor
maTaskA.Title = "Check Modeling Standards - A";
maTaskB.Title = "Check Modeling Standards - B";

% Can specify different Model Advisor configurations
maTaskA.addInputQueries(padv.builtin.query.FindFileWithAddress( ...
    Type="ma_config_file", Path=fullfile("configs","configA.json")));
maTaskB.addInputQueries(padv.builtin.query.FindFileWithAddress( ...
```

```
    Type="ma_config_file", Path=fullfile("configs","configB.json"));

% Can run on different sets of models
maTaskA.IterationQuery = padv.builtin.query.FindModels(...
    IncludePath = "control");
maTaskB.IterationQuery = padv.builtin.query.FindModelsWithLabel(...
    "ProjectLabelCategory","ProjectLabel");

% Specify unique locations for Model Advisor reports
maTaskA.ReportPath = fullfile( ...
    defaultResultPath,"model_standards_A_results");
maTaskB.ReportPath = fullfile( ...
    defaultResultPath,"model_standards_B_results");
```

See Also

Process Advisor | runprocess

Topics

“Add Tasks to Process”

“Modify Default Process Model to Fit Your Process”

“Overview of Process Model”

padv.builtin.task.RunTestsPerModel Class

Namespace: padv.builtin.task padv.builtin.task padv.builtin.task

Superclasses: padv.Task

Task for running test cases associated with each model using Simulink Test

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.task.RunTestsPerModel` class provides a task that can run the test cases associated with your models using Simulink Test.

You can add the task to your process model by using the method `addTask`. After you add the task to your process model, you can run the task from the Process Advisor app or by using the function `runprocess`. The task runs each test case for each model in your project and certain tests can generate code. You can control whether Simulink Test or the MATLAB Unit Test framework executes the test cases by using the task property `UseMATLABUnitTest`.

The Process Advisor app shows the names of the models that have test cases under **Run Tests** in the **Tasks** column. If you want to see the names of both the models and the associated test cases, use the `padv.builtin.task.RunTestsPerTestCase` task instead.

To view the source code for this built-in task, in the MATLAB Command Window, enter:

```
open padv.builtin.task.RunTestsPerModel
```

The `padv.builtin.task.RunTestsPerModel` class is a handle class.

Note When you run the task, the task runs each test case individually and only executes test-case level callbacks. The task does not execute test-file level callbacks or test-suite level callbacks.

Creation

Description

`task = padv.builtin.task.RunTestsPerModel()` creates a task for running the test cases associated with your models using Simulink Test.

`task = padv.builtin.task.RunTestsPerModel(Name=Value)` sets certain properties using one or more name-value arguments. For example, `task = padv.builtin.task.RunTestsPerModel(Name = "MyRunTestsTask")` creates a task with the specified name.

You can use this syntax to set property values for `Name`, `Title`, `InputQueries`, `IterationQuery`, `InputDependencyQuery`, `Licenses`, `LaunchToolAction`, and `LaunchToolText`.

The `padv.builtin.task.RunTestsPerModel` class also has other properties, but you cannot set those properties during task creation.

Properties

The `RunTestsPerModel` class inherits properties from `padv.Task`. The properties listed in “Specialized Inherited Properties” on page 7-0 are `padv.Task` properties that the `RunTestsPerModel` task overrides.

The task also has properties for specifying “Test Execution Options” on page 7-0 .

Specialized Inherited Properties

Name — Unique identifier for task in process

`"padv.builtin.task.RunTestsPerModel"` (default) | string

Unique identifier for task in process, specified as a string.

Example: `"TestMyModels"`

Data Types: string

Title — Human-readable name that appears in Process Advisor app

`"Run Tests"` (default) | string

Human-readable name that appears in Process Advisor app, specified as a string.

Example: `"Run My Tests"`

Data Types: string

DescriptionText — Task description

`"This task uses Simulink Test to run the test cases associated with your model. The task runs the test cases on a model-by-model basis. Certain tests may generate code."` (default) | string

Task description, specified as a string.

When you point to a task in Process Advisor and click the information icon, the tooltip shows the task description.

Example: `"This task uses Simulink Test to run the test cases associated with your model. The task runs the test cases on a model-by-model basis. Certain tests may generate code."`

Data Types: string

DescriptionCSH — Path to task documentation

path to `RunTestsPerModel` documentation (default) | string

Path to task documentation, specified as a string.

When you point to a task in Process Advisor, click the ellipsis (...), and click **Help**, Process Advisor opens the task documentation.

Example: `fullfile(pwd, "taskHelpFiles", "myTaskDocumentation.pdf")`

Data Types: string

RequiredIterationArtifactType — Artifact type that task can run on

`"sl_model_file"` (default) | ...

Type of artifact, specified as one or more of the values listed in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture

Category	Artifact Type	Description
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "sl_model_file"

Example: ["sl_model_file" "zc_file"]

IterationQuery – Find artifacts that task iterates over

padv.builtin.query.FindModelsWithTestCases (default) | padv.Query object | name of padv.Query object

Query that finds the artifacts that the task iterates over, specified as a padv.Query object or the name of a padv.Query object. When you specify IterationQuery, the task runs one time *for each* artifact returned by the query. In the Process Advisor app, the artifacts returned by IterationQuery appear under task title.

For more information about task iterations, see “Overview of Process Model”.

Example: padv.builtin.query.FindModelsWithTestCases(ExcludePath = "Control")

InputDependencyQuery – Finds artifact dependencies for task inputs

padv.builtin.query.GetDependentArtifacts | padv.Query object | name of padv.Query object

Query that finds artifact dependencies for task inputs, specified as a padv.Query object or the name of a padv.Query object.

The build system runs the query specified by InputDependencyQuery to find the dependencies for the task inputs, since those dependencies can impact if task results are up-to-date. For more information, see “Overview of Process Model”.

Example: `padv.builtin.query.GetDependentArtifacts`

Licenses — List of additional licenses that task requires

`"simulink_test"` (default) | string

List of additional licenses that the task requires, specified as a string.

Data Types: string

LaunchToolAction — Function that launches tool

`@launchToolAction` (default) | function handle

Function that launches a tool, specified as the function handle.

When you point to a task in the Process Advisor app, you can click the ellipsis (...) to see more options. For built-in tasks, you have the option to launch a tool associated with the task.

For the task `RunTestsPerModel`, you can launch Simulink Test Manager.

Data Types: `function_handle`

LaunchToolText — Description of action that LaunchToolAction property performs

`"Open Test Manager"` (default) | string

Description of the action that the `LaunchToolAction` property performs, specified as a string.

Data Types: string

CISupportOutputsByTask — Type of CI-compatible result files that task generates when run

`"JUnit"` (default)

Type of CI-compatible result files that the task itself generates when run, specified as either:

- `"JUnit"` — JUnit-style XML report for task results.
- `" "` — None. The build system generates a JUnit-style XML report for the task instead.

InputQueries — Inputs to task

`[padv.builtin.query.GetIterationArtifact, padv.builtin.query.FindTestCasesForModel]` | `padv.Query` object | name of `padv.Query` object | array of `padv.Query` objects

Inputs to the task, specified as:

- a `padv.Query` object
- the name of `padv.Query` object
- an array of `padv.Query` objects
- an array of names of `padv.Query` objects

By default, the task `RunTestsPerModel` gets the current model by using the built-in query `padv.builtin.query.GetIterationArtifact` and finds the tests associated with that model by using the built-in query `padv.builtin.query.FindTestCasesForModel`.

OutputDirectory — Location for standard task outputs

`fullfile('$DEFAULTOUTPUTDIR$', '$ITERATIONARTIFACT$', 'test_results')` (default) | string

Location for standard task outputs, specified as a string.

The built-in tasks use tokens, like `$DEFAULTOUTPUTDIR$`, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Data Types: `string`

Test Execution Options

Author — Name of report author

"Process Advisor" (default) | `string`

Name of the report author, specified as a string.

Data Types: `string`

IncludeComparisonSignalPlots — Include signal comparison plots in report

`false` or `0` (default) | `true` or `1`

Include the signal comparison plots in the report, specified as a numeric or logical `1` (`true`) or `0` (`false`).

When `true`, the report includes the signal comparison plots defined under baseline criteria, equivalence criteria, or assessments using the `verify` operator in the test case.

Example: `true`

Data Types: `logical`

IncludeCoverageResult — Include coverage metrics in report

`true` or `1` (default) | `false` or `0`

Include coverage metrics that the test collects during test execution in the report, specified as a numeric or logical `1` (`true`) or `0` (`false`).

Example: `false`

Data Types: `logical`

IncludeErrorMessages — Include error messages from test case simulations in report

`true` or `1` (default) | `false` or `0`

Include error messages from test case simulations in report, specified as a numeric or logical `1` (`true`) or `0` (`false`).

Example: `false`

Data Types: `logical`

IncludeMATLABFigures — Include figures in report

`false` or `0` (default) | `true` or `1`

Include the figures opened from a callback script, custom criteria, or by the model in the report, specified as a numeric or logical `1` (`true`) or `0` (`false`).

Example: `true`

Data Types: `logical`

IncludeMLVersion — Include MATLAB version information in report

true or 1 (default) | false or 0

Include the version of MATLAB that ran the test cases in the report, specified as a numeric or logical 1 (true) or 0 (false).

Example: false

Data Types: logical

IncludeSimulationMetadata — Include simulation metadata in report

false or 0 (default) | true or 1

Include simulation metadata for each test case or iteration in the report, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

IncludeSimulationSignalPlots — Include simulation output plots for each signal in report

true or 1 (default) | false or 0

Include the simulation output plots for each signal in the report, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: logical

IncludeTestRequirement — Include test requirement link in report

true or 1 (default) | false or 0

Include the test requirement link, defined under Requirements in the test case, in the report, specified as a numeric or logical 1 (true) or 0 (false).

Example: false

Data Types: logical

IncludeTestResults — Include test results in report

0 (default) | 1 | 2

Include all or a subset of test results in the report, specified as either:

- 0 — Passed and failed results
- 1 — Only passed results
- 2 — Only failed results

Example: 2

LaunchReport — Open generated report

0 (default) | 1 | 2

Open the generated report, specified as a numeric or logical 1 (true) or 0 (false).

Example: true

Data Types: `logical`

NumPlotColumnsPerPage — Number of columns of plots to include on report pages

1 (default) | 2 | 3 | 4

Number of columns of plots to include on report pages, specified as an integer 1, 2, 3, or 4.

Example: 4

NumPlotRowsPerPage — Number of rows of plots to include on report pages

2 (default) | 1 | 3 | 4

Number of rows of plots to include on report pages, specified as an integer 1, 2, 3, or 4.

Example: 4

ReportFormat — Format for generated report

"pdf" (default) | "docx" | "zip"

Format for the generated report, specified as either:

- "pdf" — PDF format
- "docx" — Microsoft Word document format
- "zip" — Zipped file that contains an HTML file, images, style sheet, and JavaScript files for an HTML report

Example: "zip"

ReportPath — Path to generated report

`string(fullfile('$DEFAULTOUTPUTDIR$', '$ITERATIONARTIFACT$', 'test_results'))`
(default) | string

Path to the generated report, specified as a string.

The built-in tasks use tokens, like `$DEFAULTOUTPUTDIR$`, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Data Types: `string`

ReportName — File name for generated report

"\$ITERATIONARTIFACT\$_Test" (default) | string

File name for the generated report, specified as a string.

The built-in tasks use tokens, like `$ITERATIONARTIFACT$`, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Data Types: `string`

ReportTitle — Title of report

"\$ITERATIONARTIFACT\$ REPORT" (default) | string

Title of the report, specified as a string.

The built-in tasks use tokens, like `$ITERATIONARTIFACT$`, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Data Types: `string`

ResultFileName — Name of test result file

`"$ITERATIONARTIFACT$_ResultFile"` (default) | `string`

Name of test result file, specified as a string.

The built-in tasks use tokens, like `$ITERATIONARTIFACT$`, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Data Types: `string`

SaveResultsAfterRun — Save test results to file after execution

`true` or `1` (default) | `false` or `0`

Save the test results to a file after execution, specified as a numeric or logical `1` (`true`) or `0` (`false`).

Example: `false`

Data Types: `logical`

SimulationMode — Simulation mode for running tests

`""` (default) | `"Normal"` | `"Accelerator"` | `"Rapid Accelerator"` | `"Software-in-the-Loop"` | `"Processor-in-the-Loop"`

Simulation mode for running tests, specified as `"Normal"`, `"Accelerator"`, `"Rapid Accelerator"`, `"Software-in-the-Loop"`, or `"Processor-in-the-Loop"`.

By default, the property is empty (`""`), which means the built-in task uses the simulation mode that you define in the test itself. If you specify a value other than `""`, the built-in task overrides the simulation mode set in Simulink Test Manager. You do not need to update the test parameters or settings to run the test in the new mode.

Example: `"Software-in-the-Loop"`

UseMATLABUnitTest — Use MATLAB Unit Test framework to execute test cases

`false` (`0`) (default) | `true` (`1`)

Use MATLAB Unit Test framework to execute test cases, specified as either:

- `true` (`1`) — The task runs your test cases by using the MATLAB Unit Test framework to create a test runner, creates a suite of tests from your test file, and run the tests. If you use the pipeline generator, `padv.pipeline.generatePipeline`, and your pipeline generator options specify the `GenerateJUnitForProcess` property as `true` (`1`), the task uses the MATLAB unit test XML plugin to produce JUnit-style XML format test results that integrate into CI platforms.
- `false` (`0`) — The task runs your test cases by using Simulink Test. Starting in R2023a, if you specified the task property `SimulationMode`, the task overrides the test simulation mode without having to change the test definition.

Example: `true`

Data Types: `logical`

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Run test cases for each model using Simulink Test</p> <hr/> <p>Note You do not need to manually invoke this method. When you run a task using the Process Advisor app or the <code>runprocess</code> function, the build system automatically invokes the <code>run</code> method for the task.</p> <hr/> <p>The <code>run</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this built-in task, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = run(obj, input) ... end</pre>
dryRun	<p>Dry run the task to validate task inputs and generate representative task outputs without actually running the task. The <code>dryRun</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this class, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = dryRun(obj, input) ... end</pre> <p>For more information, see “Dry Run Tasks to Test Process Model”.</p>
launchToolAction	<p>Launch Simulink Test Manager. Process Advisor uses this method when you open the tool associated with a task.</p>

Examples

Add Task to Run Tests for Each Model

Add a task to your process that can run test cases for each model using Simulink Test.

Open the process model for your project. If you do not have a process model, open the Process Advisor app to automatically create a process model.

In the process model file, add the `RunTestsPerModel` task to your process model by using the `addTask` method.

```
runTestsPerModelTask = pm.addTask(padv.builtin.task.RunTestsPerModel);
```

You can reconfigure the task behavior by using the task properties. For example, to generate a zipped HTML report file instead of a PDF:

```
runTestsPerModelTask.ReportFormat = "zip";
```

If you want to use the `MergeTestResults` task to merge the test results, you need to reconfigure the input queries for the `MergeTestResults` task to get the outputs from the `RunTestsPerModel` task. By default, the `MergeTestResults` task only gets the current model and the outputs from the task `RunTestsPerTestCase`.

```
%% Merge Test Results from Running Tests per Model
mergeTestTask = pm.addTask(padv.builtin.task.MergeTestResults(...
    InputQueries = [...
        padv.builtin.query.GetIterationArtifact,...
        padv.builtin.query.GetOutputsOfDependentTask(Task = runTestsPerModelTask)]));
```

Since that `MergeTestResults` task now depends on outputs from the `RunTestsPerModel` task, you also need to explicitly specify those dependencies in the process model.

```
mergeTestTask.dependsOn(runTestsPerModelTask);
```

Run Tests in Multiple Simulation Modes

Suppose that you want to have one instance of the `RunTestsPerModel` task that runs normal mode tests and another instance that runs software-in-the-loop (SIL) tests. You can create multiple instances of the task inside your process model and then use the `SimulationMode` to override the simulation mode set in Simulink Test Manager.

Inside your process model, create multiple instances of the `RunTestsPerModel` task. When you create multiple instances of a task, you must specify a unique name for each task object. For example:

```
milTask = pm.addTask(padv.builtin.task.RunTestsPerModel(...
    Name = "RunTestsNormalMode"));
silTask = pm.addTask(padv.builtin.task.RunTestsPerModel(...
    Name = "RunTestsSILMode"));
```

The build system uses the `Name` property as the unique identifier for the task.

Reconfigure the task instances to run tests in different simulation modes. You can run tests in different simulation modes without having to change the test definition by using the `SimulationMode` property to override the mode. For example:

```
milTask.SimulationMode = "Normal";
silTask.SimulationMode = "Software-in-the-Loop";
```

To prevent task outputs from overwriting each other, reconfigure the names and locations of the task outputs by using the associated task properties. For example:

```
% Specify normal mode outputs
milTask.OutputDirectory = defaultTestResultPath;
milTask.ReportName = '$ITERATIONARTIFACT$Normal_Test';
milTask.ResultFileName = '$ITERATIONARTIFACT$Normal_ResultFile';

% Specify SIL mode outputs
silTask.OutputDirectory = defaultTestResultPath;
silTask.ReportName = '$ITERATIONARTIFACT$_SIL_Test';
silTask.ResultFileName = '$ITERATIONARTIFACT$_SIL_ResultFile';
```

The built-in tasks use tokens, like `$ITERATIONARTIFACT$`, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

By default, the `MergeTestResults` task only gets the current model and the outputs from the task `padv.builtin.task.RunTestsPerTestCase`.

If you want to merge the test results from these two task instances using the `MergeTestResults` task, you need to reconfigure the input queries for the `MergeTestResults` task to get the outputs from those task instances. For example:

```
%% Merge Test Results (Normal and SIL)
mergeTestTask = pm.addTask(padv.builtin.task.MergeTestResults(...
    InputQueries = [...
        padv.builtin.query.GetIterationArtifact,...
        padv.builtin.query.GetOutputsOfDependentTask(Task = "RunTestsNormalMode"),...
        padv.builtin.query.GetOutputsOfDependentTask(Task = "RunTestsSILMode")]);
```

Since that `MergeTestResults` task depends on outputs from the `RunTestsPerTestCase` tasks, you need to explicitly specify those dependencies in the process model.

```
mergeTestTask.dependsOn(milTask);
mergeTestTask.dependsOn(silTask);
```

See Also

Process Advisor | `runprocess`

Topics

“Add Tasks to Process”

“Modify Default Process Model to Fit Your Process”

“Overview of Process Model”

padv.builtin.task.RunTestsPerTestCase Class

Namespace: padv.builtin.task padv.builtin.task padv.builtin.task

Superclasses: padv.Task

Task for running each test case using Simulink Test

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.task.RunTestsPerTestCase` class provides a task that can run each test case using Simulink Test.

You can add the task to your process model by using the method `addTask`. After you add the task to your process model, you can run the task from the Process Advisor app or by using the function `runprocess`. The task runs each test case individually and certain tests can generate code. You can control whether Simulink Test or the MATLAB Unit Test framework executes the test cases by using the task property `UseMATLABUnitTest`.

The Process Advisor app shows the names of both the test cases and the associated models under **Run Tests** in the **Tasks** column. If you only want to see the model names, use the `padv.builtin.task.RunTestsPerModel` task instead.

To generate a consolidated test results report and a merged coverage report for your model, you can use the built-in task `padv.builtin.task.MergeTestResults`.

To view the source code for this built-in task, in the MATLAB Command Window, enter:

```
open padv.builtin.task.RunTestsPerTestCase
```

The `padv.builtin.task.RunTestsPerTestCase` class is a `handle` class.

Note Since this task runs each test case individually, the task only executes test-case level callbacks. The task does not execute test-file level callbacks or test-suite level callbacks.

Creation

Description

`task = padv.builtin.task.RunTestsPerTestCase()` creates a task for running test cases using Simulink Test.

`task = padv.builtin.task.RunTestsPerTestCase(Name=Value)` sets certain properties using one or more name-value arguments. For example, `task = padv.builtin.task.RunTestsPerTestCase(Name = "MyRunTestsTask")` creates a task with the specified name.

You can use this syntax to set property values for `Name`, `InputQueries`, `IterationQuery`, `InputDependencyQuery`, `Licenses`, `LaunchToolAction`, and `LaunchToolText`.

The `padv.builtin.task.RunTestsPerTestCase` class also has other properties, but you cannot set those properties during task creation.

Properties

The `RunTestsPerTestCase` class inherits properties from `padv.Task`. The properties listed in “Specialized Inherited Properties” on page 7-0 are `padv.Task` properties that the `RunTestsPerTestCase` task overrides.

The task also has properties for specifying “Test Execution Options” on page 7-0 .

Specialized Inherited Properties

Name — Unique identifier for task in process

`"padv.builtin.task.RunTestsPerTestCase"` (default) | string

Unique identifier for task in process, specified as a string.

Example: `"MyRunTestsTask"`

Data Types: string

Title — Human-readable name that appears in Process Advisor app

`"Run Tests"` (default) | string

Human-readable name that appears in Process Advisor app, specified as a string.

Example: `"My Run Tests Task"`

Data Types: string

DescriptionText — Task description

`"This task uses Simulink Test to run the test cases associated with your model. The task runs the test cases on a test-by-test basis. Certain tests may generate code."` (default) | string

Task description, specified as a string.

When you point to a task in Process Advisor and click the information icon, the tooltip shows the task description.

Example: `"This task uses Simulink Test to run the test cases associated with your model. The task runs the test cases on a test-by-test basis. Certain tests may generate code."`

Data Types: string

DescriptionCSH — Path to task documentation

`path to RunTestsPerTestCase documentation` (default) | string

Path to task documentation, specified as a string.

When you point to a task in Process Advisor, click the ellipsis (...), and click **Help**, Process Advisor opens the task documentation.

Example: `fullfile(pwd, "taskHelpFiles", "myTaskDocumentation.pdf")`

Data Types: string

RequiredIterationArtifactType – Artifact type that task can run on

"sl_test_case" (default) | ...

Type of artifact, specified as one or more of the values listed in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
"sl_subsystem_file"	Subsystem file	

Category	Artifact Type	Description
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "sl_test_case"

IterationQuery – Find artifacts that task iterates over

`padv.builtin.query.FindTestCasesForModel` (default) | `padv.Query` object | name of `padv.Query` object

Query that finds the artifacts that the task iterates over, specified as a `padv.Query` object or the name of a `padv.Query` object. When you specify `IterationQuery`, the task runs one time *for each* artifact returned by the query. In the Process Advisor app, the artifacts returned by `IterationQuery` appear under task title.

For more information about task iterations, see “Overview of Process Model”.

InputDependencyQuery – Finds artifact dependencies for task inputs

`padv.builtin.query.GetDependentArtifacts` | `padv.Query` object | name of `padv.Query` object

Query that finds artifact dependencies for task inputs, specified as a `padv.Query` object or the name of a `padv.Query` object.

The build system runs the query specified by `InputDependencyQuery` to find the dependencies for the task inputs, since those dependencies can impact if task results are up-to-date. For more information, see “Overview of Process Model”.

Example: `padv.builtin.query.GetDependentArtifacts`

Licenses — List of licenses that task requires

`"simulink_test"` (default) | string

List of licenses that the task requires, specified as a string.

Data Types: string

LaunchToolAction — Function that launches tool

`@launchToolAction` (default) | function handle

Function that launches a tool, specified as the function handle.

When you point to a task in the Process Advisor app, you can click the ellipsis (...) to see more options. For built-in tasks, you have the option to launch a tool associated with the task.

For the task `RunTestsPerTestCase`, you can launch Simulink Test Manager.

Data Types: function_handle

LaunchToolText — Description of action that LaunchToolAction property performs

`"Open Test Manager"` (default) | string

Description of the action that the `LaunchToolAction` property performs, specified as a string.

Data Types: string

CISupportOutputsForTask — Type of CI-compatible result files that task generates when skipped

`""` (default) | `"JUnit"`

Type of CI-compatible result files that the task itself generates when skipped, specified as either:

- `"JUnit"` — JUnit-style XML report for task results.
- `""` — None. The build system generates a JUnit-style XML report for the task instead.

CISupportOutputsByTask — Type of CI-compatible result files that task generates when run

`"JUnit"` (default) | `""`

Type of CI-compatible result files that the task itself generates when run, specified as either:

- `"JUnit"` — JUnit-style XML report for task results.
- `""` — None. The build system generates a JUnit-style XML report for the task instead.

InputQueries — Inputs to task

`"padv.builtin.query.GetIterationArtifact"` (default) | `padv.Query` object | name of `padv.Query` object | array of `padv.Query` objects

Inputs to the task, specified as:

- a `padv.Query` object
- the name of `padv.Query` object
- an array of `padv.Query` objects

- an array of names of `padv.Query` objects

By default, the task `RunTestsPerTestCase` gets the current test case that the task is iterating over by using the built-in query `padv.builtin.query.GetIterationArtifact`.

OutputDirectory — Location for standard task outputs

`fullfile('$DEFAULTOUTPUTDIR$', '$ROOTITERATIONARTIFACT$', 'test_results')`
(default) | string

Location for standard task outputs, specified as a string.

The built-in tasks use tokens, like `$DEFAULTOUTPUTDIR$`, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Data Types: string

Test Execution Options

ResultFileName — Name of test result file

`"$ITERATIONARTIFACT$_ResultFile"` (default) | string

Name of test result file, specified as a string.

The built-in tasks use tokens, like `$ITERATIONARTIFACT$`, as placeholders for dynamic path resolution during run-time. For more information, see “Dynamically Resolve Paths with Tokens”.

Data Types: string

SimulationMode — Simulation mode for running tests

`""` (default) | "Normal" | "Accelerator" | "Rapid Accelerator" | "Software-in-the-Loop" | "Processor-in-the-Loop"

Simulation mode for running tests, specified as "Normal", "Accelerator", "Rapid Accelerator", "Software-in-the-Loop", or "Processor-in-the-Loop".

By default, the property is empty (`""`), which means the built-in task uses the simulation mode that you define in the test itself. If you specify a value other than `""`, the built-in task overrides the simulation mode set in Simulink Test Manager. You do not need to update the test parameters or settings to run the test in the new mode.

The task ignores this property when your MATLAB release is earlier than R2023a or when the `UseMATLABUnitTest` is specified as `true (1)`.

Example: "Software-in-the-Loop"

UseMATLABUnitTest — Use MATLAB Unit Test framework to execute test cases

`false (0)` (default) | `true (1)`

Use MATLAB Unit Test framework to execute test cases, specified as either:

- `true (1)` — The task runs your test cases by using the MATLAB Unit Test framework to create a test runner, creates a suite of tests from your test file, and run the tests. If you use the pipeline generator, `padv.pipeline.generatePipeline`, and your pipeline generator options specify the `GenerateJUnitForProcess` property as `true (1)`, the task uses the MATLAB unit test XML plugin to produce JUnit-style XML format test results that integrate into CI platforms.

- `false (0)` — The task runs your test cases by using Simulink Test. Starting in R2023a, if you specified the task property `SimulationMode`, the task overrides the test simulation mode without having to change the test definition.

Example: `true`

Data Types: `logical`

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Run test cases using Simulink Test</p> <hr/> <p>Note You do not need to manually invoke this method. When you run a task using the Process Advisor app or the <code>runprocess</code> function, the build system automatically invokes the <code>run</code> method for the task.</p> <hr/> <p>The <code>run</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this built-in task, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = run(obj, input) ... end</pre>
dryRun	<p>Dry run the task to validate task inputs and generate representative task outputs without actually running the task. The <code>dryRun</code> method inside this built-in task runs on a task object <code>obj</code> with task input <code>input</code> and returns a task result <code>taskResult</code>. The task result is a <code>padv.TaskResult</code> object that can store the results from pass, fail, and error assessments. If you inherit from this class, make sure to use the same method signature inside your custom task:</p> <pre>function taskResult = dryRun(obj, input) ... end</pre> <p>For more information, see “Dry Run Tasks to Test Process Model”.</p>

launchToolAction	Launch Simulink Test Manager. Process Advisor uses this method when you open the tool associated with a task.
------------------	---

Examples

Add Task to Run Test Cases

Add a task to your process that can run test cases using Simulink Test.

Open the process model for your project. If you do not have a process model, open the Process Advisor app to automatically create a process model.

In the process model file, add the `RunTestsPerTestCase` task to your process model by using the `addTask` method.

```
runTestsPerTestCaseTask = pm.addTask(padv.builtin.task.RunTestsPerTestCase);
```

You can reconfigure the task behavior by using the task properties. For example, to specify a different file name for the test results:

```
runTestsPerTestCaseTask.ResultFileName = "$ITERATIONARTIFACT$_TestResultsFile";
```

If you want to generate a consolidated test results report and merged coverage reports, you can add the built-in task `MergeTestResults` to your process. By default, the built-in task `MergeTestResults` gets the current model and the outputs from the task `RunTestsPerTestCase`.

Run Specific Test Cases Based on Tags

If you want the `RunTestsPerTestCase` task to only run on test cases that have a specific test tag, specify the `IterationQuery` using the built-in query `padv.builtin.query.FindTestCasesForModel` and specify the test tag using the `Tags` argument. For example, to have the task only run on test cases that were tagged with the test tag `FeatureA`:

```
runTestsPerTestCaseTask = pm.addTask(padv.builtin.task.RunTestsPerTestCase, ...
    IterationQuery = padv.builtin.query.FindTestCasesForModel(Tags="FeatureA"));
```

Run Specific Test Cases Based on Project Labels

Suppose that you only want the `RunTestsPerTestCase` task to run for tests that use a specific project label.

By default, the `RunTestsPerTestCase` task in the default process model uses the built-in query `padv.builtin.query.FindTestCasesForModel` as the `IterationQuery`. This means that the task runs once for each test case associated with models in the project.

To run the task for tests that use a specific project label, in the process model, you can change the `IterationQuery` for the task to:

- 1 Use the built-in query `padv.builtin.query.FindTestCasesForModel` to find the models in the project
- 2 Specify the `IncludeLabel` argument of the query to only include test cases that use a specific project label. In this example, the project label is `ModelTest` and the project label category is `TestType`.

```
milTask = pm.addTask(padv.builtin.task.RunTestsPerTestCase());
```

```
% Specify which set of artifacts to run for  
milTask.IterationQuery = ...  
    padv.builtin.query.FindTestCasesForModel(...  
        IncludeLabel = {'TestType', 'ModelTest'});
```

For more information on the built-in queries, see “Find Artifacts with Queries”. If you need to perform a query that is not already covered by a built-in query, see “Create Custom Queries”.

See Also

Process Advisor | `runprocess`

Topics

“Add Tasks to Process”

“Modify Default Process Model to Fit Your Process”

“Overview of Process Model”

Built-In Query Library

padv.builtin.query.FindArtifacts Class

Namespace: padv.builtin.query padv.builtin.query

Superclasses: padv.Query

Query for finding artifacts

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.query.FindArtifacts` class provides a query that can return artifacts in your project folder. By default, the query finds all artifacts in your project folder. You can automatically include or exclude certain artifacts by using the optional name-value arguments.

You can use this query in your process model to find artifacts for your tasks to iterate over or use as inputs. If you only need to find a single file at a specific path, you can use the built-in query `padv.builtin.query.FindFileWithAddress` instead.

The `padv.builtin.query.FindArtifacts` class is a `handle` class.

Creation

Description

`query = padv.builtin.query.FindArtifacts()` creates a query for finding the artifacts in your project folder.

`query = padv.builtin.query.FindArtifacts(Name=Value)` sets certain properties using one or more name-value arguments. For example, `padv.builtin.query.FindArtifacts(InProject = true)` creates a query that only finds artifact that were explicitly added to the project.

The `padv.builtin.query.FindArtifacts` class also has other properties, but you cannot set those properties during query creation.

Input Arguments

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `query = padv.builtin.query.FindArtifacts(ExcludePath = "Control")`

ArtifactType — Type of artifact

`"sl_model_file" | "m_file" | "zc_file" | ...`

Type of artifact, specified as one or more of the values listed in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component

Category	Artifact Type	Description
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "m_file"

Example: ["sl_model_file" "zc_file"]

ExcludeLabel — Exclude artifacts with specific project label

cell array

Exclude artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Classification", "Design"}

Data Types: cell

ExcludePath — Exclude artifacts where path contains specific text

string | character vector

Exclude artifacts where the path contains specific text, specified as a string or a character vector. Consider using `ExcludePathRegex` instead.

Example: "Control"

Data Types: string

ExcludePathRegex — Exclude artifacts where path matches regular expression pattern

string | character vector

Exclude artifacts where the path matches a regular expression pattern, specified as a string or a character vector. `ExcludePathRegex` expects UNIX-style path separators.

If you want to use a literal path, use `ExcludePath` instead. You can specify either `ExcludePath` or `ExcludePathRegex` but not both.

Example: "DD_.*\\.sldd"

Data Types: char | string

FilterSubFileArtifacts — Filter out sub-file artifacts from query results

1 (true) (default) | 0 (false)

Filter out sub-file artifacts from query results, specified as a numeric or logical 1 (true) or 0 (false).

A sub-file is a part of a larger file. For example, a subsystem is a sub-file of a model file.

Example: false

Data Types: logical

IncludeLabel — Find artifacts with specific project label

cell array

Find artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Classification", "Design"}

Data Types: cell

IncludePath — Find artifacts where path contains specific text

string

Find artifacts where the path contains specific text, specified as a string. Consider using `IncludePathRegex` instead.

Example: "Control"

Data Types: string

IncludePathRegex — Find artifacts where path matches regular expression pattern

string | character vector

Find artifacts where the path matches a regular expression pattern, specified as a character vector or string. `IncludePathRegex` expects UNIX-style path separators.

If you want to use a literal path, use `IncludePath` instead. You can specify either `IncludePath` or `IncludePathRegex` but not both.

Example: "DD_.*\\.sldd"

Data Types: char | string

InProject — Include only artifacts added to project

0 (false) (default) | 1 (true)

Include only artifacts that have been added to the project, specified as a numeric or logical 1 (true) or 0 (false).

Note If you specify `InProject` as true, you cannot use the query as an input query.

Example: `true`

Data Types: `logical`

Name — Unique identifier for query

string

Unique identifier for query, specified as a string.

Example: `"FindMyArtifacts"`

Data Types: `string`

Properties

ArtifactType — Type of artifact

`"sl_model_file" | "m_file" | "zc_file" | ...`

Type of artifact, specified as one or more of the values listed in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	<code>"m_class"</code>	MATLAB class
	<code>"m_file"</code>	MATLAB file
	<code>"m_func"</code>	MATLAB function
	<code>"m_method"</code>	MATLAB class method
	<code>"m_property"</code>	MATLAB class property
Model Advisor	<code>"ma_config_file"</code>	Model Advisor configuration file
	<code>"ma_justification_file"</code>	Model Advisor justification file
Process Advisor	<code>"padv_dep_artifacts"</code>	Related artifacts that current artifact depends on
	<code>"padv_output_file"</code>	Process Advisor output file
Project	<code>"project"</code>	Current project file
Requirements	<code>"mwreq_item"</code>	Requirement (since R2024b)
	<code>"sl_req"</code>	Requirement (for R2024a and earlier)
	<code>"sl_req_file"</code>	Requirement file
	<code>"sl_req_table"</code>	Requirements Table
Stateflow	<code>"sf_chart"</code>	Stateflow chart
	<code>"sf_graphical_fcn"</code>	Stateflow graphical function
	<code>"sf_group"</code>	Stateflow group
	<code>"sf_state"</code>	Stateflow state
	<code>"sf_state_transition_chart"</code>	Stateflow state transition chart
	<code>"sf_truth_table"</code>	Stateflow truth table

Category	Artifact Type	Description
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
"sl_test_suite_result"	Simulink Test suite result	

Example: "m_file"

Example: ["sl_model_file" "zc_file"]

IncludeLabel — Find artifacts with specific project label
cell array

Find artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Classification", "Design"}

Data Types: cell

ExcludeLabel — Exclude artifacts with specific project label

cell array

Exclude artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Classification", "Design"}

Data Types: cell

IncludePathRegex — Find artifacts where path matches regular expression pattern

string | character vector

Find artifacts where the path matches a regular expression pattern, specified as a character vector or string. `IncludePathRegex` expects UNIX-style path separators.

If you want to use a literal path, use `IncludePath` instead. You can specify either `IncludePath` or `IncludePathRegex` but not both.

Example: "DD_.*\\.sldd"

Data Types: char | string

ExcludePathRegex — Exclude artifacts where path matches regular expression pattern

string | character vector

Exclude artifacts where the path matches a regular expression pattern, specified as a string or a character vector. `ExcludePathRegex` expects UNIX-style path separators.

If you want to use a literal path, use `ExcludePath` instead. You can specify either `ExcludePath` or `ExcludePathRegex` but not both.

Example: "DD_.*\\.sldd"

Data Types: char | string

FilterSubFileArtifacts — Filter out sub-file artifacts from query results

1 (true) (default) | 0 (false)

Filter out sub-file artifacts from query results, specified as a numeric or logical 1 (true) or 0 (false).

A sub-file is a part of a larger file. For example, a subsystem is a sub-file of a model file.

Example: false

Data Types: logical

InProject — Include only artifacts added to project

0 (false) (default) | 1 (true)

Include only artifacts that have been added to the project, specified as a numeric or logical 1 (true) or 0 (false).

For more information about how to add or remove files from a project, see “Manage Project Files”.

Note If you specify `InProject` as `true`, you cannot use the query as an input query.

Example: `true`

Attributes:

Dependent `true`

Data Types: `logical`

Title – Query title

`"Find all artifacts in project that meet the specified criteria"` (default) | `string` | `character vector`

Query title, specified as a string or a character vector.

Example: `"Find my artifacts"`

Data Types: `string`

DefaultArtifactType – Default artifact type returned by query

`"padv_output_file"` (default) | `"m_file"` | `"sl_model_file"` | `"zc_file"` | ...

Default artifact type returned by the query, specified as one or more of the values in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	<code>"m_class"</code>	MATLAB class
	<code>"m_file"</code>	MATLAB file
	<code>"m_func"</code>	MATLAB function
	<code>"m_method"</code>	MATLAB class method
	<code>"m_property"</code>	MATLAB class property
Model Advisor	<code>"ma_config_file"</code>	Model Advisor configuration file
	<code>"ma_justification_file"</code>	Model Advisor justification file
Process Advisor	<code>"padv_dep_artifacts"</code>	Related artifacts that current artifact depends on
	<code>"padv_output_file"</code>	Process Advisor output file
Project	<code>"project"</code>	Current project file
Requirements	<code>"mwreq_item"</code>	Requirement (since R2024b)
	<code>"sl_req"</code>	Requirement (for R2024a and earlier)
	<code>"sl_req_file"</code>	Requirement file
	<code>"sl_req_table"</code>	Requirements Table
Stateflow	<code>"sf_chart"</code>	Stateflow chart
	<code>"sf_graphical_fcn"</code>	Stateflow graphical function

Category	Artifact Type	Description
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
"sl_test_suite_result"	Simulink Test suite result	

Example: "zc_file"

Example: ["sl_model_file" "zc_file"]

Parent — Initial query that runs before iteration query

padv.Query | Name of padv.Query object

Initial query that runs before iteration query, specified as either a padv.Query object or the Name of a padv.Query object. When you specify an iteration query for a task, the parent query is the initial query that the build system runs before running the specified iteration query.

For information on how to improve Process Advisor load times by sharing query instances across your process model, see “Best Practices for Process Model Authoring”.

Example: sharedQuery

Example: "FindMyInitialArtifacts"

Name — Unique identifier for query

string

Unique identifier for query, specified as a string.

Example: "FindMyArtifacts"

Data Types: string

ShowFileExtension — Show file extensions for returned artifacts

0 (false) | 1 (true)

Show file extensions in the Alias property of returned artifacts, specified as a numeric or logical 1 (true) or 0 (false). The Alias property controls the display name for the artifact in the **Tasks** column in Process Advisor.

By default, queries strip file extensions from the Alias property of each task iteration artifact. To show file extensions for all artifacts in the **Tasks** column, select the project setting **Show file extensions**. To keep file extensions in the results for a specific query, specify the query property ShowFileExtension as true.

Example: true

Data Types: logical

SortArtifacts — Setting for automatically sorting artifacts by address

true or 1 (default) | false or 0

Setting for automatically sorting artifacts by address, specified as a numeric or logical 1 (true) or 0 (false). When a query returns artifacts, the artifacts should be in a consistent order. By default, the build system sorts artifacts by the artifact address.

Alternatively, you can sort artifacts in a different order by overriding the internal sortArtifacts method in a subclass that defines a custom sort behavior. For an example, see “Sort Artifacts in Specific Order”.

The build system automatically calls the sortArtifacts method when using the process model. The sortArtifacts method expects two input arguments: a padv.Query object and a list of padv.Artifact objects returned by the run method. The sortArtifacts method should return a list of sorted padv.Artifact objects.

Example: `SortArtifacts = false`

Data Types: `logical`

FunctionHandle — Handle to function that function-based query runs

`function_handle`

Handle to the function that a function-based query runs, specified as a `function_handle`.

If you define your query functionality inside a function and you or the build system call `run` on the query, the query runs the function specified by the `function_handle`.

The built-in queries are defined inside classes and do not use the `FunctionHandle`.

Example: `FunctionHandle = @FunctionForQuery`

Data Types: `function_handle`

Methods

Specialized Public Methods

This class overrides the following inherited methods.

<code>run</code>	<p>Run query to find the artifacts that meet the criteria specified by the query.</p> <p>The query returns a <code>padv.Artifact</code> object or an array of <code>padv.Artifact</code> objects that represent those artifacts.</p> <hr/> <p>Note You do not need to manually invoke this method inside your process model. The build system automatically invokes the <code>run</code> method to find artifacts.</p> <hr/> <p>The <code>run</code> method inside this built-in query runs on a query object <code>obj</code> and returns artifacts <code>artifacts</code>. If you inherit from this built-in query, make sure to use the same method signature inside your custom query:</p> <pre>function artifacts = run(obj,~) ... end</pre>
------------------	--

Examples

Find Data Dictionaries for Task in Process

You can use the `FindArtifacts` query in your process model to find artifacts for your tasks to iterate over (`IterationQuery`) or use as inputs (`addInputQueries`). For example, you can use the

FindArtifacts query to find the data dictionaries in your project folder and have a task run one time for each data dictionary.

Open a project. For this example, open the Process Advisor example project.

```
processAdvisorExampleStart
```

Suppose that you have a custom task, MyCustomTask, that you add to your process model. You can use the built-in query `padv.builtin.query.FindArtifacts` to find specific types of artifacts. To find the data dictionaries in the project, you specify the `ArtifactType` argument as `"sl_data_dictionary_file"`. Edit the process model to include this code:

```
taskObj = addTask(pm, "MyCustomTask",...
    IterationQuery = padv.builtin.query.FindArtifacts(...
    ArtifactType = "sl_data_dictionary_file"),...
    InputQueries = padv.builtin.query.GetIterationArtifact);
```

When you specify `InputQueries` as `padv.builtin.query.GetIterationArtifact`, that allows the task to use the artifacts returned by `IterationQuery` as inputs to the task.

In Process Advisor, view the updated **Tasks** by clicking **Refresh Tasks** and switching to the **Project** view. For the task `MyCustomTask`, there is one task iteration for each data dictionary.

Test FindArtifacts Query Outside Process Model

Although you typically use a query inside your process model, you can run an instance of the `FindArtifacts` query outside of your process model to confirm which artifacts the query returns.

Open a project. For this example, you can open the Process Advisor example project.

```
processAdvisorExampleStart
```

Create an instance of the query. You can use the arguments of the query to filter the query results. For example, you can use the `IncludeLabel` argument to have the query only return artifacts that use the `Design` project label from the `Classification` project label category.

```
q = padv.builtin.query.FindArtifacts(...
IncludeLabel = {'Classification', 'Design'});
```

Run the query and inspect the array of artifacts that the query returns.

```
q.run
```

```
ans =
```

```
1x26 Artifact array with properties:
```

```
Type
Parent
ArtifactAddress
```

Find Artifacts that Digital Thread Tracks

To identify which artifacts the digital thread can monitor for changes, you can run an instance of the `FindArtifacts` query on your project.

Open a project. For this example, you can open the Process Advisor example project.

```
processAdvisorExampleStart
```

Create an instance of the `FindArtifacts` query without specifying any additional arguments.

```
q = padv.builtin.query.FindArtifacts();
```

Run the query and inspect the array of artifacts that the query returns.

```
trackedArtifacts = q.run
```

```
trackedArtifacts =
```

```
  1x44 Artifact array with properties:
```

```
  Type
  Parent
  ArtifactAddress
  Alias
```

You can use the properties of the `padv.Artifact` objects to identify the type, address, and name of the artifact. For example, the `Alias` property stores the name of the artifact.

```
trackedArtifacts.Alias
```

Note If there is an artifact in your project that the `FindArtifacts` query does not find, the digital thread does not detect changes to that artifact.

Capabilities and Limitations

This table identifies functionality that is supported by the query.

Functionality	Supported?
Input query for task	Only when the query property <code>InProject</code> is <code>false</code> .
Iteration query for task	Yes. See <code>IterationQuery</code> .

See Also

`padv.builtin.query.FindExternalCodeCache` |
`padv.builtin.query.FindFilesWithLabel` | `padv.builtin.query.FindModels` |
`padv.builtin.query.FindModelsWithLabel` | `padv.builtin.query.FindRequirements`

Topics

“Overview of Process Model”

“Modify Default Process Model to Fit Your Process”

padv.builtin.query.FindCodeForModel Class

Namespace: padv.builtin.query padv.builtin.query

Superclasses: padv.Query

Query for finding generated code files and buildInfo.mat for model

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.query.FindCodeForModel` class provides a query that can return the generated code files and `buildInfo.mat` for a model. You can automatically include or exclude certain artifacts by using the optional name-value arguments.

You can use this query in your process model to find artifacts for your tasks to iterate over or use as inputs.

The `padv.builtin.query.FindCodeForModel` class is a `handle` class.

Creation

Description

`query = padv.builtin.query.FindCodeForModel()` creates a query for finding the generated code files and `buildInfo.mat` for a model.

`query = padv.builtin.query.FindCodeForModel(Name=Value)` sets certain properties using one or more name-value arguments. For example, `padv.builtin.query.FindCodeForModel(Name = "MyCodeQuery")` creates a query object with the name "MyCodeQuery".

The `padv.builtin.query.FindCodeForModel` class also has other properties, but you cannot set those properties during query creation.

Input Arguments

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `query = padv.builtin.query.FindCodeForModel(Name = "MyCodeQuery")`

ExcludeLabel — Exclude artifacts with specific project label

cell array

Exclude artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: `{"Category", "Label"}`

Data Types: `cell`

ExcludePath — Exclude artifacts where path contains specific text

`string` | `character vector`

Exclude artifacts where the path contains specific text, specified as a string or a character vector.

Example: `"Control"`

Data Types: `string`

IncludeLabel — Find artifacts with specific project label

`cell array`

Find artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: `{"Category", "Label"}`

Data Types: `cell`

IncludePath — Find artifacts where path contains specific text

`string`

Find artifacts where the path contains specific text, specified as a string.

Example: `"Control"`

Data Types: `string`

Name — Unique identifier for query

`string`

Unique identifier for query, specified as a string.

Example: `"FindMyCode"`

Data Types: `string`

Parent — Initial query run before iteration query

`"padv.builtin.query.FindModels"` (default) | `padv.Query object` | `Name of padv.Query object`

Initial query run before iteration query, specified as either a `padv.Query` object or the `Name` of a `padv.Query` object. When you specify a `padv.Query` object as the iteration query for a task, the `Parent` query is the initial query that the build system runs before running the specified iteration query.

The build system ignores the `Parent` query when you specify a query as an input query or dependency query for a task.

For information on how to improve Process Advisor load times by sharing query instances across your process model, see “Best Practices for Process Model Authoring”.

Example: `padv.builtin.query.FindModels`

Example: `padv.builtin.query.FindModels(IncludePath = "Control")`

Properties

IncludeLabel — Find artifacts with specific project label

cell array

Find artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Category", "Label"}

Data Types: cell

ExcludeLabel — Exclude artifacts with specific project label

cell array

Exclude artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Category", "Label"}

Data Types: cell

IncludePath — Find artifacts where path contains specific text

string

Find artifacts where the path contains specific text, specified as a string.

Example: "Control"

Data Types: string

ExcludePath — Exclude artifacts where path contains specific text

string | character vector

Exclude artifacts where the path contains specific text, specified as a string or a character vector.

Example: "Control"

Data Types: string

Title — Query title

"The generated code files for a given model" (default) | string | character vector

Query title, specified as a string or a character vector.

Example: "Find my generated code and buildInfo"

Data Types: string

DefaultArtifactType — Default artifact type returned by query

"coder_code_files" (default) | ...

Default artifact type returned by the query, specified as one or more of the values in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file

Category	Artifact Type	Description
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file

Category	Artifact Type	Description
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "coder_code_files"

Example: ["coder_code_files" "other_file"]

Parent — Initial query run before iteration query

"padv.builtin.query.FindModels" (default) | padv.Query object | Name of padv.Query object

Initial query run before iteration query, specified as either a padv.Query object or the Name of a padv.Query object. When you specify a padv.Query object as the iteration query for a task, the Parent query is the initial query that the build system runs before running the specified iteration query.

The build system ignores the Parent query when you specify a query as an input query or dependency query for a task.

For information on how to improve Process Advisor load times by sharing query instances across your process model, see "Best Practices for Process Model Authoring".

Example: padv.builtin.query.FindModels

Example: padv.builtin.query.FindModels(IncludePath = "Control")

Name — Unique identifier for query

string

Unique identifier for query, specified as a string.

Example: "FindMyCode"

Data Types: string

ShowFileExtension — Show file extensions for returned artifacts

0 (false) | 1 (true)

Show file extensions in the `Alias` property of returned artifacts, specified as a numeric or logical `1` (`true`) or `0` (`false`). The `Alias` property controls the display name for the artifact in the **Tasks** column in Process Advisor.

By default, queries strip file extensions from the `Alias` property of each task iteration artifact. To show file extensions for all artifacts in the **Tasks** column, select the project setting **Show file extensions**. To keep file extensions in the results for a specific query, specify the query property `ShowFileExtension` as `true`.

Example: `true`

Data Types: `logical`

SortArtifacts — Setting for automatically sorting artifacts by address

`true` or `1` (default) | `false` or `0`

Setting for automatically sorting artifacts by address, specified as a numeric or logical `1` (`true`) or `0` (`false`). When a query returns artifacts, the artifacts should be in a consistent order. By default, the build system sorts artifacts by the artifact address.

Alternatively, you can sort artifacts in a different order by overriding the internal `sortArtifacts` method in a subclass that defines a custom sort behavior. For an example, see “Sort Artifacts in Specific Order”.

The build system automatically calls the `sortArtifacts` method when using the process model. The `sortArtifacts` method expects two input arguments: a `padv.Query` object and a list of `padv.Artifact` objects returned by the `run` method. The `sortArtifacts` method should return a list of sorted `padv.Artifact` objects.

Example: `SortArtifacts = false`

Data Types: `logical`

FunctionHandle — Handle to function that function-based query runs

`function_handle`

Handle to the function that a function-based query runs, specified as a `function_handle`.

If you define your query functionality inside a function and you or the build system call `run` on the query, the query runs the function specified by the `function_handle`.

The built-in queries are defined inside classes and do not use the `FunctionHandle`.

Example: `FunctionHandle = @FunctionForQuery`

Data Types: `function_handle`

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Run query to find the artifacts that meet the criteria specified by the query.</p> <p>The query returns a <code>padv.Artifact</code> object or an array of <code>padv.Artifact</code> objects that represent those artifacts.</p> <hr/> <p>Note You do not need to manually invoke this method inside your process model. The build system automatically invokes the <code>run</code> method to find artifacts.</p> <hr/> <p>The <code>run</code> method inside this built-in query runs on a query object <code>obj</code> and returns artifacts <code>artifacts</code> that are associated with the artifact <code>iterationArtifact</code>. If you inherit from this built-in query, make sure to use the same method signature inside your custom query:</p> <pre>function artifacts = run(obj,iterationArtifact) ... end</pre>
-----	---

Examples

Find and Analyze Generated Code for Models

You can use the `FindCodeForModel` query in your process model to find generated code and `buildInfo.mat` files for your tasks to iterate over (`IterationQuery`) or use as inputs (`addInputQueries`). For example, you can use the `FindCodeForModel` query to find the code a code generation task, like `padv.builtin.task.GenerateCode`, generates and use those files as the input to a code analysis task like `padv.builtin.task.RunCodeInspection`.

Open a project. For this example, you can open the Process Advisor example project.

```
processAdvisorExampleStart
```

Suppose that you want to create one subprocess to contain your code generation tasks and another subprocess to contain your code analysis tasks. Your code analysis tasks need access to the generated code, but the tasks themselves cannot directly depend on the code generation task because that relationship would cross the subprocess boundary.

To pass the generated code from your code generation subprocess to your code analysis subprocess, you can update each of your code analysis tasks to find and use the generated model code as a task input by specifying `FindCodeForModel` as the input query for your code analysis tasks. Since the code analysis subprocess depends on the code that the code generation subprocess generates, you also need to specify a dependency between those subprocesses. For example, you can have the following process model:

```
function processmodel(pm)
    % Defines the project's processmodel
```

```

arguments
    pm padv.ProcessModel
end

% Add "Code Generation" subprocess
spCodeGen = pm.addSubprocess("Code Generation Tasks");
spCodeGen.addTask(padv.builtin.task.GenerateCode);

% Add "Code Analysis" subprocess
spCodeAnalysis = pm.addSubprocess("Code Analysis Tasks");
% Update task to find and use model code as an input to the task
spCodeAnalysis.addTask(padv.builtin.task.RunCodeInspection(...
    InputQueries=padv.builtin.query.FindCodeForModel()));

% Dependency between "Code Generation" and "Code Analysis" subprocesses
spCodeAnalysis.dependsOn(spCodeGen);

end

```

For information on subprocesses and subprocess boundaries, see “Group Tasks with Subprocesses”.

In Process Advisor, view the updated **Tasks** by clicking **Refresh Tasks** and switching to the **Project** view. When you run the code analysis task, the task automatically runs the code generation task and uses the generated code as an input for code analysis.

Note that in the previous example process model, the `GenerateCode` and `RunCodeInspection` tasks both use the same `FindModels` iteration query. To potentially improve process model loading times, you can share a single query object, in this case `findModels`, across the tasks.

```

function processmodel(pm)
    % Defines the project's processmodel

arguments
    pm padv.ProcessModel
end

findModels = padv.builtin.query.FindModels(Name="ModelsQuery");

% Add "Code Generation" subprocess
spCodeGen = pm.addSubprocess("Code Generation Tasks");
codeGenTask = spCodeGen.addTask(padv.builtin.task.GenerateCode(...
    IterationQuery = findModels));

% Add "Code Analysis" subprocess
spCodeAnalysis = pm.addSubprocess("Code Analysis Tasks");
% Update task to find and use model code as an input to the task
spCodeAnalysis.addTask(padv.builtin.task.RunCodeInspection(...
    IterationQuery = findModels,...
    InputQueries=padv.builtin.query.FindCodeForModel(Parent = findModels)));

% Dependency between "Code Generation" and "Code Analysis" subprocesses
spCodeAnalysis.dependsOn(spCodeGen);

end

```

For more information, see “Best Practices for Process Model Authoring”.

Capabilities and Limitations

This table identifies functionality that is supported by the query.

Functionality	Supported?
Input query for task	Yes.
Iteration query for task	Yes.

Note If you use this query as an input query and specify non-empty values for `IncludeLabel`, `ExcludeLabel`, `IncludePath`, or `ExcludePath`, your task results can unexpectedly become outdated. If you see this behavior, consider using a different query, like `padv.builtin.query.FindArtifacts`, instead. For more information and a list of queries that are not impacted by this limitation, see “Other Limitations”.

See Also

`padv.builtin.task.AnalyzeModelCode` | `padv.builtin.task.GenerateCode` |
`padv.builtin.task.RunCodeInspection`

Topics

“Best Practices for Process Model Authoring”
“Group Tasks with Subprocesses”

padv.builtin.query.FindDesignModels Class

Namespace: padv.builtin.query padv.builtin.query

Superclasses: padv.Query

Query for finding units and components

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.query.FindDesignModels` class provides a query that can return the units and components in your project. The query uses the same unit and component classification as the Model Design and Model Testing Dashboards. A unit is a functional entity in your software architecture that you can execute and test independently or as part of larger system tests. A component is an entity that integrates multiple testable units together. Some software development standards, like certain model maintainability objectives, apply to both units and components in a software architecture. You can use the `FindDesignModels` query to find the Simulink and System Composer models in your design that you need to assess. If you only need to find the units in your design, you can use the built-in query `padv.builtin.query.FindUnits` instead. For information how to classify the models in your project, see “Categorize Models in Hierarchy as Components or Units”.

You can use this query in your process model to find the units and components in your project and run tasks on those artifacts. For example, the built-in task `padv.builtin.task.CollectMetrics` can collect model maintainability metrics for the units and components in your project. The task uses `padv.builtin.query.FindDesignModels` as the iteration query to find and iterate over those units and components.

The `padv.builtin.query.FindDesignModels` class is a `handle` class.

Creation

Description

`query = padv.builtin.query.FindDesignModels()` creates a query for finding the units and components in your project.

`query = padv.builtin.query.FindDesignModels(Name=Value)` sets certain properties using one or more name-value arguments. For example, `query = padv.builtin.query.FindDesignModels(ExcludePath = "Control")` creates a query that finds the units and components in the project, but excludes units and components that have "Control" in the file address.

The `padv.builtin.query.FindDesignModels` class also has other properties, but you cannot set those properties during query creation.

Input Arguments

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `query = padv.builtin.query.FindDesignModels(ExcludePath = "Control")`

ExcludeLabel — Exclude artifacts with specific project label

cell array

Exclude artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: `{"Classification", "Design"}`

Data Types: `cell`

ExcludePath — Exclude artifacts where path contains specific text

string | character vector

Exclude artifacts where the path contains specific text, specified as a string or a character vector.

Example: `"Control"`

Data Types: `char` | `string`

IncludeLabel — Find artifacts with specific project label

cell array

Find artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: `{"Classification", "Design"}`

Data Types: `cell`

IncludePath — Find artifacts where path contains specific text

string | character vector

Find artifacts where the path contains specific text, specified as a string or a character vector.

Example: `"Control"`

Data Types: `char` | `string`

Name — Unique identifier for query

string | character vector

Unique identifier for query, specified as a string or a character vector.

Example: `"FindUnitsAndComponents"`

Data Types: `char` | `string`

Properties

IncludeLabel — Find artifacts with specific project label

cell array

Find artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Classification", "Design"}

Data Types: cell

ExcludeLabel — Exclude artifacts with specific project label

cell array

Exclude artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Classification", "Design"}

Data Types: cell

IncludePath — Find artifacts where path contains specific text

string | character vector

Find artifacts where the path contains specific text, specified as a string or a character vector.

Example: "Control"

Data Types: char | string

ExcludePath — Exclude artifacts where path contains specific text

string | character vector

Exclude artifacts where the path contains specific text, specified as a string or a character vector.

Example: "Control"

Data Types: char | string

Title — Query title

"All models under the hierarchy of Unit and Component models" (default) | string | character vector

Query title, specified as a string or a character vector.

Example: "Units and Components"

Data Types: char | string

DefaultArtifactType — Default artifact type returned by query

"sl_model_file" (default) | "zc_file" | ...

Default artifact type returned by the query, specified as one or more of the values listed in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class

Category	Artifact Type	Description
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component

Category	Artifact Type	Description
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "zc_file"

Example: ["sl_model_file" "zc_file"]

Parent — Query that build system can run first

`padv.Query object` | Name of `padv.Query` object

Query that the build system can run first, specified as either a `padv.Query` object or the Name of a `padv.Query` object.

If you use `padv.builtin.query.FindDesignModels` as an iteration query in your process model, the build system automatically runs the Parent query first. If there is an existing query that you want the build system to run first, specify that query as the Parent query. For example, the built-in query `FindModelsWithTestCases` specifies `FindModels` as a Parent query.

If you use `padv.builtin.query.FindDesignModels` as an input query or dependency query for a task, the build system ignores the Parent query.

Name — Unique identifier for query

string | character vector

Unique identifier for query, specified as a string or a character vector.

Example: "FindUnitsAndComponents"

Data Types: char | string

ShowFileExtension — Show file extensions for returned artifacts

0 (false) | 1 (true)

Show file extensions in the `Alias` property of returned artifacts, specified as a numeric or logical `1` (`true`) or `0` (`false`). The `Alias` property controls the display name for the artifact in the **Tasks** column in Process Advisor.

By default, queries strip file extensions from the `Alias` property of each task iteration artifact. To show file extensions for all artifacts in the **Tasks** column, select the project setting **Show file extensions**. To keep file extensions in the results for a specific query, specify the query property `ShowFileExtension` as `true`.

Example: `true`

Data Types: `logical`

SortArtifacts — Setting for automatically sorting artifacts by address

`true` or `1` (default) | `false` or `0`

Setting for automatically sorting artifacts by address, specified as a numeric or logical `1` (`true`) or `0` (`false`). When a query returns artifacts, the artifacts should be in a consistent order. By default, the build system sorts artifacts by the artifact address.

Alternatively, you can sort artifacts in a different order by overriding the internal `sortArtifacts` method in a subclass that defines a custom sort behavior. For an example, see “Sort Artifacts in Specific Order”.

The build system automatically calls the `sortArtifacts` method when using the process model. The `sortArtifacts` method expects two input arguments: a `padv.Query` object and a list of `padv.Artifact` objects returned by the `run` method. The `sortArtifacts` method should return a list of sorted `padv.Artifact` objects.

Example: `SortArtifacts = false`

Data Types: `logical`

FunctionHandle — Handle to function that function-based query runs

`function_handle`

Handle to the function that a function-based query runs, specified as a `function_handle`.

If you define your query functionality inside a function and you or the build system call `run` on the query, the query runs the function specified by the `function_handle`.

The built-in queries are defined inside classes and do not use the `FunctionHandle`.

Example: `FunctionHandle = @FunctionForQuery`

Data Types: `function_handle`

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Run query to find the artifacts that meet the criteria specified by the query.</p> <p>The query returns a <code>padv.Artifact</code> object or an array of <code>padv.Artifact</code> objects that represent those artifacts.</p> <hr/> <p>Note You do not need to manually invoke this method inside your process model. The build system automatically invokes the <code>run</code> method to find artifacts.</p> <hr/> <p>The <code>run</code> method inside this built-in query runs on a query object <code>obj</code> and returns artifacts <code>artifacts</code>. If you inherit from this built-in query, make sure to use the same method signature inside your custom query:</p> <pre>function artifacts = run(obj,~) ... end</pre>
-----	--

Examples

Find Design Models for Task in Process

You can use the `FindDesignModels` query in your process model to find units and components that your tasks can iterate over (`IterationQuery`). Suppose you have a custom task that you want to run for each unit and component in your project. You can find the units and components in your project by using the built-in query `FindDesignModels`.

Open a project. For this example, open the Process Advisor example project.

```
processAdvisorExampleStart
```

To have the custom task run for each unit and component, specify the query as the iteration query for the task. For example, in your process model:

```
pm.addTask("MyCustomTask", ...
    IterationQuery = padv.builtin.query.FindDesignModels);
```

In Process Advisor, view the updated **Tasks** by clicking **Refresh Tasks** and switching to the **Project** view. For the task `MyCustomTask`, there is one task iteration for each unit and component in the project. The Process Advisor example project contains the component `Flight_Control` and four units `AHRS_Voter`, `Actuator_Control`, `InnerLoop_Control`, and `OuterLoop_Control`.

To view how the example project is classifying units and components, open the options for the Model Testing Dashboard.

```
modelTestingDashboard
```

In the dashboard toolstrip, click **Options**. The **Classification** section shows that the digital thread classifies models with the project label `Software Component` as components and models with the project label `Software Unit` as units.

For information how to classify the models in your project, see “Categorize Models in Hierarchy as Components or Units”.

Test Query Locally in Command Window

If you want to test a query before using the query in your process model, you can run the query directly from the MATLAB Command Window.

Open a project. For this example, open the Process Advisor example project.

```
processAdvisorExampleStart
```

In the MATLAB Command Window, create a query object that represents the query.

```
q = padv.builtin.query.FindDesignModels;
```

Run the query by using the `run` method. The query returns the artifacts that it finds as a `padv.Artifact` object or an array of `padv.Artifact` objects.

```
artifacts = q.run
```

```
artifacts =
```

```
1x5 Artifact array with properties:
```

```
Type
Parent
ArtifactAddress
Alias
```

You can inspect the properties of each `padv.Artifact` object to view information about the artifact. For example, you can use the `Alias` property to view the artifact names.

```
artifacts.Alias
```

```
ans =
```

```
"Flight_Control.slx"
```

```
ans =
```

```
"AHRS_Voter.slx"
```

```
ans =
```

```
"Actuator_Control.slx"
```

```
ans =
```

```
"OuterLoop_Control.slx"
```

```
ans =
```

```
"InnerLoop_Control.slx"
```

The `Alias` property returns the artifact names as they appear in Process Advisor.

Capabilities and Limitations

This table identifies functionality that is supported by the query.

Functionality	Supported?
Input query for task	No.
Iteration query for task	Yes. See <code>IterationQuery</code> .

Note If you use this query as an input query and specify non-empty values for `IncludeLabel`, `ExcludeLabel`, `IncludePath`, or `ExcludePath`, your task results can unexpectedly become outdated. If you see this behavior, consider using a different query, like `padv.builtin.query.FindArtifacts`, instead. For more information and a list of queries that are not impacted by this limitation, see “Other Limitations”.

See Also

`padv.Artifact` | `padv.builtin.task.CollectMetrics` | `padv.builtin.query.FindUnits`

Topics

“Categorize Models in Hierarchy as Components or Units”

padv.builtin.query.FindExternalCodeCache Class

Namespace: padv.builtin.query padv.builtin.query

Superclasses: padv.Query, padv.builtin.query.FindArtifacts

Query for finding external code cache files in project folder

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.query.FindExternalCodeCache` class provides a query that can return the external code cache files (`.slxc.bk`) in your project folder. You can automatically include or exclude certain files by using the optional name-value arguments.

You can use this query in your process model to find artifacts for your tasks to use as inputs. For example, you can use this query to find external code cache files that you generate using the built-in task `padv.builtin.task.GenerateCode`. The built-in task `padv.builtin.task.GenerateCode` generates an external code cache when you specify the task property `GenerateExternalCodeCache` as `true`. To unpack the code generation target from the cache files, you can use the utility function `padv.util.unpackExternalCodeCache`.

The `padv.builtin.query.FindExternalCodeCache` class is a `handle` class.

Creation

Description

`query = padv.builtin.query.FindExternalCodeCache()` creates a query for finding the external code cache files (`.slxc.bk`) in your project folder.

`query = padv.builtin.query.FindExternalCodeCache(Name = queryName)` specifies a new name, `queryName`, for the query object. Each query in the process model must have a unique name.

The `padv.builtin.query.FindExternalCodeCache` class also has other properties, but you cannot set those properties during query creation.

Input Arguments

queryName — Unique identifier for query

string

Unique identifier for query, specified as a string.

This argument specifies the value for the `Name` property.

Example: "FindMyExternalCodeCache"

Data Types: string

Properties

ArtifactType — Type of artifact

"slxc_bak_file" (default) | ...

Type of artifact, specified as one or more of the values in this table. To specify multiple values, use a cell array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file

Category	Artifact Type	Description
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "slxc_bak_file"

IncludeLabel – Find artifacts with specific project label

cell array

Find artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Classification", "Design"}

Data Types: cell

ExcludeLabel – Exclude artifacts with specific project label

cell array

Exclude artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Classification", "Design"}

Data Types: cell

IncludePathRegex — Find artifacts where path matches regular expression pattern

string | character vector

Find artifacts where the path matches a regular expression pattern, specified as a character vector or string. `IncludePathRegex` expects UNIX-style path separators.

If you want to use a literal path, use `IncludePath` instead. You can specify either `IncludePath` or `IncludePathRegex` but not both.

Example: "modelNamePrefix_.*\s\lxc.bk"

Data Types: char | string

ExcludePathRegex — Exclude artifacts where path matches regular expression pattern

string | character vector

Exclude artifacts where the path matches a regular expression pattern, specified as a string or a character vector. `ExcludePathRegex` expects UNIX-style path separators.

If you want to use a literal path, use `ExcludePath` instead. You can specify either `ExcludePath` or `ExcludePathRegex` but not both.

Example: "modelNamePrefix_.*\s\lxc.bk"

Data Types: char | string

FilterSubFileArtifacts — Filter out sub-file artifacts from query results

1 (true) (default) | 0 (false)

Filter out sub-file artifacts from query results, specified as a numeric or logical 1 (true) or 0 (false).

A sub-file is a part of a larger file. For example, a subsystem is a sub-file of a model file.

Example: false

Data Types: logical

InProject — Include only artifacts added to project

0 (false) (default) | 1 (true)

Include only artifacts that have been added to the project, specified as a numeric or logical 1 (true) or 0 (false).

For more information about how to add or remove files from a project, see “Manage Project Files”.

Example: true

Data Types: logical

Title — Query title

"All generated external code cache files" (default) | string | character vector

Query title, specified as a string or a character vector.

Example: "Find my external code cache files"

Data Types: string

DefaultArtifactType – Default artifact type returned by query

"slxc_bak_file" (default) | ...

Default artifact type returned by the query, specified as one or more of the values in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file

Category	Artifact Type	Description
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "slxc_bak_file"

Parent — Initial query that runs before iteration query

string.empty() (default) | padv.Query | Name of padv.Query object

You cannot use FindExternalCodeCache as an iteration query, so the build system does not use the parent query.

Name — Unique identifier for query

string

Unique identifier for query, specified as a string.

Example: "FindMyExternalCodeCache"

Data Types: string

ShowFileExtension — Show file extensions for returned artifacts

0 (false) | 1 (true)

Show file extensions in the Alias property of returned artifacts, specified as a numeric or logical 1 (true) or 0 (false). The Alias property controls the display name for the artifact in the **Tasks** column in Process Advisor.

By default, queries strip file extensions from the `Alias` property of each task iteration artifact. To show file extensions for all artifacts in the **Tasks** column, select the project setting **Show file extensions**. To keep file extensions in the results for a specific query, specify the query property `ShowFileExtension` as `true`.

Example: `true`

Data Types: `logical`

SortArtifacts — Setting for automatically sorting artifacts by address

`true` or `1` (default) | `false` or `0`

Setting for automatically sorting artifacts by address, specified as a numeric or logical `1` (`true`) or `0` (`false`). When a query returns artifacts, the artifacts should be in a consistent order. By default, the build system sorts artifacts by the artifact address.

Alternatively, you can sort artifacts in a different order by overriding the internal `sortArtifacts` method in a subclass that defines a custom sort behavior. For an example, see “Sort Artifacts in Specific Order”.

The build system automatically calls the `sortArtifacts` method when using the process model. The `sortArtifacts` method expects two input arguments: a `padv.Query` object and a list of `padv.Artifact` objects returned by the `run` method. The `sortArtifacts` method should return a list of sorted `padv.Artifact` objects.

Example: `SortArtifacts = false`

Data Types: `logical`

FunctionHandle — Handle to function that function-based query runs

`function_handle`

Handle to the function that a function-based query runs, specified as a `function_handle`.

If you define your query functionality inside a function and you or the build system call `run` on the query, the query runs the function specified by the `function_handle`.

The built-in queries are defined inside classes and do not use the `FunctionHandle`.

Example: `FunctionHandle = @FunctionForQuery`

Data Types: `function_handle`

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Run query to find the artifacts that meet the criteria specified by the query.</p> <p>The query returns a <code>padv.Artifact</code> object or an array of <code>padv.Artifact</code> objects that represent those artifacts.</p> <hr/> <p>Note You do not need to manually invoke this method inside your process model. The build system automatically invokes the <code>run</code> method to find artifacts.</p> <hr/> <p>The <code>run</code> method inside this built-in query runs on a query object <code>obj</code> and returns artifacts <code>artifacts</code>. If you inherit from this built-in query, make sure to use the same method signature inside your custom query:</p> <pre>function artifacts = run(obj,~) ... end</pre>
-----	--

Examples

Find and Unpack Cache Files for Task in Process

You can use the `FindExternalCodeCache` query in your task definition to find and unpack external code cache files. For example, suppose your team generates code in parallel by generating an external code cache, downstream tasks that depend on the generated code need to unpack the generated code target before performing the main task action. If you have a custom task that depends on that generated code, you can find the external code cache files by using the built-in query `FindExternalCodeCache` and unpack the code generation target by using the utility function `padv.util.unpackExternalCodeCache`.

Inside your task definition, you can find external code cache files by creating and running a `FindExternalCodeCache` query object. For example:

```
% Before main task action, access the generated code
% by finding and unpacking the external code cache
q = padv.builtin.query.FindExternalCodeCache;
artifactsArray = run(q);
    if ~isempty(artifactsArray)
        padv.util.unpackExternalCodeCache(artifactsArray)
    end

% <definition for main task action that uses the generated code>
```

For more information about parallel code generation and external code caches, see the `GenerateExternalCodeCache` property for the built-in task `padv.builtin.task.GenerateCode`.

The external code cache allows your team to generate code in parallel while maintaining up-to-date task results.

Test FindExternalCodeCache Query Outside Process Model

Although you typically use a query inside your process model, you can run an instance of the FindExternalCodeCache query outside of your process model to confirm which artifacts the query returns.

Open the parallel code generation example.

```
processAdvisorParallelExampleStart
```

Generate code by running a code generation task iteration. For example, run the code generation task on the reference model OuterLoop_Control.

```
runprocess(Tasks = "padv.builtin.task.GenerateCode", ...
  FilterArtifact = fullfile("02_Models", "OuterLoop_Control", ...
    "specification", "OuterLoop_Control.slx"));
```

Find the external code cache file by using the built-in query.

```
q = padv.builtin.query.FindExternalCodeCache;
artifactsArray = run(q);
```

Unpack the cache file.

```
padv.util.unpackExternalCodeCache(artifactsArray);
```

Capabilities and Limitations

This table identifies functionality that is supported by the query.

Functionality	Supported?
Input query for task	Yes.
Iteration query for task	No.

See Also

padv.builtin.task.GenerateCode | padv.util.unpackExternalCodeCache

padv.builtin.query.FindFilesWithLabel Class

Namespace: padv.builtin.query padv.builtin.query

Superclasses: padv.Query, padv.builtin.query.FindArtifacts

Query for finding files with project label

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.query.FindFilesWithLabel` class provides a query that can return files that use the specified project label. You can automatically include or exclude certain files by using the optional name-value arguments. If you do not need to specify a project label, you can use a built-in query like `padv.builtin.query.FindFileWithAddress` or `padv.builtin.query.FindArtifacts` instead.

You can use this query in your process model to find files for your tasks to iterate over.

The `padv.builtin.query.FindFilesWithLabel` class is a `handle` class.

Creation

Description

`query = padv.builtin.query.FindFilesWithLabel(categoryName, labelName)` creates a query for finding files that use the project label `labelName` from the project label category `categoryName`. For more information about project labels, see “Add Labels to Project Files”.

`query = padv.builtin.query.FindFilesWithLabel(____, Name=Value)` sets certain properties using one or more name-value arguments. For example, `padv.builtin.query.FindFilesWithLabel("Classification", "Design", Name = "FindMyLabeledFiles")` creates a query with the name “FindMyLabeledFiles”.

The `padv.builtin.query.FindFilesWithLabel` class also has other properties, but you cannot set those properties during query creation.

Input Arguments

categoryName — Name of category for project label

string | character vector

Name of the category for the project label, specified as a string or a character vector.

The query uses `categoryName` to specify the first entry for the query property `IncludeLabel`. For more information about project labels, see “Add Labels to Project Files”.

Example: “Classification”

Data Types: char | string

LabelName — Name of project label

string | character vector

Name of project label, specified as a string or a character vector.

The query uses `labelName` to specify the second entry for the query property `IncludeLabel`. For more information about project labels, see “Add Labels to Project Files”.

Example: "Design"

Data Types: char | string

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: query =
`padv.builtin.query.FindFilesWithLabel("Classification","Design",Name = "FindMyLabeledFiles")`

ExcludeLabel — Exclude files with specific project label

cell array

Exclude files with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

For more information about project labels, see “Add Labels to Project Files”.

Example: {"Classification","Design"}

Data Types: cell

ExcludePath — Exclude files where path contains specific text

string | character vector

Exclude files where the path contains specific text, specified as a string or a character vector. Consider using `ExcludePathRegex` instead.

Example: "Control"

Data Types: string

ExcludePathRegex — Exclude files where path matches regular expression pattern

string | character vector

Exclude files where the path matches a regular expression pattern, specified as a string or a character vector. `ExcludePathRegex` expects UNIX-style path separators.

If you want to use a literal path, use `ExcludePath` instead. You can specify either `ExcludePath` or `ExcludePathRegex` but not both.

Data Types: char | string

IncludeLabel — Find files with specific project label

cell array

Find files with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Alternatively, you can specify the first and second entries using the arguments `categoryName` and `labelName`. The query uses those arguments to specify the entries for `IncludeLabel`. For more information about project labels, see “Add Labels to Project Files”.

Example: {"Classification", "Design"}

Data Types: cell

IncludePath — Find files where path contains specific text

string | character vector

Find files where the path contains specific text, specified as a string. Consider using `IncludePathRegex` instead.

Example: "Control"

Data Types: char | string

IncludePathRegex — Find files where path matches regular expression pattern

string | character vector

Find files where the path matches a regular expression pattern, specified as a character vector or string. `IncludePathRegex` expects UNIX-style path separators.

If you want to use a literal path, use `IncludePath` instead. You can specify either `IncludePath` or `IncludePathRegex` but not both.

Data Types: char | string

Name — Unique identifier for query

string | character vector

Unique identifier for query, specified as a string or a character vector.

Example: "FindMyLabeledFiles"

Data Types: char | string

Properties

ArtifactType — Type of artifact

"m_file" | "other_file" | ...

Type of artifact, specified as one or more of the values in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method

Category	Artifact Type	Description
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram

Category	Artifact Type	Description
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "m_file"

Example: ["m_file" "other_file"]

IncludeLabel — Find files with specific project label

cell array

Find files with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Alternatively, you can specify the first and second entries using the arguments `categoryName` and `labelName`. The query uses those arguments to specify the entries for `IncludeLabel`. For more information about project labels, see “Add Labels to Project Files”.

Example: {"Classification", "Design"}

Data Types: cell

ExcludeLabel — Exclude files with specific project label

cell array

Exclude files with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

For more information about project labels, see “Add Labels to Project Files”.

Example: {"Classification", "Design"}

Data Types: cell

IncludePathRegex — Find files where path matches regular expression pattern

string | character vector

Find files where the path matches a regular expression pattern, specified as a string or character vector. `IncludePathRegex` expects UNIX-style path separators.

If you want to use a literal path, use `IncludePath` instead. You can specify either `IncludePath` or `IncludePathRegex` but not both.

Data Types: `char` | `string`

ExcludePathRegex — Exclude files where path matches regular expression pattern

`string` | `character vector`

Exclude files where the path matches a regular expression pattern, specified as a string or a character vector. `ExcludePathRegex` expects UNIX-style path separators.

If you want to use a literal path, use `ExcludePath` instead. You can specify either `ExcludePath` or `ExcludePathRegex` but not both.

Data Types: `char` | `string`

FilterSubFileArtifacts — Filter out sub-file artifacts from query results

`1 (true) (default)` | `0 (false)`

Filter out sub-file artifacts from query results, specified as a numeric or logical `1 (true)` or `0 (false)`.

A sub-file is a part of a larger file. For example, a subsystem is a sub-file of a model file.

Example: `false`

Data Types: `logical`

InProject — Include only files added to project

`0 (false) (default)` | `1 (true)`

Include only files that have been added to the project, specified as a numeric or logical `1 (true)` or `0 (false)`.

For more information about how to add or remove files from a project, see “Manage Project Files”.

Example: `true`

Attributes:

Dependent

Data Types: `logical`

Title — Query title

`"All files with label ''"` (default) | `string` | `character vector`

Query title, specified as a string or a character vector.

When you specify a project label name, the query automatically updates the `Title` to include that project label name. For example, `"All files with label 'Design'"`.

Example: `"Find my labeled files"`

Data Types: `string`

DefaultArtifactType — Default artifact type returned by query

`"padv_output_file"` (default) | `"m_file"` | `"other_file"` | ...

Default artifact type returned by the query, specified as one or more of the values in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture

Category	Artifact Type	Description
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "m_file"

Example: ["m_file" "other_file"]

Parent – Initial query that runs before iteration query

padv.Query | Name of padv.Query object

Initial query that runs before iteration query, specified as either a padv.Query object or the Name of a padv.Query object. When you specify an iteration query for a task, the parent query is the initial query that the build system runs before running the specified iteration query.

For information on how to improve Process Advisor load times by sharing query instances across your process model, see “Best Practices for Process Model Authoring”.

Example: sharedQuery

Example: "FindMyInitialArtifacts"

ShowFileExtension – Show file extensions for returned files

0 (false) | 1 (true)

Show file extensions in the Alias property of returned files, specified as a numeric or logical 1 (true) or 0 (false). The Alias property controls the display name for the artifact in the **Tasks** column in Process Advisor.

By default, queries strip file extensions from the Alias property of each task iteration artifact. To show file extensions for all artifacts in the **Tasks** column, select the project setting **Show file**

extensions. To keep file extensions in the results for a specific query, specify the query property `ShowFileExtension` as `true`.

Example: `true`

Data Types: `logical`

SortArtifacts — Setting for automatically sorting artifacts by address

`true` or `1` (default) | `false` or `0`

Setting for automatically sorting artifacts by address, specified as a numeric or logical `1` (`true`) or `0` (`false`). When a query returns artifacts, the artifacts should be in a consistent order. By default, the build system sorts artifacts by the artifact address.

Alternatively, you can sort artifacts in a different order by overriding the internal `sortArtifacts` method in a subclass that defines a custom sort behavior. For an example, see “Sort Artifacts in Specific Order”.

The build system automatically calls the `sortArtifacts` method when using the process model. The `sortArtifacts` method expects two input arguments: a `padv.Query` object and a list of `padv.Artifact` objects returned by the `run` method. The `sortArtifacts` method should return a list of sorted `padv.Artifact` objects.

Example: `SortArtifacts = false`

Data Types: `logical`

FunctionHandle — Handle to function that function-based query runs

`function_handle`

Handle to the function that a function-based query runs, specified as a `function_handle`.

If you define your query functionality inside a function and you or the build system call `run` on the query, the query runs the function specified by the `function_handle`.

The built-in queries are defined inside classes and do not use the `FunctionHandle`.

Example: `FunctionHandle = @FunctionForQuery`

Data Types: `function_handle`

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Run query to find the artifacts that meet the criteria specified by the query.</p> <p>The query returns a <code>padv.Artifact</code> object or an array of <code>padv.Artifact</code> objects that represent those artifacts.</p> <hr/> <p>Note You do not need to manually invoke this method inside your process model. The build system automatically invokes the <code>run</code> method to find artifacts.</p> <hr/> <p>The <code>run</code> method inside this built-in query runs on a query object <code>obj</code> and returns artifacts <code>artifacts</code>. If you inherit from this built-in query, make sure to use the same method signature inside your custom query:</p> <pre>function artifacts = run(obj,~) ... end</pre>
-----	--

Examples

Find Files with Specific Project Label

Suppose that you have a project that contains several MATLAB scripts and you want a custom task to only run for MATLAB script files with a specific project label.

In your process model, you can use the `FindFilesWithLabel` query in your task definition to find files that your tasks can iterate over (`IterationQuery`). For example, if the scripts that you want to run the task on use the project label `ProjectTooling` from the project label category `Tools`:

```
taskObj = addTask(pm, "MyCustomTask",...
    IterationQuery = padv.builtin.query.FindFilesWithLabel(...
    "Tools", "ProjectTooling"),...
    InputQueries = padv.builtin.query.GetIterationArtifact);
```

When you specify `InputQueries` as `padv.builtin.query.GetIterationArtifact`, that allows the task to use the artifacts returned by `IterationQuery` as inputs to the task.

In Process Advisor, view the updated **Tasks** by clicking **Refresh Tasks** and switching to the **Project** view. For the task `MyCustomTask`, there is one task iteration for each file with the `ProjectTooling` project label.

Test FindFilesWithLabel Query Outside Process Model

Although you typically use a query inside your process model, you can run an instance of the `FindFilesWithLabel` query outside of your process model to confirm which artifacts the query returns.

Open a project. For this example, you can open the Process Advisor example project.

```
processAdvisorExampleStart
```

Create an instance of the query. You can use the arguments of the query to filter the query results. For example, to find files that use the Design project label from the Classification project label category:

```
q = padv.builtin.query.FindFilesWithLabel("Classification","Design");
```

Run the query and inspect the array of artifacts that the query returns.

```
run(q)
```

```
ans =
```

```
1x24 Artifact array with properties:
```

```
Type
Parent
ArtifactAddress
Alias
```

The `ArtifactAddress` property contains the address of the file. If you only need to find a specific file at a specific address, you can use the built-in query `padv.builtin.query.FindFileWithAddress` instead.

Capabilities and Limitations

This table identifies functionality that is supported by the query.

Functionality	Supported?
Input query for task	No.
Iteration query for task	Yes.

See Also

`padv.builtin.query.FindArtifacts` | `padv.builtin.query.FindFileWithAddress`

padv.builtin.query.FindFileWithAddress Class

Namespace: padv.builtin.query padv.builtin.query

Superclasses: padv.Query

Query for finding file with address

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.query.FindFileWithAddress` class provides a query that can return the file at the specified address in the project. You can specify additional settings by using the optional name-value arguments.

You can use this query in your process model to find artifacts for your tasks to iterate over or use as inputs. For example, you can use this query to find a Model Advisor configuration file to use with the built-in task `padv.builtin.task.RunModelStandards`.

The `padv.builtin.query.FindFileWithAddress` class is a `handle` class.

Creation

Description

`q = padv.builtin.query.FindFileWithAddress(Type = ArtifactType, Path = FilePath)` finds a file, of type `ArtifactType`, at the address specified by `FilePath`.

`q = padv.builtin.query.FindFileWithAddress(____, Name=Value)` finds and returns a file using the settings specified by one or more name-value arguments. For example, if you do not want the build system to track changes to the returned file, specify `TrackArtifacts=false`.

The `padv.builtin.query.FindFileWithAddress` class also has other properties, but you cannot set those properties during query creation.

Input Arguments

ArtifactType — Type of artifact

"padv_output_file" (default) | "ma_config_file" | "m_file" | ...

Type of artifact, specified as one or more of the values in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method

Category	Artifact Type	Description
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram

Category	Artifact Type	Description
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

The `Type` argument controls the `DefaultArtifactType` property value.

Example: "ma_config_file"

Example: ["sl_model_file" "m_file"]

FilePath — Path to file

string | character vector | cell array of character vectors

Path to file, specified as a string, a character vector, or a cell array of character vectors.

Example: `fullfile("02_Models", "AHRV_Voter", "specification", "AHRV_Voter.slx")`

Example: `[fullfile("myFiles", "myModel.slx"), fullfile("myFiles", "myScript.m")]`

Data Types: char | string | cell

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `query = padv.builtin.query.FindCodeForModel(Name = "MyCodeQuery")`

Name — Unique identifier for query

string

Unique identifier for query, specified as a string.

Example: "FindMyFile"

Data Types: string

Type — Type of artifact

"padv_output_file" (default) | "ma_config_file" | "m_file" | ...

Type of artifact, specified as one or more of the values in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture

Category	Artifact Type	Description
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
"sl_test_suite_result"	Simulink Test suite result	

The `Type` argument controls the `DefaultArtifactType` property value.

Example: "ma_config_file"

Example: ["sl_model_file" "m_file"]

Path — Path to file

string | character vector | cell array of character vectors

Path to file, specified as a string, a character vector, or a cell array of character vectors.

Example: `fullfile("tools", "sampleChecks.json")`

Example: `[fullfile("myFiles", "myModel.slx"), fullfile("myFiles", "myScript.m")]`

Data Types: char | string | cell

ValidateFileExistence — Validate that file exists before returning in query results

1 (true) (default) | 0 (false)

Validate that the file exists before attempting to return the file in the query results, specified as a numeric or logical 1 (true) or 0 (false).

Example: false

Data Types: logical

TrackArtifacts — Setting that controls whether build system tracks changes to file the query returns

1 (true) (default) | 0 (false)

Setting that controls whether the build system tracks changes to the file the query returns, specified as a numeric or logical 1 (true) or 0 (false).

For more information, see “Turn Off Change Tracking for File” on page 8-63 and “Exclude Files from Change Tracking in Process Advisor”.

Note If you specify `TrackArtifacts=false`, you can no longer use that query object as an iteration query. The build system needs to track changes iteration artifacts to identify the iterations for the task.

Example: false

Data Types: logical

Properties**Path — Path to file**

string | character vector | cell array of character vectors

Path to file, specified as a string, a character vector, or a cell array of character vectors.

Example: `fullfile("tools","sampleChecks.json")`

Example: `[fullfile("myFiles","myModel.slx"), fullfile("myFiles","myScript.m")]`

Data Types: char | string | cell

ValidateFileExistence — Validate that file exists before returning in query results

1 (true) (default) | 0 (false)

Validate that the file exists before attempting to return the file in the query results, specified as a numeric or logical 1 (true) or 0 (false).

Example: false

Data Types: logical

TrackArtifacts — Setting that controls whether build system tracks changes to file the query returns

1 (true) (default) | 0 (false)

Setting that controls whether the build system tracks changes to the file the query returns, specified as a numeric or logical 1 (true) or 0 (false).

For more information, see “Turn Off Change Tracking for File” on page 8-63.

Note If you specify `TrackArtifacts=false`, you can no longer use that query object as an iteration query. The build system needs to track changes iteration artifacts to identify the iterations for the task.

Example: false

Data Types: logical

Title – Query title

"Files: ''" (default) | string | character vector

Query title, specified as a string or a character vector.

Example: "Files: 'tools\sampleChecks.json'"

Data Types: string

DefaultArtifactType – Type of artifact

"padv_output_file" (default) | "ma_config_file" | "m_file" | ...

Type of artifact, specified as one or more of the values in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram

Category	Artifact Type	Description
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

The Type argument controls the DefaultArtifactType property value.

Example: "ma_config_file"

Example: ["sl_model_file" "m_file"]

Parent — Initial query run before iteration query

padv.Query object | Name of padv.Query object

Initial query run before iteration query, specified as either a `padv.Query` object or the Name of a `padv.Query` object. When you specify a `padv.Query` object as the iteration query for a task, the Parent query is the initial query that the build system runs before running the specified iteration query.

The build system ignores the Parent query when you specify a query as an input query or dependency query for a task.

For information on how to improve Process Advisor load times by sharing query instances across your process model, see “Best Practices for Process Model Authoring”.

Example: `padv.builtin.query.FindModels`

Example: `padv.builtin.query.FindModels(IncludePath = "Control")`

Name — Unique identifier for query

string

Unique identifier for query, specified as a string.

Example: "FindMyFile"

Data Types: string

ShowFileExtension — Show file extensions for returned artifacts

0 (false) | 1 (true)

Show file extensions in the Alias property of returned artifacts, specified as a numeric or logical 1 (true) or 0 (false). The Alias property controls the display name for the artifact in the **Tasks** column in Process Advisor.

By default, queries strip file extensions from the Alias property of each task iteration artifact. To show file extensions for all artifacts in the **Tasks** column, select the project setting **Show file extensions**. To keep file extensions in the results for a specific query, specify the query property `ShowFileExtension` as true.

Example: true

Data Types: logical

SortArtifacts — Setting for automatically sorting artifacts by address

true or 1 (default) | false or 0

Setting for automatically sorting artifacts by address, specified as a numeric or logical 1 (true) or 0 (false). When a query returns artifacts, the artifacts should be in a consistent order. By default, the build system sorts artifacts by the artifact address.

Alternatively, you can sort artifacts in a different order by overriding the internal `sortArtifacts` method in a subclass that defines a custom sort behavior. For an example, see “Sort Artifacts in Specific Order”.

The build system automatically calls the `sortArtifacts` method when using the process model. The `sortArtifacts` method expects two input arguments: a `padv.Query` object and a list of `padv.Artifact` objects returned by the run method. The `sortArtifacts` method should return a list of sorted `padv.Artifact` objects.

Example: `SortArtifacts = false`

Data Types: `logical`

FunctionHandle — Handle to function that function-based query runs

`function_handle`

Handle to the function that a function-based query runs, specified as a `function_handle`.

If you define your query functionality inside a function and you or the build system call `run` on the query, the query runs the function specified by the `function_handle`.

The built-in queries are defined inside classes and do not use the `FunctionHandle`.

Example: `FunctionHandle = @FunctionForQuery`

Data Types: `function_handle`

Methods

Specialized Public Methods

This class overrides the following inherited methods.

<code>run</code>	<p>Run query to find the artifacts that meet the criteria specified by the query.</p> <p>The query returns a <code>padv.Artifact</code> object or an array of <code>padv.Artifact</code> objects that represent those artifacts.</p> <hr/> <p>Note You do not need to manually invoke this method inside your process model. The build system automatically invokes the <code>run</code> method to find artifacts.</p> <hr/> <p>The <code>run</code> method inside this built-in query runs on a query object <code>obj</code> and returns artifacts <code>artifacts</code>. If you inherit from this built-in query, make sure to use the same method signature inside your custom query:</p> <pre style="margin-left: 20px;">function artifacts = run(obj,~) ... end</pre>
------------------	---

Examples

Find and Use Model Advisor Configuration File

By default, the `RunModelStandards` task runs a subset of high-integrity checks. If you want the task to run the Model Advisor checks specified by the Model Advisor configuration file, you can add the configuration file as an input to the task.

Open a project. For this example, you can open the Process Advisor example project.

```
processAdvisorExampleStart
```

Edit the process model to use the following process model instead.

```
function processmodel(pm)
    % Defines the project's processmodel

    arguments
        pm padv.ProcessModel
    end

    % Add built-in task for Checking Modeling Standards
    maTask = pm.addTask(padv.builtin.task.RunModelStandards);

    % Reconfigure task to specify which Model Advisor configuration to use
    maTask.addInputQueries(padv.builtin.query.FindFileWithAddress( ...
        Type = "ma_config_file",...
        Path = fullfile("tools","sampleChecks.json")));

end
```

This process model finds the Model Advisor configuration file by using the query FindFileWithAddress and then specifies that query as the input query for the built-in task RunModelStandards by using the addInputQueries function.

For the FindFileWithAddress query:

- The first argument, "ma_config_file", specifies that the file is a Model Advisor configuration file.
- The second argument specifies the path to the Model Advisor configuration file. In this example, the configuration file is a file, sampleChecks.json, in the tools folder in the project.

Find Multiple Files

To find multiple files by their addresses, you can use vectors of the same length.

For example, to find a Model Advisor configuration file and a model file by using the same query, specify the artifact type (Type) and the file path (Path) using vectors of the same length. For example:

```
padv.builtin.query.FindFileWithAddress(...
    Type=["ma_config_file", "sl_model_file"],...
    Path=[fullfile("tools","sampleChecks.json"),...
        fullfile("02_Models","AHRS_Voter","specification","AHRS_Voter.slx")])
```

If you only specify one value for Type, the query uses the same artifact type for each specified file specified by Path. For example, the following query finds two Model Advisor configuration files.

```
padv.builtin.query.FindFileWithAddress(...
    Type="ma_config_file",...
    Path=[fullfile("tools","sampleChecks.json"), fullfile("tools","myCustomChecks.json")])
```

Turn Off Change Tracking for File

If you do not want the build system to mark a task as outdated when you make changes to a specific input artifact, you can turn off change tracking for that artifact by specifying the name-value argument `TrackArtifacts` as `false` when you use the built-in query `FindFileWithAddress` to find the artifact.

Open a project. For this example, you can open the Process Advisor example project.

```
processAdvisorExampleStart
```

Edit the process model to use the following process model instead. This process model code turns off change tracking for a Model Advisor configuration file, `myMAConfig.json`, that is an input for the **Check Modeling Standards** task.

```
function processmodel(pm)
    % Defines the project's processmodel

    arguments
        pm padv.ProcessModel
    end

    % Add built-in task for Checking Modeling Standards
    maTask = pm.addTask(padv.builtin.task.RunModelStandards);
    % Reconfigure task to specify which Model Advisor configuration to use
    % but turn change tracking off for the configuration file
    maTask.addInputQueries(...
        padv.builtin.query.FindFileWithAddress( ...
            Type='ma_config_file', Path=which('myMAConfig.json'), ...
            TrackArtifacts=false));
end
```

When you run the task, the Process Advisor **I/O** column shows the outputs as **Untracked**. If you make a change to an untracked file, the build system does not mark the task as outdated.

Tasks	I/O	Details
▼ ● Check Modeling Standards	📄	✓ 3 ⚠ 2
✓ AHRs_Voter	📄	✓ 3 ⚠ 2
○ Actuator_Control		
○ Flight_Control		
○ InnerLoop_Control		
○ OuterLoop_Control		
▼ ○ Detect Design Errors		
○ AHRs_Voter		
○ Actuator_Control		
○ Flight_Control		
○ InnerLoop_Control		
○ OuterLoop_Control		

Outputs:

AHRs_Voter_ModelAdvisor.html

Inputs:

AHRs_Voter

myMAConfig.json ⚠

Dependencies:

AHRs_Voter

bus_types

csSingleInstance

DD_AHRs_Voter

processmodel.m

Untracked

By default, the build system marks files outside the project as **Untracked** because you cannot track changes to files outside the project. The change tracking setting for the artifact is stored in the `Track` property in the `ArtifactAddress` property for the `padv.Artifact` object.

Test Query in Command Window

Although you typically use queries inside your process model, you can run `FindFileWithAddress` queries outside of your process model to confirm which artifacts a query returns.

Open a project. For this example, you can open the Process Advisor example project.

```
processAdvisorExampleStart
```

Create an instance of the query. For example, create a query that finds a file with the artifact type Model Advisor configuration file (`ma_config_file`) at the file path specified by `fullfile("tools", "sampleChecks.json")`.

```
q = padv.builtin.query.FindFileWithAddress( ...
    Type = "ma_config_file", ...
    Path = fullfile("tools", "sampleChecks.json"))
```

Run the query.

```
run(q)
```

The query returns the specified artifact.

```
ans =
    "tools\sampleChecks.json"
```

Capabilities and Limitations

This table identifies functionality that is supported by the query.

Functionality	Supported?
Input query for task	Yes.
Iteration query for task	Only when the query property <code>TrackArtifacts</code> is true.

See Also

`padv.Artifact` | `padv.builtin.task.RunModelStandards` |
`padv.builtin.query.FindFilesWithLabel`

Topics

“Reconfigure Task Behavior”

padv.builtin.query.FindMAJustificationFileForModel Class

Namespace: padv.builtin.query padv.builtin.query

Superclasses: padv.Query

Query for finding Model Advisor justification file for model

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.query.FindMAJustificationFileForModel` class provides a query that can return the Model Advisor justification file associated with a model. You can automatically include or exclude certain artifacts by using the optional name-value arguments.

You can use this query in your process model to find artifacts for your tasks to iterate over or use as inputs.

The `padv.builtin.query.FindMAJustificationFileForModel` class is a handle class.

Creation

Description

`query = padv.builtin.query.FindMAJustificationFileForModel(JustificationFolder = relativePathToFolder)` creates a query for finding the Model Advisor justification file for a model. The query searches for the justification file within the specified folder `relativePathToFolder`. The query expects that the current iteration artifact is a model and that the Model Advisor justification file name is the model name followed by `_justifications.json`. The query returns the justification file as a `padv.Artifact` object of type `ma_justification_file`.

`query = padv.builtin.query.FindMAJustificationFileForModel(____, Name=Value)` sets certain properties using one or more name-value arguments. For example, `padv.builtin.query.FindMAJustificationFileForModel(Name = "FindMyJustification")` creates a query object with the name "FindMyJustification".

The `padv.builtin.query.FindMAJustificationFileForModel` class also has other properties, but you cannot set those properties during query creation.

Input Arguments

relativePathToFolder — Relative path to folder that contains Model Advisor justification files

string | character vector

Relative path to folder that contains Model Advisor justification files (`.json`) for the models in the project, specified as a string or a character vector.

Example: `fullfile("Justifications","ModelAdvisor")`

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, . . . ,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `query = padv.builtin.query.FindMAJustificationFileForModel(Name = "FindMyJustifications")`

Name — Unique identifier for query

`string`

Unique identifier for query, specified as a string.

Example: `"FindMyJustification"`

Data Types: `string`

Parent — Initial query run before iteration query

`"padv.builtin.query.FindModels"` (default) | `padv.Query` object | Name of `padv.Query` object

Initial query run before iteration query, specified as either a `padv.Query` object or the Name of a `padv.Query` object. When you specify a `padv.Query` object as the iteration query for a task, the Parent query is the initial query that the build system runs before running the specified iteration query.

The build system ignores the Parent query when you specify a query as an input query or dependency query for a task.

For information on how to improve Process Advisor load times by sharing query instances across your process model, see “Best Practices for Process Model Authoring”.

Example: `padv.builtin.query.FindModels`

Example: `padv.builtin.query.FindModels(IncludePath = "Control")`

Properties

JustificationFolder — Relative path to folder that contains Model Advisor justification files

`string` | `character vector`

Relative path to folder that contains Model Advisor justification files (`.json`) for the models in the project, specified as a string or a character vector.

Example: `fullfile("Justifications","ModelAdvisor")`

Data Types: `char` | `string`

Title — Query title

`"Model Advisor justification file for a model"` (default) | `string` | `character vector`

Query title, specified as a string or a character vector.

Example: `"Find my Model Advisor justification file"`

Data Types: string

DefaultArtifactType — Default artifact type returned by query

"padv_output_file" (default) | ...

Default artifact type returned by the query, specified as one or more of the values in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file

Category	Artifact Type	Description
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Parent — Initial query run before iteration query

"padv.builtin.query.FindModels" (default) | padv.Query object | Name of padv.Query object

Initial query run before iteration query, specified as either a padv.Query object or the Name of a padv.Query object. When you specify a padv.Query object as the iteration query for a task, the Parent query is the initial query that the build system runs before running the specified iteration query.

The build system ignores the Parent query when you specify a query as an input query or dependency query for a task.

For information on how to improve Process Advisor load times by sharing query instances across your process model, see "Best Practices for Process Model Authoring".

Example: padv.builtin.query.FindModels

Example: padv.builtin.query.FindModels(IncludePath = "Control")

Name — Unique identifier for query

string

Unique identifier for query, specified as a string.

Example: "FindMyJustification"

Data Types: `string`

ShowFileExtension — Show file extensions for returned artifacts

`0 (false) | 1 (true)`

Show file extensions in the `Alias` property of returned artifacts, specified as a numeric or logical `1 (true)` or `0 (false)`. The `Alias` property controls the display name for the artifact in the **Tasks** column in Process Advisor.

By default, queries strip file extensions from the `Alias` property of each task iteration artifact. To show file extensions for all artifacts in the **Tasks** column, select the project setting **Show file extensions**. To keep file extensions in the results for a specific query, specify the query property `ShowFileExtension` as `true`.

Example: `true`

Data Types: `logical`

SortArtifacts — Setting for automatically sorting artifacts by address

`true` or `1 (default) | false` or `0`

Setting for automatically sorting artifacts by address, specified as a numeric or logical `1 (true)` or `0 (false)`. When a query returns artifacts, the artifacts should be in a consistent order. By default, the build system sorts artifacts by the artifact address.

Alternatively, you can sort artifacts in a different order by overriding the internal `sortArtifacts` method in a subclass that defines a custom sort behavior. For an example, see “Sort Artifacts in Specific Order”.

The build system automatically calls the `sortArtifacts` method when using the process model. The `sortArtifacts` method expects two input arguments: a `padv.Query` object and a list of `padv.Artifact` objects returned by the `run` method. The `sortArtifacts` method should return a list of sorted `padv.Artifact` objects.

Example: `SortArtifacts = false`

Data Types: `logical`

FunctionHandle — Handle to function that function-based query runs

`function_handle`

Handle to the function that a function-based query runs, specified as a `function_handle`.

If you define your query functionality inside a function and you or the build system call `run` on the query, the query runs the function specified by the `function_handle`.

The built-in queries are defined inside classes and do not use the `FunctionHandle`.

Example: `FunctionHandle = @FunctionForQuery`

Data Types: `function_handle`

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Run query to find the artifacts that meet the criteria specified by the query.</p> <p>The query returns a <code>padv.Artifact</code> object or an array of <code>padv.Artifact</code> objects that represent those artifacts.</p> <hr/> <p>Note You do not need to manually invoke this method inside your process model. The build system automatically invokes the <code>run</code> method to find artifacts.</p> <hr/> <p>The <code>run</code> method inside this built-in query runs on a query object <code>obj</code> and returns artifacts <code>artifacts</code> that are associated with the artifact <code>iterationArtifact</code>. If you inherit from this built-in query, make sure to use the same method signature inside your custom query:</p> <pre>function artifacts = run(obj, iterationArtifact) ... end</pre>
-----	--

Examples

Use Justifications When Checking Modeling Standards

You can use the `FindMAJustificationFileForModel` query in your process model to find Model Advisor justification files for your tasks to iterate over (`IterationQuery`) or use as inputs (`addInputQueries`). For example, you can use the `FindMAJustificationFileForModel` query to find the Model Advisor justification files associated with each model and provide those files as the input to the built-in task `padv.builtin.task.RunModelStandards`.

Open a project. For this example, you can open the Process Advisor example project.

```
processAdvisorExampleStart
```

If you want the built-in task `padv.builtin.task.RunModelStandards` to use your Model Advisor justification files when checking modeling standards, you can reconfigure the task to add the justification files as inputs. Add the built-in query `padv.builtin.query.FindMAJustificationFileForModel` as an input query for the task and specify the folder, `JustificationFolder`, that contains the justification files. For example, if your justification files are in the directory `Justifications/ModelAdvisor` relative to your project root, use the function `addInputQueries` to add those justification files as inputs to the task:

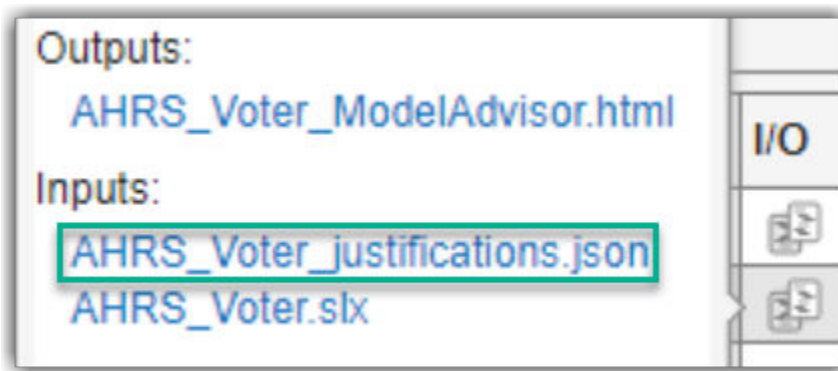
```

%% Check modeling standards
% Tools required: Model Advisor
if includeModelStandardsTask
    maTask = pm.addTask(padv.builtin.task.RunModelStandards());

    % Find and use justification files
    maTask.addInputQueries(...
        padv.builtin.query.FindMAJustificationFileForModel(...
            JustificationFolder=fullfile("Justifications","ModelAdvisor"));
end

```

In Process Advisor, view the updated **Tasks** by clicking **Refresh Tasks** and switching to the **Project** view. When you run the **Check Modeling Standards** task, the justification file appears as an input in the **I/O** column in Process Advisor.



Capabilities and Limitations

This table identifies functionality that is supported by the query.

Functionality	Supported?
Input query for task	Yes.
Iteration query for task	Yes.

Version History

Introduced in R2023a

See Also

`padv.builtin.task.RunModelStandards`

Topics

“Justify Model Advisor Violations from Check Analysis”

padv.builtin.query.FindModels Class

Namespace: padv.builtin.query padv.builtin.query

Superclasses: padv.Query, padv.builtin.query.FindArtifacts

Query for finding models

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.query.FindModels` class provides a query that can return models in your project folder. By default, the query finds the Simulink and System Composer models in your project folder. You can automatically include or exclude certain files by using the optional name-value arguments.

You can use this query in your process model to find models for your tasks to iterate over or use as inputs.

The `padv.builtin.query.FindModels` class is a handle class.

Creation

Description

`query = padv.builtin.query.FindModels()` creates a query for finding each of the models in your project folder.

`query = padv.builtin.query.FindModels(Name=Value)` sets certain properties using one or more name-value arguments. For example, `padv.builtin.query.FindModels(IncludePathRegex = "modelPrefix.*\.slx")` creates a query that can find SLX files where the path contains `modelPrefix`.

The `padv.builtin.query.FindModels` class also has other properties, but you cannot set those properties during query creation.

Input Arguments

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `query = padv.builtin.query.FindModels(IncludePathRegex = "modelPrefix.*\.slx")`

Name — Unique identifier for query

string | character vector

Unique identifier for query, specified as a string or a character vector.

Example: "FindMyModels"

Data Types: char | string

IncludeLabel — Find models with specific project label

cell array

Find models with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Classification", "Design"}

Data Types: cell

ExcludeLabel — Exclude models with specific project label

cell array

Exclude models with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Classification", "Design"}

Data Types: cell

IncludePath — Find models where path contains specific text

string | character vector

Find models where the path contains specific text, specified as a string. Consider using `IncludePathRegex` instead.

Example: "Control"

Data Types: char | string

ExcludePath — Exclude models where path contains specific text

string | character vector

Exclude models where the path contains specific text, specified as a string or a character vector. Consider using `ExcludePathRegex` instead.

Example: "Control"

Data Types: string

IncludePathRegex — Find models where path matches regular expression pattern

string | character vector

Find models where the path matches a regular expression pattern, specified as a character vector or string. `IncludePathRegex` expects UNIX-style path separators.

If you want to use a literal path, use `IncludePath` instead. You can specify either `IncludePath` or `IncludePathRegex` but not both.

Example: "modelPrefix.*\.slx"

Data Types: char | string

ExcludePathRegex — Exclude models where path matches regular expression pattern

string | character vector

Exclude models where the path matches a regular expression pattern, specified as a string or a character vector. `ExcludePathRegex` expects UNIX-style path separators.

If you want to use a literal path, use `ExcludePath` instead. You can specify either `ExcludePath` or `ExcludePathRegex` but not both.

Example: `"modelPrefix.*\.slx"`

Data Types: `char` | `string`

InProject — Include only models added to project

0 (false) (default) | 1 (true)

Include only models that have been added to the project, specified as a numeric or logical 1 (true) or 0 (false).

Note If you specify `InProject` as true, you cannot use the query as an input query.

Example: `true`

Data Types: `logical`

Properties

ArtifactType — Type of artifact

`["sl_model_file" "zc_file"]` (default) | `"sl_model_file"` | `"zc_file"` | ...

Type of artifact, specified as one or more of the values in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	<code>"m_class"</code>	MATLAB class
	<code>"m_file"</code>	MATLAB file
	<code>"m_func"</code>	MATLAB function
	<code>"m_method"</code>	MATLAB class method
	<code>"m_property"</code>	MATLAB class property
Model Advisor	<code>"ma_config_file"</code>	Model Advisor configuration file
	<code>"ma_justification_file"</code>	Model Advisor justification file
Process Advisor	<code>"padv_dep_artifacts"</code>	Related artifacts that current artifact depends on
	<code>"padv_output_file"</code>	Process Advisor output file
Project	<code>"project"</code>	Current project file
Requirements	<code>"mwreq_item"</code>	Requirement (since R2024b)
	<code>"sl_req"</code>	Requirement (for R2024a and earlier)
	<code>"sl_req_file"</code>	Requirement file
	<code>"sl_req_table"</code>	Requirements Table

Category	Artifact Type	Description
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
"sl_subsystem"	Subsystem	
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
"sl_test_seq"	Test Sequence	

Category	Artifact Type	Description
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "sl_model_file"

Example: ["sl_model_file" "zc_file"]

IncludeLabel — Find models with specific project label

cell array

Find models with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Classification", "Design"}

Data Types: cell

ExcludeLabel — Exclude models with specific project label

cell array

Exclude models with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Classification", "Design"}

Data Types: cell

IncludePathRegex — Find models where path matches regular expression pattern

string | character vector

Find models where the path matches a regular expression pattern, specified as a string or character vector. IncludePathRegex expects UNIX-style path separators.

If you want to use a literal path, use IncludePath instead. You can specify either IncludePath or IncludePathRegex but not both.

Example: "modelPrefix.*\.slx"

Data Types: char | string

ExcludePathRegex — Exclude models where path matches regular expression pattern

string | character vector

Exclude models where the path matches a regular expression pattern, specified as a string or a character vector. ExcludePathRegex expects UNIX-style path separators.

If you want to use a literal path, use ExcludePath instead. You can specify either ExcludePath or ExcludePathRegex but not both.

Example: "modelPrefix.*\.slx"

Data Types: char | string

FilterSubFileArtifacts — Filter out sub-file artifacts from query results

1 (true) (default) | 0 (false)

Filter out sub-file artifacts from query results, specified as a numeric or logical 1 (true) or 0 (false).

A sub-file is a part of a larger file. For example, a subsystem is a sub-file of a model file. If you want a query to return both models and subsystems, you can specify `FilterSubFileArtifacts` as false and `ArtifactType` as ["sl_model_file" "sl_subsystem"].

Example: false

Data Types: logical

InProject — Include only models added to project

0 (false) (default) | 1 (true)

Include only models that have been added to the project, specified as a numeric or logical 1 (true) or 0 (false).

For more information about how to add or remove files from a project, see “Manage Project Files”.

Note If you specify `InProject` as true, you cannot use the query as an input query.

Example: true

Attributes:

Dependent

Data Types: logical

Title — Query title

"All models in the project" (default) | string | character vector

Query title, specified as a string or a character vector.

Example: "Find my models"

Data Types: string

DefaultArtifactType — Default artifact type returned by query

"sl_model_file" (default) | "zc_file" | ...

Default artifact type returned by the query, specified as one or more of the values in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file

Category	Artifact Type	Description
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result

Category	Artifact Type	Description
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "sl_model_file"

Example: ["sl_model_file" "zc_file"]

Parent — Initial query that runs before iteration query

padv.Query | Name of padv.Query object

Initial query that runs before iteration query, specified as either a padv.Query object or the Name of a padv.Query object. When you specify an iteration query for a task, the parent query is the initial query that the build system runs before running the specified iteration query.

For information on how to improve Process Advisor load times by sharing query instances across your process model, see “Best Practices for Process Model Authoring”.

Example: sharedQuery

Example: "FindMyInitialArtifacts"

Name — Unique identifier for query

string | character vector

Unique identifier for query, specified as a string or a character vector.

Example: "FindMyModels"

Data Types: char | string

ShowFileExtension — Show file extensions for returned models

0 (false) | 1 (true)

Show file extensions in the Alias property of returned models, specified as a numeric or logical 1 (true) or 0 (false). The Alias property controls the display name for the artifact in the **Tasks** column in Process Advisor.

By default, queries strip file extensions from the Alias property of each task iteration artifact. To show file extensions for all artifacts in the **Tasks** column, select the project setting **Show file extensions**. To keep file extensions in the results for a specific query, specify the query property ShowFileExtension as true.

Example: true

Data Types: logical

SortArtifacts — Setting for automatically sorting artifacts by address`true or 1 (default) | false or 0`

Setting for automatically sorting artifacts by address, specified as a numeric or logical 1 (`true`) or 0 (`false`). When a query returns artifacts, the artifacts should be in a consistent order. By default, the build system sorts artifacts by the artifact address.

Alternatively, you can sort artifacts in a different order by overriding the internal `sortArtifacts` method in a subclass that defines a custom sort behavior. For an example, see “Sort Artifacts in Specific Order”.

The build system automatically calls the `sortArtifacts` method when using the process model. The `sortArtifacts` method expects two input arguments: a `padv.Query` object and a list of `padv.Artifact` objects returned by the `run` method. The `sortArtifacts` method should return a list of sorted `padv.Artifact` objects.

Example: `SortArtifacts = false`

Data Types: `logical`

FunctionHandle — Handle to function that function-based query runs`function_handle`

Handle to the function that a function-based query runs, specified as a `function_handle`.

If you define your query functionality inside a function and you or the build system call `run` on the query, the query runs the function specified by the `function_handle`.

The built-in queries are defined inside classes and do not use the `FunctionHandle`.

Example: `FunctionHandle = @FunctionForQuery`

Data Types: `function_handle`

Methods**Specialized Public Methods**

This class overrides the following inherited methods.

run	<p>Run query to find the artifacts that meet the criteria specified by the query.</p> <p>The query returns a <code>padv.Artifact</code> object or an array of <code>padv.Artifact</code> objects that represent those artifacts.</p> <hr/> <p>Note You do not need to manually invoke this method inside your process model. The build system automatically invokes the <code>run</code> method to find artifacts.</p> <hr/> <p>The <code>run</code> method inside this built-in query runs on a query object <code>obj</code> and returns artifacts <code>artifacts</code>. If you inherit from this built-in query, make sure to use the same method signature inside your custom query:</p> <pre>function artifacts = run(obj,~) ... end</pre>
-----	--

Examples

Find Models for Task in Process

You can use the `FindModels` query in your task definition to find models that your tasks can iterate over (`IterationQuery`) or use as inputs (`InputQueries`). For example, suppose that you only want to run the **Check Modeling Standards** task for SLX model files where the model name contains `Control`.

Open a project. For this example, open the Process Advisor example project.

```
processAdvisorExampleStart
```

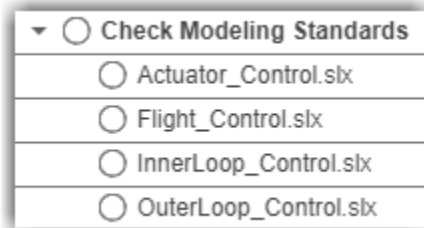
By default, the **Check Modeling Standards** task iterates over each model in the project because the task uses the built-in query `padv.builtin.query.FindModels` as the `IterationQuery`.

In the process model, reconfigure which models the **Check Modeling Standards** task iterates over by updating the `IterationQuery` for the task. The associated built-in task is `padv.builtin.task.RunModelStandards`. You can update the code to find the models in the project by using the built-in query `padv.builtin.query.FindModels`, but use the `IncludePathRegex` argument to have the query only return SLX files where the model name contains `Control`.

```
if includeModelStandardsTask
    maTask = pm.addTask(padv.builtin.task.RunModelStandards());
    ...
    % Specify which set of artifacts to run for
    maTask.IterationQuery = ...
        padv.builtin.query.FindModels(IncludePathRegex = "Control.*\.slx");
```

end

For this example, when you save the process model and refresh Process Advisor, the model `AHRS_Voter.slx` does not appear under the task title in Process Advisor because `AHRS_Voter.slx` does not include `Control` in the path.



Test FindModels Query Outside Process Model

Although you typically use a query inside your process model, you can run an instance of the `FindModels` query outside of your process model to confirm which artifacts the query returns.

Open a project. For this example, you can open the Process Advisor example project.

```
processAdvisorExampleStart
```

Create an instance of the query. You can use the arguments of the query to filter the query results. For example, you can use the `IncludeLabel` argument to have the query only return artifacts that use the `Design` project label from the `Classification` project label category.

```
q = padv.builtin.query.FindModels(...
IncludeLabel = {"Classification", "Design"});
```

Run the query and inspect the array of artifacts that the query returns.

```
run(q)
```

```
ans =
```

```
1x5 Artifact array with properties:
```

```
Type
Parent
ArtifactAddress
```

Capabilities and Limitations

This table identifies functionality that is supported by the query.

Functionality	Supported?
Input query for task	Only when the query property <code>InProject</code> is <code>false</code> .
Iteration query for task	Yes.

See Also

`padv.builtin.query.FindArtifacts` | `padv.builtin.query.FindModelsWithLabel` |
`padv.builtin.query.FindModelsWithTestCases`

padv.builtin.query.FindModelsWithLabel Class

Namespace: padv.builtin.query padv.builtin.query

Superclasses: padv.Query, padv.builtin.query.FindArtifacts

Query for finding models with project label

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.query.FindModelsWithLabel` class provides a query that can return each of the models that use the specified project label. You can automatically include or exclude certain files by using the optional name-value arguments. If you do not need to specify a project label, you can use the built-in query `padv.builtin.query.FindModels` instead.

You can use this query in your process model to find models for your tasks to iterate over.

The `padv.builtin.query.FindModelsWithLabel` class is a handle class.

Creation

Description

`query = padv.builtin.query.FindModelsWithLabel(categoryName, labelName)` creates a query for finding models that use the project label `labelName` from the project label category `categoryName`. For more information about project labels, see "Add Labels to Project Files".

`query = padv.builtin.query.FindModelsWithLabel(____, Name=Value)` sets certain properties using one or more name-value arguments. For example, `padv.builtin.query.FindModels(IncludePathRegex = "modelPrefix.*\.slx")` creates a query that can find SLX files where the path contains `modelPrefix`.

The `padv.builtin.query.FindModelsWithLabel` class also has other properties, but you cannot set those properties during query creation.

Input Arguments

categoryName — Name of category for project label

string | character vector

Name of the category for the project label, specified as a string or a character vector.

The query uses `categoryName` to specify the first entry for the query property `IncludeLabel`. For more information about project labels, see "Add Labels to Project Files".

Example: "Classification"

Data Types: char | string

labelName — Name of project label

string | character vector

Name of project label, specified as a string or a character vector.

The query uses `labelName` to specify the second entry for the query property `IncludeLabel`. For more information about project labels, see “Add Labels to Project Files”.

Example: "Design"

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, . . . , NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `query = padv.builtin.query.FindModelsWithLabel("Classification","Design",IncludePathRegex = "modelPrefix.*\.slx")`

ExcludeLabel — Exclude models with specific project label

cell array

Exclude models with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

For more information about project labels, see “Add Labels to Project Files”.

Example: {"Classification","Design"}

Data Types: `cell`

ExcludePath — Exclude models where path contains specific text

string | character vector

Exclude models where the path contains specific text, specified as a string or a character vector. Consider using `ExcludePathRegex` instead.

Example: "Control"

Data Types: `string`

ExcludePathRegex — Exclude models where path matches regular expression pattern

string | character vector

Exclude models where the path matches a regular expression pattern, specified as a string or a character vector. `ExcludePathRegex` expects UNIX-style path separators.

If you want to use a literal path, use `ExcludePath` instead. You can specify either `ExcludePath` or `ExcludePathRegex` but not both.

Example: "modelPrefix.*\.slx"

Data Types: `char` | `string`

IncludeLabel — Find models with specific project label

cell array

Find models with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Alternatively, you can specify the first and second entries using the arguments `categoryName` and `labelName`. The query uses those arguments to specify the entries for `IncludeLabel`. For more information about project labels, see “Add Labels to Project Files”.

Example: {"Classification", "Design"}

Data Types: cell

IncludePath — Find models where path contains specific text

string | character vector

Find models where the path contains specific text, specified as a string. Consider using `IncludePathRegex` instead.

Example: "Control"

Data Types: char | string

IncludePathRegex — Find models where path matches regular expression pattern

string | character vector

Find models where the path matches a regular expression pattern, specified as a character vector or string. `IncludePathRegex` expects UNIX-style path separators.

If you want to use a literal path, use `IncludePath` instead. You can specify either `IncludePath` or `IncludePathRegex` but not both.

Example: "modelPrefix.*\slx"

Data Types: char | string

Name — Unique identifier for query

string | character vector

Unique identifier for query, specified as a string or a character vector.

Example: "FindMyLabeledModels"

Data Types: char | string

Properties

ArtifactType — Type of artifact

"sl_model_file" | "zc_file" | ...

Type of artifact, specified as one or more of the values in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property

Category	Artifact Type	Description
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file

Category	Artifact Type	Description
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "sl_model_file"

Example: ["sl_model_file" "zc_file"]

IncludeLabel — Find models with specific project label

cell array

Find models with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Alternatively, you can specify the first and second entries using the arguments `categoryName` and `labelName`. The query uses those arguments to specify the entries for `IncludeLabel`. For more information about project labels, see "Add Labels to Project Files".

Example: {"Classification", "Design"}

Data Types: cell

ExcludeLabel — Exclude models with specific project label

cell array

Exclude models with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

For more information about project labels, see "Add Labels to Project Files".

Example: {"Classification", "Design"}

Data Types: cell

IncludePathRegex — Find models where path matches regular expression pattern

string | character vector

Find models where the path matches a regular expression pattern, specified as a string or character vector. `IncludePathRegex` expects UNIX-style path separators.

If you want to use a literal path, use `IncludePath` instead. You can specify either `IncludePath` or `IncludePathRegex` but not both.

Example: "modelPrefix.*\.slx"

Data Types: char | string

ExcludePathRegex — Exclude models where path matches regular expression pattern

string | character vector

Exclude models where the path matches a regular expression pattern, specified as a string or a character vector. `ExcludePathRegex` expects UNIX-style path separators.

If you want to use a literal path, use `ExcludePath` instead. You can specify either `ExcludePath` or `ExcludePathRegex` but not both.

Example: "modelPrefix.*\.slx"

Data Types: char | string

FilterSubFileArtifacts — Filter out sub-file artifacts from query results

1 (true) (default) | 0 (false)

Filter out sub-file artifacts from query results, specified as a numeric or logical 1 (true) or 0 (false).

A sub-file is a part of a larger file. For example, a subsystem is a sub-file of a model file.

Example: false

Data Types: logical

InProject — Include only models added to project

0 (false) (default) | 1 (true)

Include only models that have been added to the project, specified as a numeric or logical 1 (true) or 0 (false).

For more information about how to add or remove files from a project, see “Manage Project Files”.

Example: true

Attributes:

Dependent

Data Types: logical

Title — Query title

"All models with label ''" (default) | string | character vector

Query title, specified as a string or a character vector.

When you specify a project label name, the query automatically updates the `Title` to include that project label name. For example, "All models with label 'Design'".

Example: "Find my labeled models"

Data Types: string

DefaultArtifactType — Default artifact type returned by query

"padv_output_file" (default) | "sl_model_file" | "zc_file" | ...

Default artifact type returned by the query, specified as one or more of the values in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture

Category	Artifact Type	Description
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "sl_model_file"

Example: ["sl_model_file" "zc_file"]

Parent – Initial query that runs before iteration query

`padv.Query` | Name of `padv.Query` object

Initial query that runs before iteration query, specified as either a `padv.Query` object or the Name of a `padv.Query` object. When you specify an iteration query for a task, the parent query is the initial query that the build system runs before running the specified iteration query.

For information on how to improve Process Advisor load times by sharing query instances across your process model, see “Best Practices for Process Model Authoring”.

Example: `sharedQuery`

Example: "FindMyInitialArtifacts"

ShowFileExtension – Show file extensions for returned models

0 (false) | 1 (true)

Show file extensions in the Alias property of returned models, specified as a numeric or logical 1 (true) or 0 (false). The Alias property controls the display name for the artifact in the **Tasks** column in Process Advisor.

By default, queries strip file extensions from the Alias property of each task iteration artifact. To show file extensions for all artifacts in the **Tasks** column, select the project setting **Show file**

extensions. To keep file extensions in the results for a specific query, specify the query property `ShowFileExtension` as `true`.

Example: `true`

Data Types: `logical`

SortArtifacts — Setting for automatically sorting artifacts by address

`true` or `1` (default) | `false` or `0`

Setting for automatically sorting artifacts by address, specified as a numeric or logical `1` (`true`) or `0` (`false`). When a query returns artifacts, the artifacts should be in a consistent order. By default, the build system sorts artifacts by the artifact address.

Alternatively, you can sort artifacts in a different order by overriding the internal `sortArtifacts` method in a subclass that defines a custom sort behavior. For an example, see “Sort Artifacts in Specific Order”.

The build system automatically calls the `sortArtifacts` method when using the process model. The `sortArtifacts` method expects two input arguments: a `padv.Query` object and a list of `padv.Artifact` objects returned by the `run` method. The `sortArtifacts` method should return a list of sorted `padv.Artifact` objects.

Example: `SortArtifacts = false`

Data Types: `logical`

FunctionHandle — Handle to function that function-based query runs

`function_handle`

Handle to the function that a function-based query runs, specified as a `function_handle`.

If you define your query functionality inside a function and you or the build system call `run` on the query, the query runs the function specified by the `function_handle`.

The built-in queries are defined inside classes and do not use the `FunctionHandle`.

Example: `FunctionHandle = @FunctionForQuery`

Data Types: `function_handle`

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Run query to find the artifacts that meet the criteria specified by the query.</p> <p>The query returns a <code>padv.Artifact</code> object or an array of <code>padv.Artifact</code> objects that represent those artifacts.</p> <hr/> <p>Note You do not need to manually invoke this method inside your process model. The build system automatically invokes the <code>run</code> method to find artifacts.</p> <hr/> <p>The <code>run</code> method inside this built-in query runs on a query object <code>obj</code> and returns artifacts <code>artifacts</code>. If you inherit from this built-in query, make sure to use the same method signature inside your custom query:</p> <pre>function artifacts = run(obj,~) ... end</pre>
-----	--

Examples

Find Models with Specific Project Label

You can use the `FindModelsWithLabel` query in your task definition to find models that your tasks can iterate over (`IterationQuery`). For example, suppose that you want the built-in task `padv.builtin.task.RunModelStandards` to only run for models that use the project label `RunModelAdvisor` from the project label category `ModelLabels`.

Open a project. For this example, open the Process Advisor example project.

```
processAdvisorExampleStart
```

By default, the **Check Modeling Standards** task iterates over each model in the project because the task uses the built-in query `padv.builtin.query.FindModels` as the `IterationQuery`.

In the process model, reconfigure which models the **Check Modeling Standards** task iterates over by updating the `IterationQuery` for the task. The associated built-in task is `padv.builtin.task.RunModelStandards`. You can update the code to find only models in the project that use the project label `RunModelAdvisor` from the project label category `ModelLabels`. You can use the built-in query `padv.builtin.query.FindModelsWithLabel` to find the models that use that project label. Specify the first input argument as the project label category and the second argument as the project label name.

```
maTask = pm.addTask(padv.builtin.task.RunModelStandards());  
maTask.IterationQuery = ...  
    padv.builtin.query.FindModelsWithLabel("ModelLabels", "RunModelAdvisor");
```

Capabilities and Limitations

This table identifies functionality that is supported by the query.

Functionality	Supported?
Input query for task	No.
Iteration query for task	Yes.

See Also

[padv.builtin.query.FindArtifacts](#) | [padv.builtin.query.FindModels](#) |
[padv.builtin.query.FindModelsWithTestCases](#)

Topics

“Add Labels to Project Files”

padv.builtin.query.FindModelsWithTestCases Class

Namespace: padv.builtin.query padv.builtin.query

Superclasses: padv.Query

Query for finding models that have test cases

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.query.FindModelsWithTestCases` class provides a query that can return the models in your project that have test cases.

You can use this query in your process model to find models that are associated with test cases. For example, if you use a built-in task like `padv.builtin.task.MergeTestResults` to merge the test results for each model in your project, you can use the `FindModelsWithTestCases` query to find only models that have test cases and therefore test results that the task can merge.

The `padv.builtin.query.FindModelsWithTestCases` class is a handle class.

Creation

Description

`query = padv.builtin.query.FindModelsWithTestCases()` creates a query for finding the models in your project that have test cases.

`query = padv.builtin.query.FindModelsWithTestCases(Name=Value)` sets certain properties using one or more name-value arguments. For example, `query = padv.builtin.query.FindModelsWithTestCases(ExcludePath = "Control")` creates a query that finds models that have test cases, but excludes models that have "Control" in the file address.

The `padv.builtin.query.FindModelsWithTestCases` class also has other properties, but you cannot set those properties during query creation.

Input Arguments

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `query = padv.builtin.query.FindModelsWithTestCases(ExcludePath = "Control")`

ExcludeLabel — Exclude models with specific project label

cell array

Exclude models with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Classification", "Design"}

Data Types: cell

ExcludePath — Exclude models where path contains specific text

string | character vector

Exclude models where the path contains specific text, specified as a string or a character vector.

Example: "Control"

Data Types: char | string

IncludeLabel — Find models with specific project label

cell array

Find models with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Classification", "Design"}

Data Types: cell

IncludePath — Find models where path contains specific text

string | character vector

Find models where the path contains specific text, specified as a string or a character vector.

Example: "Control"

Data Types: char | string

Name — Unique identifier for query

string | character vector

Unique identifier for query, specified as a string or a character vector.

Example: "FindModelsWithTests"

Data Types: char | string

Parent — Initial query that runs before iteration query

"padv.builtin.query.FindModels" (default) | padv.Query | Name of padv.Query object

Initial query that runs before iteration query, specified as either a padv.Query object or the Name of a padv.Query object. When you specify an iteration query for a task, the parent query is the initial query that the build system runs before running the specified iteration query.

For information on how to improve Process Advisor load times by sharing query instances across your process model, see "Best Practices for Process Model Authoring".

Example: sharedFindModelsQuery

Example: "padv.builtin.query.FindModels"

Tags — Find models with at least one test case that uses specific test case tags

string | array of strings

Find models that have at least one test case that uses specific test case tags, specified as a string or an array of strings.

Example: "FeatureA"

Example: ["FeatureA", "FeatureB"]

Data Types: string

Properties

IncludeLabel — Find models with specific project label

cell array

Find models with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Classification", "Design"}

Data Types: cell

ExcludeLabel — Exclude models with specific project label

cell array

Exclude models with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Classification", "Design"}

Data Types: cell

IncludePath — Find models where path contains specific text

string | character vector

Find models where the path contains specific text, specified as a string or a character vector.

Example: "Control"

Data Types: char | string

ExcludePath — Exclude models where path contains specific text

string | character vector

Exclude models where the path contains specific text, specified as a string or a character vector.

Example: "Control"

Data Types: char | string

Tags — Only include models with test cases that use specific test case tags

string | array of strings

Only include models with test cases that use specific test case tags, specified as a string or an array of strings.

Example: "FeatureA"

Example: ["FeatureA", "FeatureB"]

Data Types: string

Title – Query title

"All models with associated test cases" (default) | string | character vector

Query title, specified as a string or a character vector.

Example: "Find models that have test cases"

Data Types: char | string

DefaultArtifactType – Default artifact type returned by query

["sl_model_file" "zc_file"] (default) | "sl_model_file" | "zc_file" | ...

Default artifact type returned by the query, specified as one or more of the values in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram

Category	Artifact Type	Description
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "zc_file"

Parent – Initial query that runs before iteration query

"padv.builtin.query.FindModels" (default) | padv.Query | Name of padv.Query object

Initial query that runs before iteration query, specified as either a `padv.Query` object or the Name of a `padv.Query` object. When you specify an iteration query for a task, the parent query is the initial query that the build system runs before running the specified iteration query.

For information on how to improve Process Advisor load times by sharing query instances across your process model, see "Best Practices for Process Model Authoring".

Example: `sharedFindModelsQuery`

Example: "padv.builtin.query.FindModels"

Name — Unique identifier for query

string | character vector

Unique identifier for query, specified as a string or a character vector.

Example: "FindModelsWithTests"

Data Types: char | string

ShowFileExtension — Show file extensions for returned artifacts

0 (false) | 1 (true)

Show file extensions in the `Alias` property of returned artifacts, specified as a numeric or logical 1 (true) or 0 (false). The `Alias` property controls the display name for the artifact in the **Tasks** column in Process Advisor.

By default, queries strip file extensions from the `Alias` property of each task iteration artifact. To show file extensions for all artifacts in the **Tasks** column, select the project setting **Show file extensions**. To keep file extensions in the results for a specific query, specify the query property `ShowFileExtension` as true.

Example: true

Data Types: logical

SortArtifacts — Setting for automatically sorting artifacts by address

true or 1 (default) | false or 0

Setting for automatically sorting artifacts by address, specified as a numeric or logical 1 (true) or 0 (false). When a query returns artifacts, the artifacts should be in a consistent order. By default, the build system sorts artifacts by the artifact address.

Alternatively, you can sort artifacts in a different order by overriding the internal `sortArtifacts` method in a subclass that defines a custom sort behavior. For an example, see "Sort Artifacts in Specific Order".

The build system automatically calls the `sortArtifacts` method when using the process model. The `sortArtifacts` method expects two input arguments: a `padv.Query` object and a list of `padv.Artifact` objects returned by the `run` method. The `sortArtifacts` method should return a list of sorted `padv.Artifact` objects.

Example: `SortArtifacts = false`

Data Types: logical

FunctionHandle — Handle to function that function-based query runs

function_handle

Handle to the function that a function-based query runs, specified as a `function_handle`.

If you define your query functionality inside a function and you or the build system call `run` on the query, the query runs the function specified by the `function_handle`.

The built-in queries are defined inside classes and do not use the `FunctionHandle`.

Example: `FunctionHandle = @FunctionForQuery`

Data Types: function_handle

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Run query to find the artifacts that meet the criteria specified by the query.</p> <p>The query returns a <code>padv.Artifact</code> object or an array of <code>padv.Artifact</code> objects that represent those artifacts.</p> <hr/> <p>Note You do not need to manually invoke this method inside your process model. The build system automatically invokes the <code>run</code> method to find artifacts.</p> <hr/> <p>The <code>run</code> method inside this built-in query runs on a query object <code>obj</code> and returns artifacts that are associated with the artifact <code>iterationArtifact</code>. If you inherit from this built-in query, make sure to use the same method signature inside your custom query:</p> <pre>function artifacts = run(obj,iterationArtifact) ... end</pre>
-----	--

Examples

Find Models with Test Cases for Task in Process

You can use the `FindModelsWithTestCases` query in your process model to find units and components that your tasks can iterate over (`IterationQuery`). Suppose you have a custom task that you want to run for each model in your project that has test cases. You can find those models by using the built-in query `FindModelsWithTestCases`.

Open a project. For this example, open the Process Advisor example project.

```
processAdvisorExampleStart
```

To have the custom task run for each of those models, specify the query as the iteration query for the task. For example, in your process model:

```
pm.addTask("MyCustomTask",...
    IterationQuery = padv.builtin.query.FindModelsWithTestCases);
```

In Process Advisor, view the updated **Tasks** by clicking **Refresh Tasks** and switching to the **Project** view. For the task `MyCustomTask`, there is one task iteration for each model that has test cases. The

Process Advisor example project has two models that have test cases: AHRV_Voter and OuterLoop_Control.

Capabilities and Limitations

This table identifies functionality that is supported by the query.

Functionality	Supported?
Input query for task	No.
Iteration query for task	Yes. See IterationQuery.

See Also

padv.builtin.task.MergeTestResults | padv.builtin.query.FindModels |
padv.builtin.query.FindModels | padv.builtin.query.FindTestCasesForModel

padv.builtin.query.FindProjectFile Class

Namespace: padv.builtin.query padv.builtin.query

Superclasses: padv.Query

Query for finding project file

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.query.FindProjectFile` class provides a query that can return the current project file.

You can use this query in your process model to have your tasks iterate over the project one time or use the project file as an input.

The `padv.builtin.query.FindProjectFile` class is a `handle` class.

Creation

Description

`query = padv.builtin.query.FindProjectFile()` creates a query for finding the project file for the current project.

Properties

Title – Query title

"Current project" (default) | string | character vector

Query title, specified as a string or a character vector.

Example: "Find my current project file"

Data Types: string

DefaultArtifactType – Default artifact type returned by query

"project" (default) | ...

Default artifact type returned by the query, specified as one or more of the values in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method

Category	Artifact Type	Description
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram

Category	Artifact Type	Description
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "project"

Example: ["project" "other_file"]

Parent — Initial query that runs before iteration query

`padv.Query` | Name of `padv.Query` object

Initial query that runs before iteration query, specified as either a `padv.Query` object or the Name of a `padv.Query` object. When you specify an iteration query for a task, the parent query is the initial query that the build system runs before running the specified iteration query.

For information on how to improve Process Advisor load times by sharing query instances across your process model, see “Best Practices for Process Model Authoring”.

Example: `sharedQuery`

Example: "FindMyProjectFile"

Name — Unique identifier for query

"`padv.builtin.query.FindProjectFile`" (default) | string

Unique identifier for query, specified as a string.

Example: "FindMyProjectFile"

Data Types: string

ShowFileExtension — Show file extensions for returned artifacts

0 (false) | 1 (true)

Show file extensions in the **Alias** property of returned artifacts, specified as a numeric or logical 1 (`true`) or 0 (`false`). The **Alias** property controls the display name for the artifact in the **Tasks** column in Process Advisor.

By default, queries strip file extensions from the **Alias** property of each task iteration artifact. To show file extensions for all artifacts in the **Tasks** column, select the project setting **Show file**

extensions. To keep file extensions in the results for a specific query, specify the query property `ShowFileExtension` as `true`.

Example: `true`

Data Types: `logical`

SortArtifacts — Setting for automatically sorting artifacts by address

`true` or `1` (default) | `false` or `0`

Setting for automatically sorting artifacts by address, specified as a numeric or logical `1` (`true`) or `0` (`false`). When a query returns artifacts, the artifacts should be in a consistent order. By default, the build system sorts artifacts by the artifact address.

Alternatively, you can sort artifacts in a different order by overriding the internal `sortArtifacts` method in a subclass that defines a custom sort behavior. For more information, see “Sort Artifacts in Specific Order”.

The build system automatically calls the `sortArtifacts` method when using the process model. The `sortArtifacts` method expects two input arguments: a `padv.Query` object and a list of `padv.Artifact` objects returned by the `run` method. The `sortArtifacts` method should return a list of sorted `padv.Artifact` objects.

Example: `SortArtifacts = false`

Data Types: `logical`

FunctionHandle — Handle to function that function-based query runs

`function_handle`

Handle to the function that a function-based query runs, specified as a `function_handle`.

If you define your query functionality inside a function and you or the build system call `run` on the query, the query runs the function specified by the `function_handle`.

The built-in queries are defined inside classes and do not use the `FunctionHandle`.

Example: `FunctionHandle = @FunctionForQuery`

Data Types: `function_handle`

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Run query to find the artifacts that meet the criteria specified by the query.</p> <p>The query returns a <code>padv.Artifact</code> object or an array of <code>padv.Artifact</code> objects that represent those artifacts.</p> <hr/> <p>Note You do not need to manually invoke this method inside your process model. The build system automatically invokes the <code>run</code> method to find artifacts.</p> <hr/> <p>The <code>run</code> method inside this built-in query runs on a query object <code>obj</code> and returns artifacts <code>artifacts</code>. If you inherit from this built-in query, make sure to use the same method signature inside your custom query:</p> <pre>function artifacts = run(obj,~) ... end</pre>
-----	--

Examples

Find Project File for Task in Process

You can use the `FindProjectFile` query in your process model to find the project file for your tasks to iterate over (`IterationQuery`) and use as an input (`addInputQueries`). For example, to have a custom task run one time for your project and to use the project file as an input to the task, you can specify the `FindProjectFile` as the iteration query and as an input query for the task.

Open a project. For this example, open the Process Advisor example project.

```
processAdvisorExampleStart
```

Suppose that you have a custom task, `MyCustomTask`, that you add to your process model. You can use the built-in query `padv.builtin.query.FindProjectFile` to find the project file. Edit the process model to include this code:

```
taskObj = addTask(pm, "MyCustomTask", ...
    IterationQuery = padv.builtin.query.FindProjectFile(), ...
    InputQueries = padv.builtin.query.GetIterationArtifact());
```

When you specify `InputQueries` as `padv.builtin.query.GetIterationArtifact`, that allows the task to use the artifacts returned by `IterationQuery` as inputs to the task.

In Process Advisor, view the updated **Tasks** by clicking **Refresh Tasks** and switching to the **Project** view. For the task `MyCustomTask`, there is one task iteration for the project.

Test FindProjectFile Query Outside Process Model

Although you typically use a query inside your process model, you can run an instance of the `FindProjectFile` query outside of your process model to confirm which artifacts the query returns.

Open a project. For this example, you can open the Process Advisor example project.

```
processAdvisorExampleStart
```

Create an instance of the `FindProjectFile` query.

```
q = padv.builtin.query.FindProjectFile();
```

Run the query and inspect the array of artifacts that the query returns.

```
q.run
```

```
ans =
```

```
Artifact with properties:
```

```

    Type: "project"
    Parent: [0x0 padv.Artifact]
    ArtifactAddress: [1x1 padv.util.ArtifactAddress]
    Alias: [0x0 string]
```

The `ArtifactAddress` property contains additional information such as the `OwningProject` for projects in a project reference hierarchy.

Capabilities and Limitations

This table identifies functionality that is supported by the query.

Functionality	Supported?
Input query for task	Yes. See <code>addInputQueries</code> .
Iteration query for task	Yes. See <code>IterationQuery</code> .

See Also

`padv.Artifact` | `padv.builtin.query.FindArtifacts` | `padv.builtin.query.FindModels`

Topics

“Overview of Process Model”

“Modify Default Process Model to Fit Your Process”

padv.builtin.query.FindRefModels Class

Namespace: padv.builtin.query padv.builtin.query

Superclasses: padv.Query

Query for finding referenced models

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.query.FindRefModels` class provides a query that can return the referenced models in your project. A referenced model is a model that another model references. You can automatically include or exclude certain models by using the optional name-value arguments.

You can use this query in your process model to find referenced models for your tasks to iterate over. To find the top models in your project, you can use `padv.builtin.query.FindTopModels` instead.

The `padv.builtin.query.FindRefModels` class is a `handle` class.

Creation

Description

`query = padv.builtin.query.FindRefModels()` creates a query for finding the referenced models in your project.

`query = padv.builtin.query.FindRefModels(Name=Value)` sets certain properties using one or more name-value arguments. For example to find referenced models that include `Control` in the full file path, you can create and run the query `padv.builtin.query.FindRefModels(IncludePath="Control")`.

You can use this syntax to set property values for `Name`, `IncludeLabel`, `ExcludeLabel`, `IncludePath`, and `ExcludePath`.

The `padv.builtin.query.FindRefModels` class also has other properties, but you cannot set those properties during query creation.

Properties

IncludeLabel — Find referenced models with specific project label

cell array

Find referenced models with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: `{"Classification","Design"}`

Data Types: `cell`

ExcludeLabel — Exclude referenced models with specific project label

cell array

Exclude referenced models with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Classification","Design"}

Data Types: cell

IncludePath — Find referenced models where path contains specific text

string | character vector

Find referenced models where the path contains specific text, specified as a string.

Example: "Control"

Data Types: char | string

ExcludePath — Exclude referenced models where path contains specific text

string | character vector

Exclude referenced models where the path contains specific text, specified as a string or a character vector.

Example: "Control"

Data Types: string

Title — Query title

"All non-top models" (default) | string | character vector

Query title, specified as a string or a character vector.

Example: "Find my referenced models"

Data Types: string

DefaultArtifactType — Default artifact type returned by query

"sl_model_file" (default) | "zc_file" | ...

Default artifact type returned by the query, specified as one or more of the values in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file

Category	Artifact Type	Description
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result

Category	Artifact Type	Description
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "sl_model_file"

Example: ["sl_model_file" "zc_file"]

Parent — Initial query that runs before iteration query

padv.Query | Name of padv.Query object

Initial query that runs before iteration query, specified as either a padv.Query object or the Name of a padv.Query object. When you specify an iteration query for a task, the parent query is the initial query that the build system runs before running the specified iteration query.

For information on how to improve Process Advisor load times by sharing query instances across your process model, see “Best Practices for Process Model Authoring”.

Example: sharedQuery

Example: "FindMyInitialArtifacts"

Name — Unique identifier for query

string

Unique identifier for query, specified as a string.

Example: "FindMyRefModels"

Data Types: string

ShowFileExtension — Show file extensions for returned artifacts

0 (false) | 1 (true)

Show file extensions in the Alias property of returned artifacts, specified as a numeric or logical 1 (true) or 0 (false). The Alias property controls the display name for the artifact in the **Tasks** column in Process Advisor.

By default, queries strip file extensions from the Alias property of each task iteration artifact. To show file extensions for all artifacts in the **Tasks** column, select the project setting **Show file extensions**. To keep file extensions in the results for a specific query, specify the query property ShowFileExtension as true.

Example: true

Data Types: logical

SortArtifacts — Setting for automatically sorting artifacts by address

true or 1 (default) | false or 0

Setting for automatically sorting artifacts by address, specified as a numeric or logical 1 (`true`) or 0 (`false`). When a query returns artifacts, the artifacts should be in a consistent order. By default, the build system sorts artifacts by the artifact address.

Alternatively, you can sort artifacts in a different order by overriding the internal `sortArtifacts` method in a subclass that defines a custom sort behavior. For an example, see “Sort Artifacts in Specific Order”.

The build system automatically calls the `sortArtifacts` method when using the process model. The `sortArtifacts` method expects two input arguments: a `padv.Query` object and a list of `padv.Artifact` objects returned by the `run` method. The `sortArtifacts` method should return a list of sorted `padv.Artifact` objects.

Example: `SortArtifacts = false`

Data Types: `logical`

FunctionHandle — Handle to function that function-based query runs

`function_handle`

Handle to the function that a function-based query runs, specified as a `function_handle`.

If you define your query functionality inside a function and you or the build system call `run` on the query, the query runs the function specified by the `function_handle`.

The built-in queries are defined inside classes and do not use the `FunctionHandle`.

Example: `FunctionHandle = @FunctionForQuery`

Data Types: `function_handle`

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Run query to find the artifacts that meet the criteria specified by the query.</p> <p>The query returns a <code>padv.Artifact</code> object or an array of <code>padv.Artifact</code> objects that represent those artifacts.</p> <hr/> <p>Note You do not need to manually invoke this method inside your process model. The build system automatically invokes the <code>run</code> method to find artifacts.</p> <hr/> <p>The <code>run</code> method inside this built-in query runs on a query object <code>obj</code> and returns artifacts <code>artifacts</code>. If you inherit from this built-in query, make sure to use the same method signature inside your custom query:</p> <pre>function artifacts = run(obj,~) ... end</pre>
-----	--

Examples

Find Referenced Models for Task in Process

You can use the `FindRefModels` query in your process model to find artifacts for your tasks to iterate over (`IterationQuery`).

Open a project. For this example, open the Process Advisor example project.

```
processAdvisorExampleStart
```

Suppose that you have a custom task, `MyCustomTask`, that you add to your process model. To have the task run one time for each referenced model in the project, you can specify the built-in query `FindRefModels` as the iteration query for the task. Edit the process model to include this code:

```
taskObj = addTask(pm, "MyCustomTask", ...
    IterationQuery = padv.builtin.query.FindRefModels());
```

In Process Advisor, view the updated **Tasks** by clicking **Refresh Tasks** and switching to the **Project** view. For the task `MyCustomTask`, there is one task iteration for each referenced model.

If you wanted to filter out models that have `Control` in the full file path, you can modify the query properties to exclude those models. For example:

```
taskObj = addTask(pm, "MyCustomTask", ...
    IterationQuery = padv.builtin.query.FindRefModels(...
    ExcludePath = "Control"));
```

If you refresh Process Advisor, the **Tasks** column only shows the AHRS_Voter model because it does not have Control in its full file path.

Capabilities and Limitations

This table identifies functionality that is supported by the query.

Functionality	Supported?
Input query for task	No.
Iteration query for task	Yes. See IterationQuery.

See Also

`padv.Artifact` | `padv.builtin.query.FindModels` |
`padv.builtin.query.FindModelsWithLabel` |
`padv.builtin.query.FindModelsWithTestCases` | `padv.builtin.query.FindTopModels`

Topics

“Overview of Process Model”
“Modify Default Process Model to Fit Your Process”

padv.builtin.query.FindRequirements Class

Namespace: padv.builtin.query padv.builtin.query

Superclasses: padv.Query, padv.builtin.query.FindArtifacts

Query for finding requirements

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.query.FindRequirements` class provides a query that can return the requirements files in your project folder. You can automatically include or exclude certain files by using the optional name-value arguments.

You can use this query in your process model to find requirements files for your tasks to iterate over or use as inputs.

The `padv.builtin.query.FindRequirements` class is a handle class.

Creation

Description

`query = padv.builtin.query.FindRequirements()` creates a query for finding each of the requirements files (`.slreqx`) in your project folder.

`query = padv.builtin.query.FindRequirements(Name=Value)` sets certain properties using one or more name-value arguments. For example, `padv.builtin.query.FindRequirements(IncludePathRegex = "HighLevel.*\.slreqx")` creates a query that can find SLREQX files where the path contains HighLevel.

The `padv.builtin.query.FindRequirements` class also has other properties, but you cannot set those properties during query creation.

Input Arguments

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `query = padv.builtin.query.FindRequirements(IncludePathRegex = "HighLevel.*\.slreqx")`

Name — Unique identifier for query

string | character vector

Unique identifier for query, specified as a string or a character vector.

Example: "FindMyRequirements"

Data Types: `char` | `string`

IncludeLabel — Find requirements with specific project label

`cell` array

Find requirements with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: `{"Level", "System"}`

Data Types: `cell`

ExcludeLabel — Exclude requirements with specific project label

`cell` array

Exclude requirements with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: `{"Level", "System"}`

Data Types: `cell`

IncludePath — Find requirements where path contains specific text

`string`

Find requirements where the path contains specific text, specified as a string. Consider using `IncludePathRegex` instead.

Example: `"HighLevel"`

Data Types: `string`

ExcludePath — Exclude requirements where path contains specific text

`string` | character vector

Exclude requirements where the path contains specific text, specified as a string or a character vector. Consider using `ExcludePathRegex` instead.

Example: `"System"`

Data Types: `string`

IncludePathRegex — Find requirements where path matches regular expression pattern

`string` | character vector

Find requirements where the path matches a regular expression pattern, specified as a character vector or string. `IncludePathRegex` expects UNIX-style path separators.

If you want to use a literal path, use `IncludePath` instead. You can specify either `IncludePath` or `IncludePathRegex` but not both.

Example: `"reqPrefix.*\.slx"`

Data Types: `char` | `string`

ExcludePathRegex — Exclude requirements where path matches regular expression pattern

`string` | character vector

Exclude requirements where the path matches a regular expression pattern, specified as a string or a character vector. `ExcludePathRegex` expects UNIX-style path separators.

If you want to use a literal path, use `ExcludePath` instead. You can specify either `ExcludePath` or `ExcludePathRegex` but not both.

Example: `"reqPrefix.*\.slx"`

Data Types: `char` | `string`

Properties

ArtifactType — Type of artifact

`"mwreq_file"` (since R2024b) | `"sl_req_file"` | ...

Type of artifact, specified as one or more of the values in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	<code>"m_class"</code>	MATLAB class
	<code>"m_file"</code>	MATLAB file
	<code>"m_func"</code>	MATLAB function
	<code>"m_method"</code>	MATLAB class method
	<code>"m_property"</code>	MATLAB class property
Model Advisor	<code>"ma_config_file"</code>	Model Advisor configuration file
	<code>"ma_justification_file"</code>	Model Advisor justification file
Process Advisor	<code>"padv_dep_artifacts"</code>	Related artifacts that current artifact depends on
	<code>"padv_output_file"</code>	Process Advisor output file
Project	<code>"project"</code>	Current project file
Requirements	<code>"mwreq_item"</code>	Requirement (since R2024b)
	<code>"sl_req"</code>	Requirement (for R2024a and earlier)
	<code>"sl_req_file"</code>	Requirement file
	<code>"sl_req_table"</code>	Requirements Table
Stateflow	<code>"sf_chart"</code>	Stateflow chart
	<code>"sf_graphical_fcn"</code>	Stateflow graphical function
	<code>"sf_group"</code>	Stateflow group
	<code>"sf_state"</code>	Stateflow state
	<code>"sf_state_transition_chart"</code>	Stateflow state transition chart
	<code>"sf_truth_table"</code>	Stateflow truth table
Simulink	<code>"sl_block_diagram"</code>	Block diagram

Category	Artifact Type	Description
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "mwreq_file"

Example: ["mwreq_file" "other_file"]

IncludeLabel – Find requirements with specific project label

cell array

Find requirements with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Level", "System"}

Data Types: cell

ExcludeLabel — Exclude requirements with specific project label

cell array

Exclude requirements with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Level", "System"}

Data Types: cell

IncludePathRegex — Find requirements where path matches regular expression pattern

string | character vector

Find requirements where the path matches a regular expression pattern, specified as a string or character vector. `IncludePathRegex` expects UNIX-style path separators.

If you want to use a literal path, use `IncludePath` instead. You can specify either `IncludePath` or `IncludePathRegex` but not both.

Example: "HighLevel.*\\.slreqx"

Data Types: char | string

ExcludePathRegex — Exclude requirements where path matches regular expression pattern

string | character vector

Exclude requirements where the path matches a regular expression pattern, specified as a string or a character vector. `ExcludePathRegex` expects UNIX-style path separators.

If you want to use a literal path, use `ExcludePath` instead. You can specify either `ExcludePath` or `ExcludePathRegex` but not both.

Example: "HighLevel.*\\.slreqx"

Data Types: char | string

FilterSubFileArtifacts — Filter out sub-file artifacts from query results

1 (true) (default) | 0 (false)

Filter out sub-file artifacts from query results, specified as a numeric or logical 1 (true) or 0 (false).

A sub-file is a part of a larger file. For example, a subsystem is a sub-file of a model file.

Example: false

Data Types: logical

InProject — Include only requirements added to project

0 (false) (default) | 1 (true)

Include only requirements files that have been added to the project, specified as a numeric or logical 1 (true) or 0 (false).

For more information about how to add or remove files from a project, see “Manage Project Files”.

Note If you specify `InProject` as `true`, you cannot use the query as an input query.

Example: `true`

Attributes:

Dependent

Data Types: `logical`

Title – Query title

"All requirements in the project" (default) | string | character vector

Query title, specified as a string or a character vector.

Example: "Find my requirements"

Data Types: `string`

DefaultArtifactType – Default artifact type returned by query

"mwreq_file" (since R2024b) | "sl_req_file" | ...

Default artifact type returned by the query, specified as one or more of the values in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function

Category	Artifact Type	Description
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
"sl_test_suite_result"	Simulink Test suite result	

Example: "mwreq_file"

Example: ["mwreq_file" "other_file"]

Parent — Initial query that runs before iteration query

`padv.Query` | Name of `padv.Query` object

Initial query that runs before iteration query, specified as either a `padv.Query` object or the Name of a `padv.Query` object. When you specify an iteration query for a task, the parent query is the initial query that the build system runs before running the specified iteration query.

For information on how to improve Process Advisor load times by sharing query instances across your process model, see “Best Practices for Process Model Authoring”.

Example: `sharedQuery`

Example: "FindMyInitialArtifacts"

ShowFileExtension — Show file extensions for returned requirements

0 (false) | 1 (true)

Show file extensions in the `Alias` property of returned requirements, specified as a numeric or logical 1 (true) or 0 (false). The `Alias` property controls the display name for the artifact in the **Tasks** column in Process Advisor.

By default, queries strip file extensions from the `Alias` property of each task iteration artifact. To show file extensions for all artifacts in the **Tasks** column, select the project setting **Show file extensions**. To keep file extensions in the results for a specific query, specify the query property `ShowFileExtension` as `true`.

Example: `true`

Data Types: `logical`

SortArtifacts — Setting for automatically sorting requirements by address

`true` or 1 (default) | `false` or 0

Setting for automatically sorting requirements by address, specified as a numeric or logical 1 (true) or 0 (false). When a query returns artifacts, the artifacts should be in a consistent order. By default, the build system sorts artifacts by the artifact address.

Alternatively, you can sort artifacts in a different order by overriding the internal `sortArtifacts` method in a subclass that defines a custom sort behavior. For an example, see “Sort Artifacts in Specific Order”.

The build system automatically calls the `sortArtifacts` method when using the process model. The `sortArtifacts` method expects two input arguments: a `padv.Query` object and a list of `padv.Artifact` objects returned by the `run` method. The `sortArtifacts` method should return a list of sorted `padv.Artifact` objects.

Example: `SortArtifacts = false`

Data Types: `logical`

FunctionHandle — Handle to function that function-based query runs

`function_handle`

Handle to the function that a function-based query runs, specified as a `function_handle`.

If you define your query functionality inside a function and you or the build system call `run` on the query, the query runs the function specified by the `function_handle`.

The built-in queries are defined inside classes and do not use the `FunctionHandle`.

Example: `FunctionHandle = @FunctionForQuery`

Data Types: `function_handle`

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Run query to find the artifacts that meet the criteria specified by the query.</p> <p>The query returns a <code>padv.Artifact</code> object or an array of <code>padv.Artifact</code> objects that represent those artifacts.</p> <hr/> <p>Note You do not need to manually invoke this method inside your process model. The build system automatically invokes the <code>run</code> method to find artifacts.</p> <hr/> <p>The <code>run</code> method inside this built-in query runs on a query object <code>obj</code> and returns artifacts <code>artifacts</code>. If you inherit from this built-in query, make sure to use the same method signature inside your custom query:</p> <pre>function artifacts = run(obj,~) ... end</pre>
-----	--

Examples

Find Requirements for Task in Process

You can use the `FindRequirements` query in your task definition to find requirements files that your tasks can iterate over (`IterationQuery`) or use as inputs (`InputQueries`). For example, suppose that you want to add a custom task, `MyCustomTask`, that runs for each requirement in the project.

Open a project. For this example, you can open the Process Advisor example project.

```
processAdvisorExampleStart
```

By default, a new task runs one time for the project.

In the process model, you can add a new custom task and have the task iterate over each requirement in the project by specifying the task `IterationQuery` as `padv.builtin.query.FindRequirements`.

```
taskObj = addTask(pm, "MyCustomTask", ...  
    IterationQuery = padv.builtin.query.FindRequirements, ...  
    InputQueries = padv.builtin.query.GetIterationArtifact);
```

This code allows the task to use the artifacts returned by `IterationQuery` as inputs to the task by specifying `InputQueries` as `padv.builtin.query.GetIterationArtifact`.

For the Process Advisor example project, when you save the process model and refresh Process Advisor, the custom task **MyCustomTask** has task iterations for the requirement sets `HighLevelReqs` and `SystemReqs`.

Capabilities and Limitations

This table identifies functionality that is supported by the query.

Functionality	Supported?
Input query for task	Only when the query property <code>InProject</code> is <code>false</code> .
Iteration query for task	Yes.

See Also

`padv.builtin.query.FindArtifacts` | `padv.builtin.query.FindRequirementsForModel`

padv.builtin.query.FindRequirementsForModel Class

Namespace: padv.builtin.query padv.builtin.query

Superclasses: padv.Query

Query for finding requirements for model

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.query.FindRequirementsForModel` class provides a query that can return each of the requirements associated with a model. You can automatically include or exclude certain artifacts by using the optional name-value arguments.

You can use this query in your process model to find artifacts for your tasks to iterate over or use as inputs. To find each of the requirements in your project, you can use `padv.builtin.query.FindRequirements` instead.

The `padv.builtin.query.FindRequirementsForModel` class is a handle class.

Creation

Description

`query = padv.builtin.query.FindRequirementsForModel()` creates a query for finding each of the requirements associated with a model.

`query = padv.builtin.query.FindRequirementsForModel(Name=Value)` sets certain properties using one or more name-value arguments. For example, `padv.builtin.query.FindRequirementsForModel(IncludePath="System")` creates a query object to find requirements that include System in the full file path.

The `padv.builtin.query.FindRequirementsForModel` class also has other properties, but you cannot set those properties during query creation.

Input Arguments

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `query = padv.builtin.query.FindRequirementsForModel(IncludePath="System")`

ExcludeLabel — Exclude requirements with specific project label

cell array

Exclude requirements with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Category", "Label"}

Data Types: cell

ExcludePath — Exclude artifacts where path contains specific text

string | character vector

Exclude artifacts where the path contains specific text, specified as a string or a character vector.

Example: "HLR"

Data Types: string

IncludeLabel — Find artifacts with specific project label

cell array

Find artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Category", "Label"}

Data Types: cell

IncludePath — Find artifacts where path contains specific text

string

Find artifacts where the path contains specific text, specified as a string.

Example: "HLR"

Data Types: string

Name — Unique identifier for query

string

Unique identifier for query, specified as a string.

Example: "FindMyRequirements"

Data Types: string

Parent — Initial query run before iteration query

"padv.builtin.query.FindModels" (default) | padv.Query object | Name of padv.Query object

Initial query run before iteration query, specified as either a `padv.Query` object or the Name of a `padv.Query` object. When you specify a `padv.Query` object as the iteration query for a task, the Parent query is the initial query that the build system runs before running the specified iteration query.

The build system ignores the Parent query when you specify a query as an input query or dependency query for a task.

For information on how to improve Process Advisor load times by sharing query instances across your process model, see "Best Practices for Process Model Authoring".

Example: `padv.builtin.query.FindModels`

Example: `padv.builtin.query.FindModels(IncludePath = "Control")`

Properties

IncludeLabel — Find artifacts with specific project label

cell array

Find artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: `{"Category", "Label"}`

Data Types: `cell`

ExcludeLabel — Exclude artifacts with specific project label

cell array

Exclude artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: `{"Category", "Label"}`

Data Types: `cell`

IncludePath — Find artifacts where path contains specific text

string

Find artifacts where the path contains specific text, specified as a string.

Example: `"HLR"`

Data Types: `string`

ExcludePath — Exclude artifacts where path contains specific text

string | character vector

Exclude artifacts where the path contains specific text, specified as a string or a character vector.

Example: `"HLR"`

Data Types: `string`

Title — Query title

"All requirements for a model" (default) | string | character vector

Query title, specified as a string or a character vector.

Example: `"Find my requirements"`

Data Types: `string`

DefaultArtifactType — Default artifact type returned by query

"mwreq_file" (since R2024b) | "sl_req" | ...

Default artifact type returned by the query, specified as one or more of the values in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component

Category	Artifact Type	Description
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "sl_req"

Example: ["sl_req" "sl_req_file"]

Parent — Initial query run before iteration query

"padv.builtin.query.FindModels" (default) | padv.Query object | Name of padv.Query object

Initial query run before iteration query, specified as either a padv.Query object or the Name of a padv.Query object. When you specify a padv.Query object as the iteration query for a task, the Parent query is the initial query that the build system runs before running the specified iteration query.

The build system ignores the Parent query when you specify a query as an input query or dependency query for a task.

For information on how to improve Process Advisor load times by sharing query instances across your process model, see "Best Practices for Process Model Authoring".

Example: padv.builtin.query.FindModels

Example: padv.builtin.query.FindModels(IncludePath = "Control")

Name — Unique identifier for query

string

Unique identifier for query, specified as a string.

Example: "FindMyRequirements"

Data Types: string

ShowFileExtension — Show file extensions for returned artifacts`0 (false) | 1 (true)`

Show file extensions in the `Alias` property of returned artifacts, specified as a numeric or logical `1 (true)` or `0 (false)`. The `Alias` property controls the display name for the artifact in the **Tasks** column in Process Advisor.

By default, queries strip file extensions from the `Alias` property of each task iteration artifact. To show file extensions for all artifacts in the **Tasks** column, select the project setting **Show file extensions**. To keep file extensions in the results for a specific query, specify the query property `ShowFileExtension` as `true`.

Example: `true`

Data Types: `logical`

SortArtifacts — Setting for automatically sorting artifacts by address`true or 1 (default) | false or 0`

Setting for automatically sorting artifacts by address, specified as a numeric or logical `1 (true)` or `0 (false)`. When a query returns artifacts, the artifacts should be in a consistent order. By default, the build system sorts artifacts by the artifact address.

Alternatively, you can sort artifacts in a different order by overriding the internal `sortArtifacts` method in a subclass that defines a custom sort behavior. For an example, see “Sort Artifacts in Specific Order”.

The build system automatically calls the `sortArtifacts` method when using the process model. The `sortArtifacts` method expects two input arguments: a `padv.Query` object and a list of `padv.Artifact` objects returned by the `run` method. The `sortArtifacts` method should return a list of sorted `padv.Artifact` objects.

Example: `SortArtifacts = false`

Data Types: `logical`

FunctionHandle — Handle to function that function-based query runs`function_handle`

Handle to the function that a function-based query runs, specified as a `function_handle`.

If you define your query functionality inside a function and you or the build system call `run` on the query, the query runs the function specified by the `function_handle`.

The built-in queries are defined inside classes and do not use the `FunctionHandle`.

Example: `FunctionHandle = @FunctionForQuery`

Data Types: `function_handle`

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Run query to find the artifacts that meet the criteria specified by the query.</p> <p>The query returns a <code>padv.Artifact</code> object or an array of <code>padv.Artifact</code> objects that represent those artifacts.</p> <hr/> <p>Note You do not need to manually invoke this method inside your process model. The build system automatically invokes the run method to find artifacts.</p> <hr/> <p>The run method inside this built-in query runs on a query object <code>obj</code> and returns artifacts <code>artifacts</code> that are associated with the artifact <code>iterationArtifact</code>. If you inherit from this built-in query, make sure to use the same method signature inside your custom query:</p> <pre>function artifacts = run(obj,iterationArtifact) ... end</pre>
-----	---

Examples

Find and Run Tasks Using Requirements

You can use the `FindRequirementsForModel` query in your process model to find requirements for your tasks to iterate over (`IterationQuery`) or use as inputs (`addInputQueries`).

Open a project. For this example, you can open the Process Advisor example project.

```
processAdvisorExampleStart
```

Edit the process model to use the following process model instead.

```
function processmodel(pm)
    % Defines the project's processmodel

    arguments
        pm padv.ProcessModel
    end

    t1 = pm.addTask("TaskInputsAreReqs",...
        Iterationquery = padv.builtin.query.FindModels(...
        IncludePath = "AHRV_Voter.slx"),...
        InputQueries = padv.builtin.query.FindRequirementsForModel());

    t2 = pm.addTask("TaskIteratesOverModelReqs",...
        IterationQuery = padv.builtin.query.FindRequirementsForModel(...
        Parent = padv.builtin.query.FindModels(IncludePath = "AHRV_Voter.slx")));

end
```

This process model adds two custom tasks to the process. The code configures task `t1` to run on the model `InnerLoop_Control` and to use the requirements associated with that model as inputs to the task. The code configures task `t2` to run on each requirement associated with the model `InnerLoop_Control`.

In Process Advisor, view the updated **Tasks** by clicking **Refresh Tasks** and switching to the **Project** view.

Capabilities and Limitations

This table identifies functionality that is supported by the query.

Functionality	Supported?
Input query for task	Yes.
Iteration query for task	Yes.

Note If you use this query as an input query and specify non-empty values for `IncludeLabel`, `ExcludeLabel`, `IncludePath`, or `ExcludePath`, your task results can unexpectedly become outdated. If you see this behavior, consider using a different query, like `padv.builtin.query.FindArtifacts`, instead. For more information and a list of queries that are not impacted by this limitation, see “Other Limitations”.

See Also

`padv.builtin.query.FindRequirements`

padv.builtin.query.FindTestCasesForModel Class

Namespace: padv.builtin.query padv.builtin.query

Superclasses: padv.Query

Query for finding test cases for model

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.query.FindTestCasesForModel` class provides a query that can return the test cases associated with a model, including test cases associated with subsystem references. You can automatically include or exclude certain test cases by using the optional name-value arguments.

You can use this query in your process model to find test cases for your tasks to iterate over or use as inputs.

The `padv.builtin.query.FindTestCasesForModel` class is a handle class.

Creation

Description

`query = padv.builtin.query.FindTestCasesForModel()` creates a query for finding the test cases associated with a model, including test cases associated with subsystem references.

`query = padv.builtin.query.FindTestCasesForModel(Name=Value)` sets certain properties using one or more name-value arguments. For example, `padv.builtin.query.FindTestCasesForModel(Tags = "TagA")` creates a query object for finding test cases that use the test case tag "TagA".

Note If you use this query as an input query and specify non-empty values for `IncludeLabel`, `ExcludeLabel`, `IncludePath`, or `ExcludePath`, your task results can unexpectedly become outdated. If you see this behavior, consider using a different query, like `padv.builtin.query.FindArtifacts`, instead. For more information and a list of queries that are not impacted by this limitation, see "Other Limitations".

The `padv.builtin.query.FindTestCasesForModel` class also has other properties, but you cannot set those properties during query creation.

Input Arguments

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `query = padv.builtin.query.FindTestCasesForModel(Tags = "TagA")`

ExcludeLabel — Exclude artifacts with specific project label

cell array

Exclude artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Category", "Label"}

Data Types: cell

ExcludePath — Exclude artifacts where path contains specific text

string | character vector

Exclude artifacts where the path contains specific text, specified as a string or a character vector.

Example: "Control"

Data Types: string

IncludeLabel — Find artifacts with specific project label

cell array

Find artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Category", "Label"}

Data Types: cell

IncludePath — Find artifacts where path contains specific text

string

Find artifacts where the path contains specific text, specified as a string.

Example: "Control"

Data Types: string

Name — Unique identifier for query

string

Unique identifier for query, specified as a string.

Example: "FindMyTestCases"

Data Types: string

Parent — Initial query run before iteration query

"padv.builtin.query.FindModels" (default) | padv.Query object | Name of padv.Query object

Initial query run before iteration query, specified as either a padv.Query object or the Name of a padv.Query object. When you specify a padv.Query object as the iteration query for a task, the Parent query is the initial query that the build system runs before running the specified iteration query.

The build system ignores the Parent query when you specify a query as an input query or dependency query for a task.

For information on how to improve Process Advisor load times by sharing query instances across your process model, see "Best Practices for Process Model Authoring".

Example: `padv.builtin.query.FindModels`

Example: `padv.builtin.query.FindModels(IncludePath = "Control")`

Tags — Only include test cases with specific test case tags

string | character vector | cell array

Only include test cases that use a specific test case tag or tags, specified as a string, a character vector, or cell array.

Example: `"TagA"`

Example: `{"tag1", "tag2"}`

Data Types: `char` | `string` | `cell`

Properties

IncludeLabel — Find artifacts with specific project label

cell array

Find artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: `{"Category", "Label"}`

Data Types: `cell`

ExcludeLabel — Exclude artifacts with specific project label

cell array

Exclude artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: `{"Category", "Label"}`

Data Types: `cell`

IncludePath — Find artifacts where path contains specific text

string

Find artifacts where the path contains specific text, specified as a string.

Example: `"Control"`

Data Types: `string`

ExcludePath — Exclude artifacts where path contains specific text

string | character vector

Exclude artifacts where the path contains specific text, specified as a string or a character vector.

Example: `"Control"`

Data Types: `string`

Tags — Only include test cases with specific test case tags

string | character vector | cell array

Only include test cases that use a specific test case tag or tags, specified as a string, a character vector, or cell array.

Example: "TagA"

Example: {"tag1", "tag2"}

Data Types: char | string | cell

Title – Query title

"All test cases for a given model" (default) | string | character vector

Query title, specified as a string or a character vector.

Example: "Find my test cases"

Data Types: string

DefaultArtifactType – Default artifact type returned by query

"sl_test_case" (default) | ...

Default artifact type returned by the query, specified as one or more of the values in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart

Category	Artifact Type	Description
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
"sl_test_suite_result"	Simulink Test suite result	

Example: "sl_test_case"

Example: ["sl_test_case" "other_file"]

Parent – Initial query run before iteration query

"padv.builtin.query.FindModels" (default) | padv.Query object | Name of padv.Query object

Initial query run before iteration query, specified as either a `padv.Query` object or the `Name` of a `padv.Query` object. When you specify a `padv.Query` object as the iteration query for a task, the `Parent` query is the initial query that the build system runs before running the specified iteration query.

The build system ignores the `Parent` query when you specify a query as an input query or dependency query for a task.

For information on how to improve Process Advisor load times by sharing query instances across your process model, see “Best Practices for Process Model Authoring”.

Example: `padv.builtin.query.FindModels`

Example: `padv.builtin.query.FindModels(IncludePath = "Control")`

Name — Unique identifier for query

string

Unique identifier for query, specified as a string.

Example: "FindMyTestCases"

Data Types: string

ShowFileExtension — Show file extensions for returned artifacts

0 (false) | 1 (true)

Show file extensions in the `Alias` property of returned artifacts, specified as a numeric or logical 1 (`true`) or 0 (`false`). The `Alias` property controls the display name for the artifact in the **Tasks** column in Process Advisor.

By default, queries strip file extensions from the `Alias` property of each task iteration artifact. To show file extensions for all artifacts in the **Tasks** column, select the project setting **Show file extensions**. To keep file extensions in the results for a specific query, specify the query property `ShowFileExtension` as `true`.

Example: `true`

Data Types: logical

SortArtifacts — Setting for automatically sorting artifacts by address

`true` or 1 (default) | `false` or 0

Setting for automatically sorting artifacts by address, specified as a numeric or logical 1 (`true`) or 0 (`false`). When a query returns artifacts, the artifacts should be in a consistent order. By default, the build system sorts artifacts by the artifact address.

Alternatively, you can sort artifacts in a different order by overriding the internal `sortArtifacts` method in a subclass that defines a custom sort behavior. For an example, see “Sort Artifacts in Specific Order”.

The build system automatically calls the `sortArtifacts` method when using the process model. The `sortArtifacts` method expects two input arguments: a `padv.Query` object and a list of `padv.Artifact` objects returned by the `run` method. The `sortArtifacts` method should return a list of sorted `padv.Artifact` objects.

Example: `SortArtifacts = false`

Data Types: `logical`

FunctionHandle — Handle to function that function-based query runs

`function_handle`

Handle to the function that a function-based query runs, specified as a `function_handle`.

If you define your query functionality inside a function and you or the build system call `run` on the query, the query runs the function specified by the `function_handle`.

The built-in queries are defined inside classes and do not use the `FunctionHandle`.

Example: `FunctionHandle = @FunctionForQuery`

Data Types: `function_handle`

Methods

Specialized Public Methods

This class overrides the following inherited methods.

<code>run</code>	<p>Run query to find the artifacts that meet the criteria specified by the query.</p> <p>The query returns a <code>padv.Artifact</code> object or an array of <code>padv.Artifact</code> objects that represent those artifacts.</p> <hr/> <p>Note You do not need to manually invoke this method inside your process model. The build system automatically invokes the <code>run</code> method to find artifacts.</p> <hr/> <p>The <code>run</code> method inside this built-in query runs on a query object <code>obj</code> and returns artifacts <code>artifacts</code> that are associated with the artifact <code>iterationArtifact</code>. If you inherit from this built-in query, make sure to use the same method signature inside your custom query:</p> <pre>function artifacts = run(obj,iterationArtifact) ... end</pre>
------------------	---

Examples

Only Run Tests for Test Cases with Specific Tags

You can use the `FindTestCasesForModel` query in your process model to find test cases for your tasks to iterate over (`IterationQuery`) or use as inputs (`addInputQueries`). For example, suppose

that you want the **Run Tests** task to only run on test cases that use the specific test case tag `TagA`. You can use the built-in query `padv.builtin.query.FindTestCasesForModel` to find the test cases and the `Tags` input argument to have the query only return test cases that use the specified test case tag. Suppose you have a project that contains test cases that use the test case tags `TagA` and `TagB`.

Consider the following process model.

```
function processmodel(pm)
    % Defines the project's processmodel

    arguments
        pm padv.ProcessModel
    end

    % Run Tests for TagA
    milTaskA = addTask(pm,padv.builtin.task.RunTestsPerTestCase(Name = "RunTestsForTagA"));
    milTaskA.Title = "Run Tests for TagA";
    milTaskA.IterationQuery = padv.builtin.query.FindTestCasesForModel(Tags = "TagA");

    % Run Tests for TagB
    milTaskB = pm.addTask(padv.builtin.task.RunTestsPerTestCase(Name = "RunTestsForTagB"));
    milTaskB.Title = "Run Tests for TagB";
    milTaskB.IterationQuery = padv.builtin.query.FindTestCasesForModel(Tags = "TagB");

end
```

This process model uses `FindTestCasesForModel` as the iteration query for the built-in task `RunTestsPerTestCase` to have the task only run for each test case that uses the specified test case tag. Note that since this process model uses multiple instances of the task `RunTestsPerTestCase`, the code specifies unique `Name` values for each task.

In Process Advisor, view the updated **Tasks** by clicking **Refresh Tasks** and switching to the **Project** view. When you run the tasks, the tasks only run on the test cases with the specified test case tags. The test case names appear under the name of the model associated with the test case.

Capabilities and Limitations

This table identifies functionality that is supported by the query.

Functionality	Supported?
Input query for task	Yes.
Iteration query for task	Yes.

See Also

`padv.builtin.task.RunTestsPerModel` | `padv.builtin.task.RunTestsPerTestCase` | `padv.builtin.query.FindModelsWithTestCases`

padv.builtin.query.FindTopModels Class

Namespace: padv.builtin.query padv.builtin.query

Superclasses: padv.Query

Query for finding top models

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.query.FindTopModels` class provides a query that can return the top models in your project. A top model is a model that is not referenced by other models in the model hierarchy. You can automatically include or exclude certain models by using the optional name-value arguments.

You can use this query in your process model to find top models for your tasks to iterate over or use as inputs. To find the referenced models in your project, you can use `padv.builtin.query.FindRefModels` instead.

The `padv.builtin.query.FindTopModels` class is a handle class.

Creation

Description

`query = padv.builtin.query.FindTopModels()` creates a query for finding the top models in your project.

`query = padv.builtin.query.FindTopModels(Name=Value)` sets certain properties using one or more name-value arguments. For example to find top models that include `Control` in the full file path, you can create and run the query `padv.builtin.query.FindTopModels(IncludePath="Control")`.

You can use this syntax to set property values for `Name`, `IncludeLabel`, `ExcludeLabel`, `IncludePath`, and `ExcludePath`.

Note If you use this query as an input query and specify non-empty values for `IncludeLabel`, `ExcludeLabel`, `IncludePath`, or `ExcludePath`, your task results can unexpectedly become outdated. If you see this behavior, consider using a different query, like `padv.builtin.query.FindArtifacts`, instead. For more information and a list of queries that are not impacted by this limitation, see “Other Limitations”.

The `padv.builtin.query.FindTopModels` class also has other properties, but you cannot set those properties during query creation.

Properties

IncludeLabel — Find top models with specific project label

cell array

Find top models with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Classification","Design"}

Data Types: cell

ExcludeLabel — Exclude top models with specific project label

cell array

Exclude top models with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Classification","Design"}

Data Types: cell

IncludePath — Find top models where path contains specific text

string | character vector

Find top models where the path contains specific text, specified as a string.

Example: "Control"

Data Types: char | string

ExcludePath — Exclude top models where path contains specific text

string | character vector

Exclude top models where the path contains specific text, specified as a string or a character vector.

Example: "Control"

Data Types: string

Title — Query title

"All top models" (default) | string | character vector

Query title, specified as a string or a character vector.

Example: "Find my top models"

Data Types: string

DefaultArtifactType — Default artifact type returned by query

"sl_model_file" (default) | "zc_file" | ...

Default artifact type returned by the query, specified as one or more of the values in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property

Category	Artifact Type	Description
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file

Category	Artifact Type	Description
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "sl_model_file"

Example: ["sl_model_file" "zc_file"]

Parent — Initial query that runs before iteration query

`padv.Query` | Name of `padv.Query` object

Initial query that runs before iteration query, specified as either a `padv.Query` object or the Name of a `padv.Query` object. When you specify an iteration query for a task, the parent query is the initial query that the build system runs before running the specified iteration query.

For information on how to improve Process Advisor load times by sharing query instances across your process model, see "Best Practices for Process Model Authoring".

Example: `sharedQuery`

Example: "FindMyInitialArtifacts"

Name — Unique identifier for query

string

Unique identifier for query, specified as a string.

Example: "FindMyTopModels"

Data Types: `string`

ShowFileExtension — Show file extensions for returned artifacts

0 (false) | 1 (true)

Show file extensions in the `Alias` property of returned artifacts, specified as a numeric or logical 1 (`true`) or 0 (`false`). The `Alias` property controls the display name for the artifact in the **Tasks** column in Process Advisor.

By default, queries strip file extensions from the `Alias` property of each task iteration artifact. To show file extensions for all artifacts in the **Tasks** column, select the project setting **Show file extensions**. To keep file extensions in the results for a specific query, specify the query property `ShowFileExtension` as `true`.

Example: `true`

Data Types: `logical`

SortArtifacts — Setting for automatically sorting artifacts by address

`true` or `1` (default) | `false` or `0`

Setting for automatically sorting artifacts by address, specified as a numeric or logical `1` (`true`) or `0` (`false`). When a query returns artifacts, the artifacts should be in a consistent order. By default, the build system sorts artifacts by the artifact address.

Alternatively, you can sort artifacts in a different order by overriding the internal `sortArtifacts` method in a subclass that defines a custom sort behavior. For an example, see “Sort Artifacts in Specific Order”.

The build system automatically calls the `sortArtifacts` method when using the process model. The `sortArtifacts` method expects two input arguments: a `padv.Query` object and a list of `padv.Artifact` objects returned by the `run` method. The `sortArtifacts` method should return a list of sorted `padv.Artifact` objects.

Example: `SortArtifacts = false`

Data Types: `logical`

FunctionHandle — Handle to function that function-based query runs

`function_handle`

Handle to the function that a function-based query runs, specified as a `function_handle`.

If you define your query functionality inside a function and you or the build system call `run` on the query, the query runs the function specified by the `function_handle`.

The built-in queries are defined inside classes and do not use the `FunctionHandle`.

Example: `FunctionHandle = @FunctionForQuery`

Data Types: `function_handle`

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Run query to find the artifacts that meet the criteria specified by the query.</p> <p>The query returns a <code>padv.Artifact</code> object or an array of <code>padv.Artifact</code> objects that represent those artifacts.</p> <hr/> <p>Note You do not need to manually invoke this method inside your process model. The build system automatically invokes the <code>run</code> method to find artifacts.</p> <hr/> <p>The <code>run</code> method inside this built-in query runs on a query object <code>obj</code> and returns artifacts <code>artifacts</code>. If you inherit from this built-in query, make sure to use the same method signature inside your custom query:</p> <pre>function artifacts = run(obj,~) ... end</pre>
-----	--

Examples

Find Top Models for Task in Process

You can use the `FindTopModels` query in your process model to find artifacts that your tasks can iterate over (`IterationQuery`) or use as inputs (`InputQueries`).

Open a project. For this example, open the Process Advisor example project.

```
processAdvisorExampleStart
```

Suppose that you have a custom task, `MyCustomTask`, that you add to your process model. To have the task run one time for each top model in the project, you can specify the built-in query `FindTopModels` as the iteration query for the task. Edit the process model to include this code:

```
taskObj = addTask(pm, "MyCustomTask", ...
    IterationQuery = padv.builtin.query.FindTopModels());
```

In Process Advisor, view the updated **Tasks** by clicking **Refresh Tasks** and switching to the **Project** view. For the task `MyCustomTask`, there is one task iteration for the top model.

Capabilities and Limitations

This table identifies functionality that is supported by the query.

Functionality	Supported?
Input query for task	Yes. See

Functionality	Supported?
Iteration query for task	Yes. See IterationQuery.

See Also

`padv.Artifact` | `padv.builtin.query.FindModels` |
`padv.builtin.query.FindModelsWithLabel` |
`padv.builtin.query.FindModelsWithTestCases` | `padv.builtin.query.FindRefModels`

Topics

“Overview of Process Model”
“Modify Default Process Model to Fit Your Process”

padv.builtin.query.FindUnits Class

Namespace: padv.builtin.query padv.builtin.query

Superclasses: padv.Query

Query for finding units

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.query.FindUnits` class provides a query that can return the units in your project. The query uses the same unit classification as the Model Design and Model Testing Dashboards. A unit is a functional entity in your software architecture that you can execute and test independently or as part of larger system tests. Some software development standards, like certain testing objectives, apply specifically to units in a software architecture. You can use the `FindUnits` query to find the Simulink and System Composer models in your design that you need to assess. If you need to find both the units and components in your design, you can use the built-in query `padv.builtin.query.FindDesignModels` instead. For information how to classify the models in your project, see “Categorize Models in Hierarchy as Components or Units”.

You can use this query in your process model to find the units in your project and run tasks on those artifacts. For example, you can reconfigure the built-in task `padv.builtin.task.CollectMetrics` to collect model and code testing metrics for the units in your project by specifying the task iteration query as `padv.builtin.query.FindUnits`.

The `padv.builtin.query.FindUnits` class is a `handle` class.

Creation

Description

`query = padv.builtin.query.FindUnits` creates a query for finding the units in your project.

`query = padv.builtin.query.FindUnits(Name=Value)` sets certain properties using one or more name-value arguments. For example, `query = padv.builtin.query.FindUnits(ExcludePath = "Control")` creates a query that finds the units in the project, but excludes units that have "Control" in the file address.

The `padv.builtin.query.FindUnits` class also has other properties, but you cannot set those properties during query creation.

Input Arguments

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `query = padv.builtin.query.FindUnits(ExcludePath = "Control")`

ExcludeLabel — Exclude artifacts with specific project label

cell array

Exclude artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Classification", "Design"}

Data Types: cell

ExcludePath — Exclude artifacts where path contains specific text

string | character vector

Exclude artifacts where the path contains specific text, specified as a string or a character vector.

Example: "Control"

Data Types: char | string

IncludeLabel — Find artifacts with specific project label

cell array

Find artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Classification", "Design"}

Data Types: cell

IncludePath — Find artifacts where path contains specific text

string | character vector

Find artifacts where the path contains specific text, specified as a string or a character vector.

Example: "Control"

Data Types: char | string

Name — Unique identifier for query

string | character vector

Unique identifier for query, specified as a string or a character vector.

Example: "FindUnits"

Data Types: char | string

Properties**IncludeLabel — Find artifacts with specific project label**

cell array

Find artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Classification", "Design"}

Data Types: cell

ExcludeLabel — Exclude artifacts with specific project label

cell array

Exclude artifacts with specific project label, specified as a cell array where the first entry is the project label category and the second entry is the project label name.

Example: {"Classification", "Design"}

Data Types: cell

IncludePath — Find artifacts where path contains specific text

string | character vector

Find artifacts where the path contains specific text, specified as a string or a character vector.

Example: "Control"

Data Types: char | string

ExcludePath — Exclude artifacts where path contains specific text

string | character vector

Exclude artifacts where the path contains specific text, specified as a string or a character vector.

Example: "Control"

Data Types: char | string

Title — Query title

"All Unit models in project" (default) | string | character vector

Query title, specified as a string or a character vector.

Example: "Units"

Data Types: char | string

DefaultArtifactType — Default artifact type returned by query

"sl_model_file" (default) | "zc_file" | ...

Default artifact type returned by the query, specified as one or more of the values in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file

Category	Artifact Type	Description
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration

Category	Artifact Type	Description
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "zc_file"

Example: ["sl_model_file" "zc_file"]

Parent – Query that build system can run first

`padv.Query object` | Name of `padv.Query object`

Query that the build system can run first, specified as either a `padv.Query object` or the Name of a `padv.Query object`.

If you use `padv.builtin.query.FindUnits` as an iteration query in your process model, the build system automatically runs the Parent query first. If there is an existing query that you want the build system to run first, specify that query as the Parent query. For example, the built-in query `FindModelsWithTestCases` specifies `FindModels` as a Parent query.

If you use `padv.builtin.query.FindUnits` as an input query or dependency query for a task, the build system ignores the Parent query.

Name – Unique identifier for query

`string` | character vector

Unique identifier for query, specified as a string or a character vector.

Example: "FindUnits"

Data Types: `char` | `string`

ShowFileExtension – Show file extensions for returned artifacts

0 (false) | 1 (true)

Show file extensions in the Alias property of returned artifacts, specified as a numeric or logical 1 (true) or 0 (false). The Alias property controls the display name for the artifact in the **Tasks** column in Process Advisor.

By default, queries strip file extensions from the Alias property of each task iteration artifact. To show file extensions for all artifacts in the **Tasks** column, select the project setting **Show file extensions**. To keep file extensions in the results for a specific query, specify the query property `ShowFileExtension` as true.

Example: true

Data Types: `logical`

SortArtifacts — Setting for automatically sorting artifacts by address`true or 1 (default) | false or 0`

Setting for automatically sorting artifacts by address, specified as a numeric or logical 1 (`true`) or 0 (`false`). When a query returns artifacts, the artifacts should be in a consistent order. By default, the build system sorts artifacts by the artifact address.

Alternatively, you can sort artifacts in a different order by overriding the internal `sortArtifacts` method in a subclass that defines a custom sort behavior. For an example, see “Sort Artifacts in Specific Order”.

The build system automatically calls the `sortArtifacts` method when using the process model. The `sortArtifacts` method expects two input arguments: a `padv.Query` object and a list of `padv.Artifact` objects returned by the `run` method. The `sortArtifacts` method should return a list of sorted `padv.Artifact` objects.

Example: `SortArtifacts = false`

Data Types: `logical`

FunctionHandle — Handle to function that function-based query runs`function_handle`

Handle to the function that a function-based query runs, specified as a `function_handle`.

If you define your query functionality inside a function and you or the build system call `run` on the query, the query runs the function specified by the `function_handle`.

The built-in queries are defined inside classes and do not use the `FunctionHandle`.

Example: `FunctionHandle = @FunctionForQuery`

Data Types: `function_handle`

Methods**Specialized Public Methods**

This class overrides the following inherited methods.

run	<p>Run query to find the artifacts that meet the criteria specified by the query.</p> <p>The query returns a <code>padv.Artifact</code> object or an array of <code>padv.Artifact</code> objects that represent those artifacts.</p> <hr/> <p>Note You do not need to manually invoke this method inside your process model. The build system automatically invokes the <code>run</code> method to find artifacts.</p> <hr/> <p>The <code>run</code> method inside this built-in query runs on a query object <code>obj</code> and returns artifacts <code>artifacts</code>. If you inherit from this built-in query, make sure to use the same method signature inside your custom query:</p> <pre>function artifacts = run(obj,~) ... end</pre>
-----	--

Examples

Find Units for Task in Process

You can use the `FindUnits` query in your process model to find units and components that your tasks can iterate over (`IterationQuery`) or use as inputs (`InputQueries`). For example, suppose you have a custom task that you want to run for each unit in your project. You can find the units in your project by using the built-in query `FindUnits`.

Open a project. For this example, open the Process Advisor example project.

```
processAdvisorExampleStart
```

To have the custom task run for each unit, specify the `FindUnits` query as the iteration query for the task. For example, in your process model:

```
pm.addTask("MyCustomTask", ...
    IterationQuery = padv.builtin.query.FindUnits);
```

In Process Advisor, view the updated **Tasks** by clicking **Refresh Tasks** and switching to the **Project** view. For the task `MyCustomTask`, there is one task iteration for each unit in the project. The Process Advisor example project four units: `AHRS_Voter`, `Actuator_Control`, `InnerLoop_Control`, and `OuterLoop_Control`.

To view how the example project is classifying units and components, open the options for the Model Testing Dashboard.

```
modelTestingDashboard
```


In the dashboard toolstrip, click **Options**. The **Classification** section shows that the digital thread classifies models with the project label `Software Component` as components and models with the project label `Software Unit` as units.

For information how to classify the models in your project, see “Categorize Models in Hierarchy as Components or Units”.

Test Query Locally in Command Window

If you want to test a query before using the query in your process model, you can run the query directly from the MATLAB Command Window.

Open a project. For this example, open the Process Advisor example project.

```
processAdvisorExampleStart
```

In the MATLAB Command Window, create a query object that represents the query.

```
q = padv.builtin.query.FindUnits;
```

Run the query by using the `run` method. The query returns the artifacts that it finds as a `padv.Artifact` object or an array of `padv.Artifact` objects.

```
artifacts = q.run
```

```
artifacts =
```

```
    1×4 Artifact array with properties:
```

```
    Type
    Parent
    ArtifactAddress
    Alias
```

You can inspect the properties of each `padv.Artifact` object to view information about the artifact. For example, you can use the `Alias` property to view the artifact names.

```
artifacts.Alias
```

```
ans =
```

```
    "OuterLoop_Control.slx"
```

```
ans =
```

```
    "AHRS_Voter.slx"
```

```
ans =
```

```
    "Actuator_Control.slx"
```

```
ans =
```

```
    "InnerLoop_Control.slx"
```

The `Alias` property returns the artifact names as they appear in Process Advisor.

Capabilities and Limitations

This table identifies functionality that is supported by the query.

Functionality	Supported?
Input query for task	Yes. See <code>InputQueries</code> .
Iteration query for task	Yes. See <code>IterationQuery</code> .

Note If you use this query as an input query and specify non-empty values for `IncludeLabel`, `ExcludeLabel`, `IncludePath`, or `ExcludePath`, your task results can unexpectedly become outdated. If you see this behavior, consider using a different query, like `padv.builtin.query.FindArtifacts`, instead. For more information and a list of queries that are not impacted by this limitation, see “Other Limitations”.

See Also

`padv.Artifact` | `padv.builtin.task.CollectMetrics` |
`padv.builtin.query.FindDesignModels`

Topics

“Categorize Models in Hierarchy as Components or Units”

padv.builtin.query.GetDependentArtifacts Class

Namespace: padv.builtin.query padv.builtin.query

Superclasses: padv.Query

Query for getting dependent artifacts

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.query.GetDependentArtifacts` class provides a query that can return the dependent artifacts for a specific artifact.

You can use this query in your process model to find artifacts for your task to use as inputs. But typically, you use the `GetDependentArtifacts` query as the `InputDependencyQuery` for a task to find additional artifact dependencies for your task inputs, since those additional artifacts can also impact whether the task results are up-to-date. For example, if you specify a model as an input to a task and you use `GetDependentArtifacts` as the `InputDependencyQuery`, the build system can find the related artifacts that the model depends on, such as data dictionaries and other artifact dependencies. For more information, see "Overview of Process Model".

The `padv.builtin.query.GetDependentArtifacts` class is a `handle` class.

Creation

Description

`query = padv.builtin.query.GetDependentArtifacts()` creates a query for finding the dependent artifacts for a specific artifact. Typically, you use this query inside your process model or task definition to find the dependent artifacts for a task input.

Properties

Title — Query title

"Get dependent artifacts for a given artifact" (default) | string | character vector

Query title, specified as a string or a character vector.

Example: "Find related artifacts"

Data Types: char | string

DefaultArtifactType — Default artifact type returned by query

"padv_dep_artifacts" (default) | ...

Default artifact type returned by the query, specified as one or more of the values in this table. To specify multiple values, use an array.

Category	Artifact Type	Description
MATLAB	"m_class"	MATLAB class
	"m_file"	MATLAB file
	"m_func"	MATLAB function
	"m_method"	MATLAB class method
	"m_property"	MATLAB class property
Model Advisor	"ma_config_file"	Model Advisor configuration file
	"ma_justification_file"	Model Advisor justification file
Process Advisor	"padv_dep_artifacts"	Related artifacts that current artifact depends on
	"padv_output_file"	Process Advisor output file
Project	"project"	Current project file
Requirements	"mwreq_item"	Requirement (since R2024b)
	"sl_req"	Requirement (for R2024a and earlier)
	"sl_req_file"	Requirement file
	"sl_req_table"	Requirements Table
Stateflow	"sf_chart"	Stateflow chart
	"sf_graphical_fcn"	Stateflow graphical function
	"sf_group"	Stateflow group
	"sf_state"	Stateflow state
	"sf_state_transition_chart"	Stateflow state transition chart
	"sf_truth_table"	Stateflow truth table
Simulink	"sl_block_diagram"	Block diagram
	"sl_data_dictionary_file"	Data dictionary file
	"sl_embedded_matlab_fcn"	MATLAB function
	"sl_block_diagram"	Block diagram
	"sl_library_file"	Library file
	"sl_model_file"	Simulink model file
	"sl_protected_model_file"	Protected Simulink model file
	"sl_subsystem"	Subsystem
	"sl_subsystem_file"	Subsystem file
	"sl_subsystem"	Subsystem
System Composer	"zc_block_diagram"	System Composer architecture
	"zc_component"	System Composer architecture component

Category	Artifact Type	Description
	"zc_file"	System Composer architecture file
Tests	"harness_info_file"	Harness info file
	"sl_harness_block_diagram"	Harness block diagram
	"sl_harness_file"	Test harness file
	"sl_test_case"	Simulink Test case
	"sl_test_case_result"	Simulink Test case result
	"sl_test_file"	Simulink Test file
	"sl_test_iteration"	Simulink Test iteration
	"sl_test_iteration_result"	Simulink Test iteration result
	"sl_test_report_file"	Simulink Test result report
	"sl_test_result_file"	Simulink Test result file
	"sl_test_resultset"	Simulink Test result set
	"sl_test_seq"	Test Sequence
	"sl_test_suite"	Simulink Test suite
	"sl_test_suite_result"	Simulink Test suite result

Example: "padv_dep_artifacts"

Example: ["padv_dep_artifacts" "other_file"]

Parent – Initial query that runs before iteration query

padv.Query | Name of padv.Query object

Initial query that runs before iteration query, specified as either a padv.Query object or the Name of a padv.Query object. When you specify an iteration query for a task, the parent query is the initial query that the build system runs before running the specified iteration query.

For information on how to improve Process Advisor load times by sharing query instances across your process model, see "Best Practices for Process Model Authoring".

Example: sharedQuery

Example: "FindMyInitialArtifacts"

Name – Unique identifier for query

"padv.builtin.query.GetDependentArtifacts" (default) | string

Unique identifier for query, specified as a string.

Example: "FindDependentArtifacts"

Data Types: string

ShowFileExtension – Show file extensions for returned artifacts

0 (false) | 1 (true)

Show file extensions in the `Alias` property of returned artifacts, specified as a numeric or logical `1` (`true`) or `0` (`false`). The `Alias` property controls the display name for the artifact in the **Tasks** column in Process Advisor.

By default, queries strip file extensions from the `Alias` property of each task iteration artifact. To show file extensions for all artifacts in the **Tasks** column, select the project setting **Show file extensions**. To keep file extensions in the results for a specific query, specify the query property `ShowFileExtension` as `true`.

Example: `true`

Data Types: `logical`

SortArtifacts — Setting for automatically sorting artifacts by address

`true` or `1` (default) | `false` or `0`

Setting for automatically sorting artifacts by address, specified as a numeric or logical `1` (`true`) or `0` (`false`). When a query returns artifacts, the artifacts should be in a consistent order. By default, the build system sorts artifacts by the artifact address.

Alternatively, you can sort artifacts in a different order by overriding the internal `sortArtifacts` method in a subclass that defines a custom sort behavior. For an example, see “Sort Artifacts in Specific Order”.

The build system automatically calls the `sortArtifacts` method when using the process model. The `sortArtifacts` method expects two input arguments: a `padv.Query` object and a list of `padv.Artifact` objects returned by the `run` method. The `sortArtifacts` method should return a list of sorted `padv.Artifact` objects.

Example: `SortArtifacts = false`

Data Types: `logical`

FunctionHandle — Handle to function that function-based query runs

`function_handle`

Handle to the function that a function-based query runs, specified as a `function_handle`.

If you define your query functionality inside a function and you or the build system call `run` on the query, the query runs the function specified by the `function_handle`.

The built-in queries are defined inside classes and do not use the `FunctionHandle`.

Example: `FunctionHandle = @FunctionForQuery`

Data Types: `function_handle`

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Run query to find the artifacts that meet the criteria specified by the query.</p> <p>The query returns a <code>padv.Artifact</code> object or an array of <code>padv.Artifact</code> objects that represent those artifacts.</p> <hr/> <p>Note You do not need to manually invoke this method inside your process model. The build system automatically invokes the <code>run</code> method to find artifacts.</p> <hr/> <p>The <code>run</code> method inside this built-in query runs on a query object <code>obj</code> and returns artifacts that are associated with the artifact <code>iterationArtifact</code>. If you inherit from this built-in query, make sure to use the same method signature inside your custom query:</p> <pre>function artifacts = run(obj,iterationArtifact) ... end</pre>
-----	--

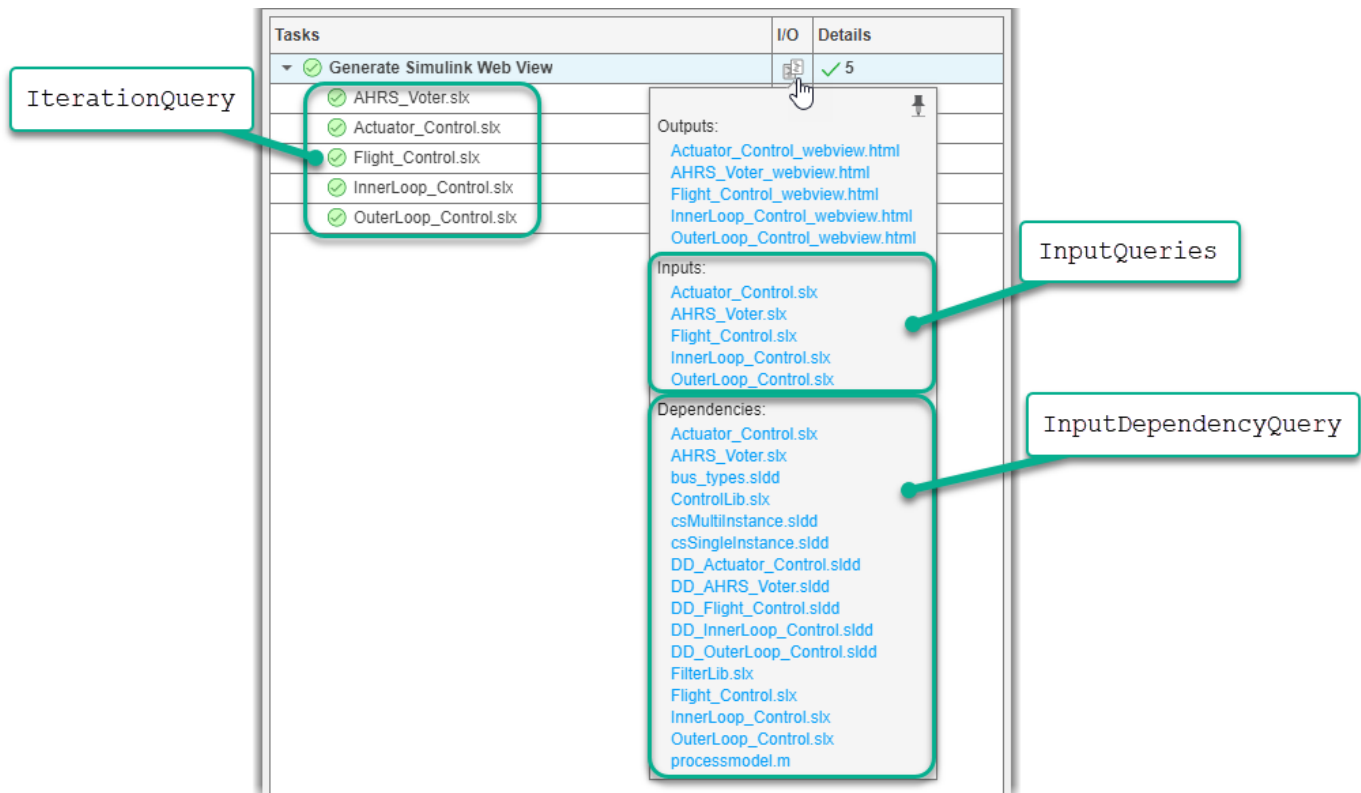
Examples

Find Related Artifact Dependencies for Task Inputs

Suppose that you have a task that uses models as task inputs. You can automatically find the related artifacts that the model depends on, such as data dictionaries, by specifying `GetDependentArtifacts` as the input dependency query for the task.

The built-in tasks typically use `GetDependentArtifacts` as the input dependency query. For example, the built-in task `GenerateSimulinkWebView` uses the `GetDependentArtifacts` query to find the data dictionaries and libraries that the models in the project depend on. The build system runs the `InputDependencyQuery` on each task input to find additional dependencies that can impact if task results are up-to-date.

In Process Advisor, the **Tasks** column shows the artifacts that the task iterates over. When you point to task results in the **I/O** column, you can see the task inputs and input dependencies.



Suppose that you define a custom task, `MyCustomTask`, inside a class file. You define when the task runs and the task inputs and dependencies by setting the task properties:

- `IterationQuery` — Determines how often the task runs by finding the artifacts that the task iterates over. For example, the task can find and iterate over the models in the project.
- `InputQueries` — Finds inputs for the task. For example, the task can find the current task iteration artifact and use that artifact as a task input.
- `InputDependencyQuery` — Finds additional dependencies related to the task inputs. For example, the task can find additional, related artifacts that the models in the project depend on.

Inside the class definition file, you can specify:

```
classdef MyCustomTask < padv.Task
    methods
        function obj = MyCustomTask(options)
            arguments
                options.Name = "MyCustomTask";
                options.IterationQuery = "padv.builtin.query.FindModels";
                options.InputQueries = "padv.builtin.query.GetIterationArtifact";
                % For each input, find dependencies that impact if the
                % task results are up-to-date
                options.InputDependencyQuery = padv.builtin.query.GetDependentArtifacts;
            end

            obj@padv.Task(options.Name,...
                IterationQuery=options.IterationQuery,...
                InputQueries=options.InputQueries,...
                InputDependencyQuery=options.InputDependencyQuery);
        end
    end
end
```



```

    end
    function taskResult = run(obj,input)
        taskResult = padv.TaskResult;
        taskResult.Status = padv.TaskStatus.Pass;
    end
end
end
end

```

When you specify `InputQueries` as `padv.builtin.query.GetIterationArtifact`, that allows the task to use the artifacts returned by `IterationQuery` as inputs to the task.

The build system runs the `IterationQuery` to determine which artifacts to run the task for. The build system then creates a task iteration, runs additional queries the task needs, runs the task, and saves the task results. For each task iteration, the build system runs the `InputQueries` to find the inputs for that specific task iteration. For each input, the build system runs the `InputDependencyQuery` to find additional dependencies that can impact if task results are up-to-date.

In this example custom task, the input dependency query finds the artifact dependencies for each model in the project. For information on custom tasks, see “Create Custom Tasks”.

Capabilities and Limitations

This table identifies functionality that is supported by the query.

Functionality	Supported?
Input query for task	Yes.
Iteration query for task	No.

See Also

`padv.builtin.query.GetIterationArtifact` |
`padv.builtin.query.GetOutputsOfDependentTask`

Topics

“Create Custom Tasks”
“Overview of Process Model”

padv.builtin.query.GetIterationArtifact Class

Namespace: padv.builtin.query padv.builtin.query

Superclasses: padv.Query

Query for getting current task iteration artifact

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.query.GetIterationArtifact` class provides a query that can return the current task iteration artifact.

You can use this query in your process model to find artifacts for your task to use as inputs. For example, if a task iterates over the models in the project, you can use the `GetIterationArtifact` query as an input query for the task to use the models in the project as inputs to the task.

The `padv.builtin.query.GetIterationArtifact` class is a handle class.

Creation

Description

`query = padv.builtin.query.GetIterationArtifact()` creates a query for getting the current task iteration artifact.

Properties

Title — Query title

"Current iteration artifact" (default) | string | character vector

Query title, specified as a string or a character vector.

Example: "Get artifact that task is iterating over"

Data Types: char | string

Parent — Initial query that runs before iteration query

padv.Query | Name of padv.Query object

Initial query that runs before iteration query, specified as either a `padv.Query` object or the Name of a `padv.Query` object. When you specify an iteration query for a task, the parent query is the initial query that the build system runs before running the specified iteration query.

For information on how to improve Process Advisor load times by sharing query instances across your process model, see "Best Practices for Process Model Authoring".

Example: `sharedQuery`

Example: "FindMyInitialArtifacts"

Name — Unique identifier for query`"padv.builtin.query.GetIterationArtifact" (default) | string`

Unique identifier for query, specified as a string.

Example: `"FindCurrentIterationArtifact"`

Data Types: `string`

ShowFileExtension — Show file extensions for returned artifacts`0 (false) | 1 (true)`

Show file extensions in the `Alias` property of returned artifacts, specified as a numeric or logical `1 (true)` or `0 (false)`. The `Alias` property controls the display name for the artifact in the **Tasks** column in Process Advisor.

By default, queries strip file extensions from the `Alias` property of each task iteration artifact. To show file extensions for all artifacts in the **Tasks** column, select the project setting **Show file extensions**. To keep file extensions in the results for a specific query, specify the query property `ShowFileExtension` as `true`.

Example: `true`

Data Types: `logical`

SortArtifacts — Setting for automatically sorting artifacts by address`true or 1 (default) | false or 0`

Setting for automatically sorting artifacts by address, specified as a numeric or logical `1 (true)` or `0 (false)`. When a query returns artifacts, the artifacts should be in a consistent order. By default, the build system sorts artifacts by the artifact address.

Alternatively, you can sort artifacts in a different order by overriding the internal `sortArtifacts` method in a subclass that defines a custom sort behavior. For an example, see “Sort Artifacts in Specific Order”.

The build system automatically calls the `sortArtifacts` method when using the process model. The `sortArtifacts` method expects two input arguments: a `padv.Query` object and a list of `padv.Artifact` objects returned by the `run` method. The `sortArtifacts` method should return a list of sorted `padv.Artifact` objects.

Example: `SortArtifacts = false`

Data Types: `logical`

FunctionHandle — Handle to function that function-based query runs`function_handle`

Handle to the function that a function-based query runs, specified as a `function_handle`.

If you define your query functionality inside a function and you or the build system call `run` on the query, the query runs the function specified by the `function_handle`.

The built-in queries are defined inside classes and do not use the `FunctionHandle`.

Example: `FunctionHandle = @FunctionForQuery`

Data Types: `function_handle`

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Run query to find the artifacts that meet the criteria specified by the query.</p> <p>The query returns a <code>padv.Artifact</code> object or an array of <code>padv.Artifact</code> objects that represent those artifacts.</p> <hr/> <p>Note You do not need to manually invoke this method inside your process model. The build system automatically invokes the <code>run</code> method to find artifacts.</p> <hr/> <p>The <code>run</code> method inside this built-in query returns the iteration artifact <code>iterationArtifact</code>. If you inherit from this built-in query, make sure to use the same method signature inside your custom query:</p> <pre>function artifact = run(~,iterationArtifact) artifact = iterationArtifact; end</pre>
-----	--

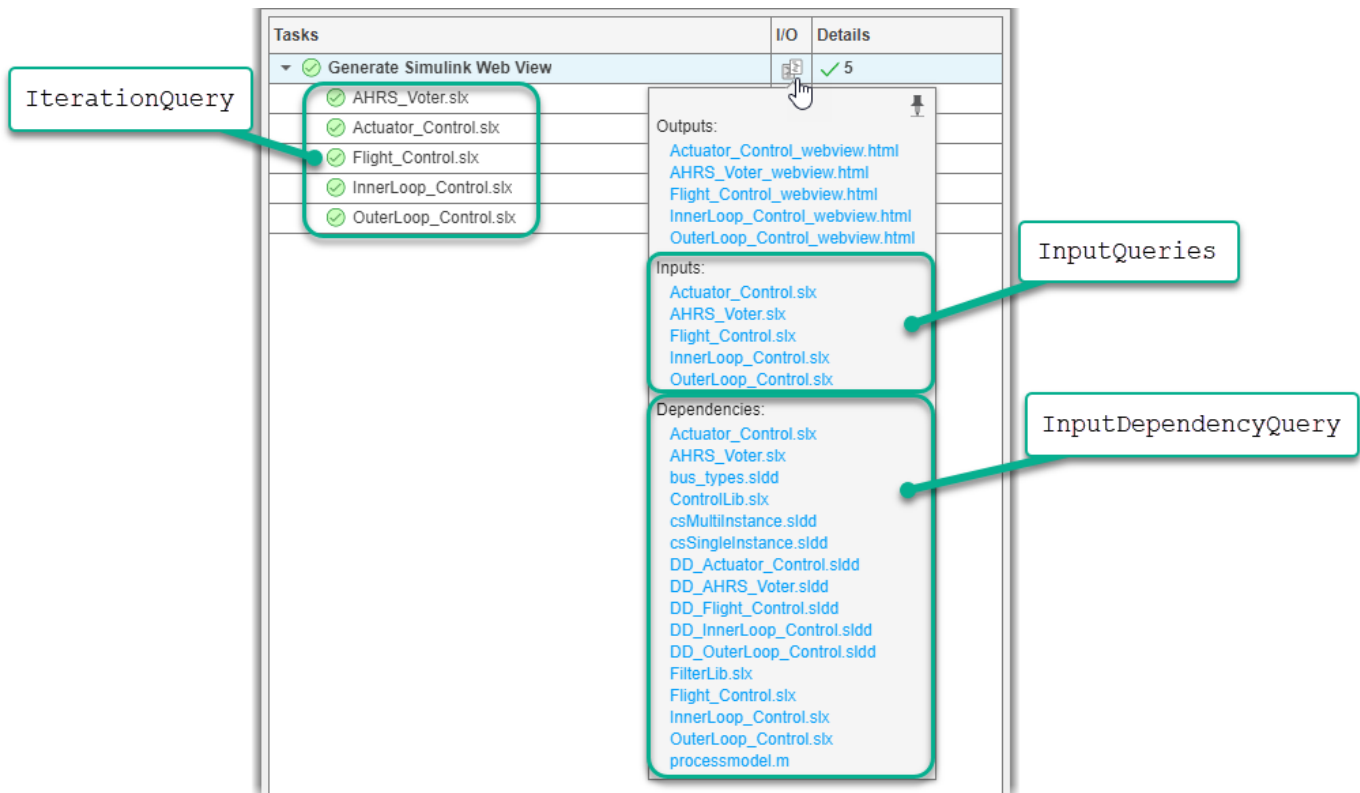
Examples

Get Current Task Iteration Artifact for Task

Suppose that you have a task that iterates over each model in the project. Although the task iterates over those models, the task does not use those models as inputs unless you specify the input query for the task. To use the models as inputs to the task, you can specify `GetIterationArtifact` as an input query for the task.

The built-in tasks typically use `GetIterationArtifact` as an input query. For example, the built-in task `GenerateSimulinkWebView` uses the `GetIterationArtifact` query as an input query to use the models that the task iterates over as inputs to the task.

In Process Advisor, the **Tasks** column shows the artifacts that the task iterates over. When you point to task results in the **I/O** column, you can see the task inputs and input dependencies.



Suppose that you define a custom task, `MyCustomTask`, inside a class file. You define when the task runs and the task inputs by setting the task properties:

- `IterationQuery` — Determines how often the task runs by finding the artifacts that the task iterates over. For example, the task can find and iterate over the models in the project.
- `InputQueries` — Finds inputs for the task. For example, the task can find the current task iteration artifact, such as a model, and use that artifact as a task input.

Inside the class definition file, you can specify:

```
classdef MyCustomTask < padv.Task
    methods
        function obj = MyCustomTask(options)
            arguments
                options.Name = "MyCustomTask";
                options.IterationQuery = "padv.builtin.query.FindModels";
                options.InputQueries = "padv.builtin.query.GetIterationArtifact";
            end

            obj@padv.Task(options.Name,...
                IterationQuery=options.IterationQuery,...
                InputQueries=options.InputQueries);
        end
        function taskResult = run(obj,input)
            taskResult = padv.TaskResult;
            taskResult.Status = padv.TaskStatus.Pass;
        end
    end
end
```

```
end  
end
```

When you specify `InputQueries` as `padv.builtin.query.GetIterationArtifact`, that allows the task to use the artifacts returned by `IterationQuery` as inputs to the task. In this example custom task, the iteration artifacts are the models in the project.

The build system runs the `IterationQuery` to determine which artifacts to run the task for. The build system then creates a task iteration, runs additional queries the task needs, runs the task, and saves the task results. For each task iteration, the build system runs the `InputQueries` to find the inputs for that specific task iteration.

For information on custom tasks, see “Create Custom Tasks”.

Capabilities and Limitations

This table identifies functionality that is supported by the query.

Functionality	Supported?
Input query for task	Yes.
Iteration query for task	No.

See Also

`padv.builtin.query.GetDependentArtifacts` |
`padv.builtin.query.GetOutputsOfDependentTask`

Topics

“Create Custom Tasks”
“Overview of Process Model”

padv.builtin.query.GetOutputsOfDependentTask Class

Namespace: padv.builtin.query padv.builtin.query

Superclasses: padv.Query

Query for getting outputs from predecessor task

Description

This class requires CI/CD Automation for Simulink Check.

The `padv.builtin.query.GetOutputsOfDependentTask` class provides a query that can return the outputs from a predecessor task.

You can use this query in your process model or task definition to find artifacts for your task to use as inputs. Typically, you use this query to pass the outputs of one task as inputs to another task. For example, if `TaskB` depends on `TaskA`, then `TaskA` is a predecessor task. You can pass the outputs from `TaskA` to `TaskB` by specifying `GetOutputsOfDependentTask` as an input query for `TaskB`.

The `padv.builtin.query.GetOutputsOfDependentTask` class is a handle class.

Creation

Description

`query = padv.builtin.query.GetOutputsOfDependentTask()` creates a query for getting the outputs from the predecessor task. You must define the predecessor task by using the function `dependsOn` on the task objects.

`query = padv.builtin.query.GetOutputsOfDependentTask(____, Name=Value)` sets certain properties using one or more name-value arguments. For example, `padv.builtin.query.GetOutputsOfDependentTask(Name="MyQuery", Task="TaskA")` creates a query named `MyQuery` that finds the outputs from `TaskA`.

The `padv.builtin.query.GetOutputsOfDependentTask` class also has other properties, but you cannot set those properties during query creation.

Input Arguments

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example:

```
padv.builtin.query.GetOutputsOfDependentTask(Name="MyQuery", Task="TaskA")
```

taskNameorInstance — Task name or instance

string | padv.Task object

Task name or instance, specified as a string or `padv.Task` object.

This argument specifies the value of the `DependentTask` property for the class instance.

Example: `"NameOfMyTask"`

Example: `padv.builtin.task.RunModelStandards`

queryName — Name of query

string

Name of query, specified as a string.

This argument specifies the value of the `Name` property for the class instance.

Example: `"NameOfMyQuery"`

Properties

DependentTask — Predecessor task name or instance

string | `padv.Task` object

Predecessor task name or instance, specified as a string or `padv.Task` object.

Example: `"NameOfMyTask"`

Example: `padv.builtin.task.RunModelStandards`

Data Types: string

Title — Query title

"Outputs of the dependent task" (default) | string | character vector

Query title, specified as a string or a character vector.

Example: `"Get artifacts from previous task"`

Data Types: char | string

Parent — Initial query that runs before iteration query

`padv.Query` | Name of `padv.Query` object

Initial query that runs before iteration query, specified as either a `padv.Query` object or the `Name` of a `padv.Query` object. When you specify an iteration query for a task, the parent query is the initial query that the build system runs before running the specified iteration query.

For information on how to improve Process Advisor load times by sharing query instances across your process model, see "Best Practices for Process Model Authoring".

Example: `sharedQuery`

Example: `"FindMyInitialArtifacts"`

Name — Unique identifier for query

"`padv.builtin.query.GetOutputsOfDependentTask`" (default) | string

Unique identifier for query, specified as a string.

Example: `"GetOutputsFromTask"`

Data Types: `string`

ShowFileExtension — Show file extensions for returned artifacts

`0 (false) | 1 (true)`

Show file extensions in the `Alias` property of returned artifacts, specified as a numeric or logical `1 (true)` or `0 (false)`. The `Alias` property controls the display name for the artifact in the **Tasks** column in Process Advisor.

By default, queries strip file extensions from the `Alias` property of each task iteration artifact. To show file extensions for all artifacts in the **Tasks** column, select the project setting **Show file extensions**. To keep file extensions in the results for a specific query, specify the query property `ShowFileExtension` as `true`.

Example: `true`

Data Types: `logical`

SortArtifacts — Setting for automatically sorting artifacts by address

`true or 1 (default) | false or 0`

Setting for automatically sorting artifacts by address, specified as a numeric or logical `1 (true)` or `0 (false)`. When a query returns artifacts, the artifacts should be in a consistent order. By default, the build system sorts artifacts by the artifact address.

Alternatively, you can sort artifacts in a different order by overriding the internal `sortArtifacts` method in a subclass that defines a custom sort behavior. For an example, see “Sort Artifacts in Specific Order”.

The build system automatically calls the `sortArtifacts` method when using the process model. The `sortArtifacts` method expects two input arguments: a `padv.Query` object and a list of `padv.Artifact` objects returned by the `run` method. The `sortArtifacts` method should return a list of sorted `padv.Artifact` objects.

Example: `SortArtifacts = false`

Data Types: `logical`

FunctionHandle — Handle to function that function-based query runs

`function_handle`

Handle to the function that a function-based query runs, specified as a `function_handle`.

If you define your query functionality inside a function and you or the build system call `run` on the query, the query runs the function specified by the `function_handle`.

The built-in queries are defined inside classes and do not use the `FunctionHandle`.

Example: `FunctionHandle = @FunctionForQuery`

Data Types: `function_handle`

Methods

Specialized Public Methods

This class overrides the following inherited methods.

run	<p>Run query to find the artifacts that meet the criteria specified by the query.</p> <p>The query returns a <code>padv.Artifact</code> object or an array of <code>padv.Artifact</code> objects that represent those artifacts.</p> <hr/> <p>Note You do not need to manually invoke this method inside your process model. The build system automatically invokes the <code>run</code> method to find artifacts.</p> <hr/> <p>The <code>run</code> method inside this built-in query runs on a query object <code>obj</code> and returns artifacts <code>artifacts</code>. If you inherit from this built-in query, make sure to use the same method signature inside your custom query:</p> <pre>function artifacts = run(obj,~) ... end</pre>
-----	--

Examples

Use Task Outputs as Task Inputs

Suppose that you want to pass the output of one task as the input to another task. You can use the built-in query `GetOutputsOfDependentTask` to find the outputs of the predecessor task and specify that query as an input query for the task.

For this example, open the Process Advisor example project.

```
processAdvisorExampleStart
```

The process model for the example specifies that the **Merge Test Results** task depends on the **Run Tests** task:

```
if includeTestsPerTestCaseTask && includeMergeTestResultsTask
    mergeTestTask.dependsOn(milTask, "WhenStatus", {'Pass', 'Fail'});
end
```

If you open the source code for the **Merge Test Results** task, you can see that the task uses the built-in query `padv.builtin.query.GetOutputsOfDependentTask` as an input query.

```
open padv.builtin.task.MergeTestResults
...
options.InputQueries = [padv.builtin.query.GetIterationArtifact,...
    padv.builtin.query.GetOutputsOfDependentTask(...
        Task="padv.builtin.task.RunTestsPerTestCase")];
options.InputDependencyQuery = padv.builtin.query.GetDependentArtifacts;
...
```

When you run the **Merge Test Results** task, the build system runs this input query, which passes the current model and the outputs of the **Run Tests** task as inputs to the **Merge Test Results** task.

For each input, the build system checks for other artifacts that the inputs depend on by running the `InputDependencyQuery`. For the **Merge Test Results** task, the input dependency query allows the build system to find related artifacts, such as data dictionaries, that the model uses. Those dependencies can impact if the task results are up-to-date or outdated.

Capabilities and Limitations

This table identifies functionality that is supported by the query.

Functionality	Supported?
Input query for task	Yes.
Iteration query for task	No.

See Also

`padv.builtin.task.AnalyzeModelCode` | `padv.builtin.task.MergeTestResults` |
`padv.builtin.task.RunCodeInspection` | `padv.builtin.query.GetDependentArtifacts` |
`padv.builtin.query.GetIterationArtifact`

Topics

“Create Custom Tasks”
“Overview of Process Model”

