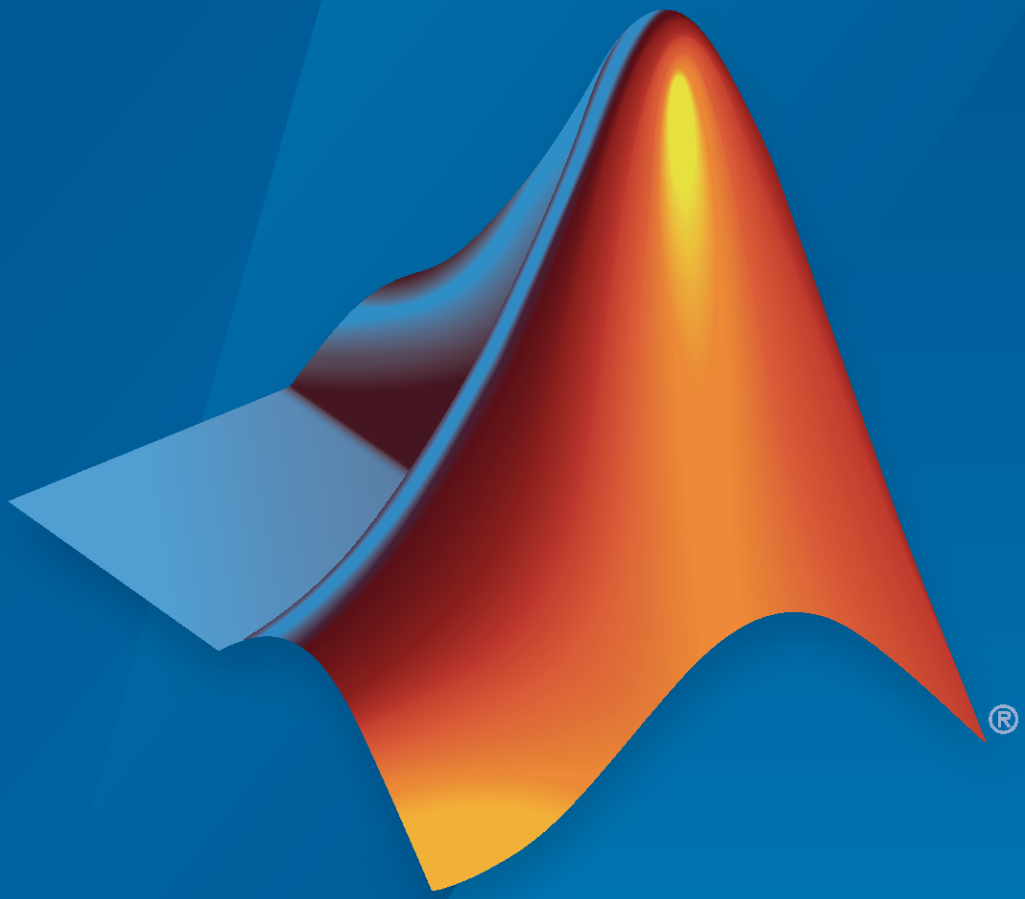
System Composer™

Reference



MATLAB® & SIMULINK®

R2020b

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

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

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

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

Phone: 508-647-7000

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

System Composer™ Reference

© COPYRIGHT 2019–2020 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

March 2019	Online only	New for Version 1.0 (Release 2019a)
September 2019	Online only	Revised for Version 1.1 (Release 2019b)
March 2020	Online only	Revised for Version 1.2 (Release 2020a)
September 2020	Online only	Revised for Version 1.3 (Release 2020b)

Contents

Functions

addChoice

Add variant choices to variant component

Syntax

```
compList = addChoice(variantComponent,choices)
compList = addChoice(variantComponent,choices,labels)
```

Description

`compList = addChoice(variantComponent,choices)` creates variant choices specified in `choices` in the specified variant component and returns their handles.

`compList = addChoice(variantComponent,choices,labels)` creates variant choices specified in `choices` with labels `labels` in the specified variant component and returns their handles.

Examples

Add Choices

Create a model, get the root architecture, create one variant component, and add two choices for the variant component.

```
model = systemcomposer.createModel('archModel');
arch = get(model,'Architecture');
variant = addVariantComponent(arch,'Component1');
compList = addChoice(variant,{'Choice1','Choice2'});
```

Input Arguments

`variantComponent` — Architecture component

variant component object

Architecture component where variant choices are added, specified as a `systemcomposer.arch.VariantComponent` object.

`choices` — Variant choice names

cell array of character vectors

Variant choice names, specified as a cell array of character vectors. The length of `choices` must be the same as `labels`.

Data Types: `char`

`labels` — Variant choice labels

cell array of character vectors

Variant choice labels, specified as a cell array of character vectors. The length of `labels` must be the same as `choices`.

Data Types: `char`

Output Arguments

`compList` — Created components

array of components

Created components, returned as an array of `systemcomposer.arch.Component` objects. This array is the same size as `choices` and `labels`.

See Also

`addVariantComponent` | `getActiveChoice` | `getChoices` | `makeVariant`

Topics

“Create Variants”

Introduced in R2019a

addComponent

Add components to architecture

Syntax

```
components = addComponent(architecture,compNames)
components = addComponent(architecture,compNames,stereotypes)
```

Description

`components = addComponent(architecture,compNames)` adds a set of components specified by the array of names.

`components = addComponent(architecture,compNames,stereotypes)` applies stereotypes specified in the `stereotypes` to the new components.

Examples

Create Model with Two Components

Create a model, get the root architecture, and create components.

```
model = systemcomposer.createModel('archModel');
arch = get(model,'Architecture');
names = {'Component1','Component2'};
comp = addComponent(arch,names);
```

Input Arguments

`architecture` — Parent architecture

architecture object

Parent architecture to which component is added, specified as a `systemcomposer.arch.Architecture` object.

`compNames` — Names of components

cell array of character vectors

Name of components, specified as a cell array of character vectors. The length of `compNames` must be the same as `stereotypes`.

Data Types: `char`

`stereotypes` — Stereotypes to apply to components

cell array of character vectors

Stereotypes to apply to components, specified as a cell array of character vectors. Each element is the fully qualified stereotype name for the corresponding component in the form `'<profile>.<stereotype>'`.

Data Types: `char`

Output Arguments

`components` — Created components
cell array of component objects

Created components, returned as a cell array of `systemcomposer.arch.Component` objects.

See Also

`addPort` | `connect`

Topics
“Components”

Introduced in R2019a

addComponent

Package: `systemcomposer.View`

Add component to view given path

Syntax

```
viewComp = addComponent(object,compPath)
```

Description

`viewComp = addComponent(object,compPath)` adds the component with the specified path.

`addComponent` is a method for the class `systemcomposer.view.ViewArchitecture`.

Input Arguments

`object` — View architecture
view architecture object

View architecture, specified as a `systemcomposer.view.ViewArchitecture` object.

`compPath` — Path to the component
character vector

Path to the component including the name of the top-model, specified as a character vector.

Data Types: `char`

Output Arguments

`viewComp` — View component
view component object

View component, returned as a `systemcomposer.view.ViewComponent` object.

See Also

`removeComponent` | `systemcomposer.view.BaseViewComponent` | `systemcomposer.view.ComponentOccurrence` | `systemcomposer.view.ViewArchitecture` | `systemcomposer.view.ViewComponent` | `systemcomposer.view.ViewElement`

Introduced in R2019b

addVariantComponent

Add variant components to architecture

Syntax

```
variantList = addVariantComponent(architecture,variantComponents)
variantList = addVariantComponent(architecture,variantComponents,'Position',
position)
```

Description

`variantList = addVariantComponent(architecture,variantComponents)` adds a set of components specified by the array of names.

`variantList = addVariantComponent(architecture,variantComponents,'Position',
position)` creates a variant component the architecture at a given position.

Examples

Create Variant with Two Components

Create model, get root architecture, and create a component with two variants.

```
model = systemcomposer.createModel('archModel');
arch = get(model,'Architecture');
names = {'Component1','Component2'}
variants = addVariantComponent(arch,names);
```

Input Arguments

`architecture` — Parent architecture

architecture object

Parent architecture to which component is added, specified as a `systemcomposer.arch.Architecture` object.

`variantComponents` — Names of variant components

cell array of character vectors

Names of variant components, specified as a cell array of character vectors.

Data Types: `char`

`position` — Vector that specifies location of top corner and bottom corner of component

1x4 array

Vector that specifies location of top corner and bottom corner of component, specified as a 1x4 array. The array denotes the top corner in terms of its x and y coordinates followed by the x and y coordinates of the bottom corner. When adding more than one variant component, a matrix of size [Nx4] may be specified where N is the number of variant components being added.

Data Types: `double`

Output Arguments

`variantList` — Handles to variant components

array of components

Handles to variant components, returned as an array of `systemcomposer.arch.VariantComponent` objects. This array is the same size as `variantComponents`.

See Also

`addChoice` | `addPort` | `connect` | `getActiveChoice` | `setActiveChoice`

Topics

“Components”

Introduced in R2019a

addElement

Add signal interface element

Syntax

```
element = addElement(interface,name)
element = addElement(interface,name,Name,Value)
```

Description

`element = addElement(interface,name)` adds an element to a signal interface with default properties.

`element = addElement(interface,name,Name,Value)` sets the properties of the element as specified in `Name,Value`.

Examples

Add an Interface and an Element

Add an interface `newsignal` to the interface dictionary of the model, and add an element `newelement` with type `double`.

```
arch = systemcomposer.createModel('newmodel',0);
interface = addInterface(arch.InterfaceDictionary,'newsignal');
element = addElement(interface,'newelement','Type','double')
```

```
element =
  SignalElement with properties:

      Interface: [1×1 systemcomposer.interface.SignalInterface]
           Name: 'newelement'
           Type: 'double'
     Dimensions: '1'
          Units: ''
     Complexity: 'real'
        Minimum: '[]'
        Maximum: '[]'
    Description: ''
           UUID: '2b47eaa6-191a-439a-ba2b-2bcc3209b912'
    ExternalUID: ''
```

Input Arguments

`interface` — New interface object

signal interface

New interface object, specified as a `systemcomposer.interface.SignalInterface` object.

`name` — Name of new element
character vector

Name of new element with a valid variable name, specified as a character vector.

Data Types: `char`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Type', 'double'`

`Type` — Data type of element
valid data type character vector

Data type of element, specified as the comma-separated pair consisting of `'Type'` and a valid data type character vector.

Data Types: `char`

`Dimensions` — Dimensions of element
positive integer array

Dimensions of element, specified as the comma-separated pair consisting of `'Dimensions'` and a positive integer array. Each element of the array is the size of the element in the corresponding direction. A scalar integer indicates a scalar or vector element and a row vector with two integers indicates a matrix element.

Data Types: `double`

`Complexity` — Complexity of element
`'real'` | `'complex'`

Complexity of element, specified as the comma-separated pair `'Complexity'` and `'real'` if the element is purely real, or `'complex'` if an imaginary part is allowed.

Data Types: `char`

Output Arguments

`element` — New interface element object
signal element

New interface element object, returned as a `systemcomposer.interface.SignalElement` object.

See Also

`getElement` | `getInterfaces` | `linkDictionary` | `systemcomposer.createDictionary` | `unlinkDictionary`

Topics
“Define Interfaces”

Introduced in R2019a

addPort

Add ports to architecture

Syntax

```
ports = addPort(architecture,portNames,portTypes)
ports = addPort(architecture,portNames,portTypes,stereotypes)
```

Description

`ports = addPort(architecture,portNames,portTypes)` adds a set of ports with specified names.

`ports = addPort(architecture,portNames,portTypes,stereotypes)` also applies stereotypes to a set of ports.

Examples

Add Ports to Architecture

Create a model, get the root architecture, add a component, and add ports.

```
model = systemcomposer.createModel('archModel');
rootArch = get(model,'Architecture');
newcomponent = addComponent(rootArch,'NewComponent');
newport = addPort(newcomponent.Architecture,'NewCompPort','in');
```

Input Arguments

`architecture` — Component architecture

architecture object

Component architecture, specified as a `systemcomposer.arch.Architecture` object. `addPort` adds ports to the architecture of a component. Use `<component>.Architecture` to access the architecture of a component.

`portNames` — Names of ports

cell array of character vectors

Names of ports, specified as a cell array of character vectors. If necessary, System Composer appends a number to the port name to ensure uniqueness. The size of `portNames`, `portTypes`, and `stereotypes` must be the same.

Data Types: `char`

`portTypes` — Port directions

cell array of character vectors

Port directions, specified as a cell array of character vectors. A port direction can be either `'in'` or `'out'`.

Data Types: `char`

`stereotypes` — Stereotypes to apply to components
cell array of stereotype objects

Stereotypes to apply to components, specified as a cell array of `systemcomposer.profile.Stereotype` objects. Each stereotype in the array must either be a stereotype that applies to all element types, or a port stereotype.

Output Arguments

`ports` — Created ports
cell array of ports

Created ports, returned as a cell array of `systemcomposer.arch.ComponentPort` or `systemcomposer.arch.ArchitecturePort` objects.

See Also

`addComponent` | `connect` | `destroy` | `systemcomposer.arch.BasePort`

Topics
“Ports”

Introduced in R2019a

addInterface

Create named interface in interface dictionary

Syntax

```
interface = addInterface(dictionary,name)
interface = addInterface(dictionary,name,'SimulinkBus',busObject)
```

Description

`interface = addInterface(dictionary,name)` adds a named interface to a specified interface dictionary.

`interface = addInterface(dictionary,name,'SimulinkBus',busObject)` constructs an interface that mirrors an existing Simulink® bus object.

Examples

Add an Interface

Add an interface `'newInterface'` to the specified data dictionary.

```
interface = addInterface(dictionary,'newInterface')
```

Add a Simulink Bus Mirrored Interface

Add a named interface that mirrors an existing Simulink bus object to a specified dictionary.

```
interface = addInterface(dictionary,'newInterface','SimulinkBus','myBus')
```

Input Arguments

`dictionary` — Data dictionary attached to architecture model

dictionary object

Data dictionary attached to architecture model, specified as a `systemcomposer.interface.Dictionary` object. This is the default data dictionary that defines local interfaces or an external data dictionary that carries interface definitions. If the model links to multiple data dictionaries, then `dictionary` must be the one that carries interface definitions. For information on how to create a dictionary, see `systemcomposer.createDictionary`.

`name` — Name of new interface

character vector

Name of new interface, specified as a character vector.

Data Types: `char`

`busObject` — Simulink bus object that new interface mirrors
bus object

Simulink bus object that new interface mirrors where the interface is already defined, specified as a Simulink bus object.

Output Arguments

`interface` — New interface object
signal interface object

New interface object, returned as a `systemcomposer.interface.SignalInterface` object.

See Also

`addElement` | `getInterface` | `getInterfaceNames` | `linkDictionary` | `systemcomposer.createDictionary`

Topics
“Define Interfaces”

Introduced in R2019a

addProperty

Define a custom property for a stereotype

Syntax

```
property = addProperty(stereotype,name)
property = addProperty(stereotype,name,Name,Value)
```

Description

`property = addProperty(stereotype,name)` returns a new `property` definition with `name` that is contained in `stereotype`.

`property = addProperty(stereotype,name,Name,Value)`returns a `property` definition that is configured with specified property values.

Examples

Add a Property

Add a component stereotype and add a `'VoltageRating'` property with value `5`.

```
profile = systemcomposer.profile.Profile.createProfile('myProfile');
stereotype = addStereotype(profile,'electricalComponent','AppliesTo','Component');
property = addProperty(stereotype,'VoltageRating','DefaultValue','5');
```

Input Arguments

`stereotype` — Stereotype to which property is added

stereotype object

Stereotype to which property is added, specified as a `systemcomposer.profile.Stereotype` object.

`name` — Name of property

character vector

Name of property unique within the stereotype, specified as a character vector.

Data Types: `char`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Type', 'double'`

`Type` — Property data type
`double` (default) | `single` | `int64` | `int32` | `int16` | `int8` | `uint64` | `uint32` | `uint8` | `boolean` | `string` | enumeration class name

Type of this property. One of valid data types or the name of a MATLAB class that defines an enumeration. For more information, see “Use Enumerated Data in Simulink Models”.

Example: `addProperty(stereotype,'Color','Type','BasicColors')`

Data Types: `char`

`Dimensions` — Dimensions of property
positive integer array

Dimensions of property, specified as a positive integer array. Empty implies no restriction.

Data Types: `double`

`Min` — Minimum value
numeric

Optional minimum value of this property. To set both `'Min'` and `'Max'` together, use the `setMinAndMax` method.

Example: `setMinAndMax(property, min, max)`

Data Types: `double`

`Max` — Maximum value
numeric

Optional maximum value of this property. To set both `'Min'` and `'Max'` together, use the `setMinAndMax` method.

Example: `setMinAndMax(property, min, max)`

Data Types: `double`

`Units` — Property units
character vector

Units of the property value, specified as a character vector. If specified, all values of this property on model elements are checked for consistency with these units according to Simulink unit checking rules. For more information, see “Unit Consistency Checking and Propagation”.

Data Types: `char`

`DefaultValue` — Default value
cell array of string value and string unit | string expression

Default value of this property, specified as a string expression or a cell array of string value and string unit.

Data Types: `double`

Output Arguments

`property` — Created property
property object

Created property, returned as a `systemcomposer.profile.Property` object.

See Also

`getProperty` | `setProperty`

Topics

“Define Profiles and Stereotypes”
“Set Tags and Properties for Analysis”

Introduced in R2019a

addStereotype

Add stereotype to profile

Syntax

```
stereotype = addStereotype(profile,stereotypeName)
stereotype = addStereotype(profile,stereotypeName,Name,Value)
```

Description

`stereotype = addStereotype(profile,stereotypeName)` adds a new stereotype with the specified name.

`stereotype = addStereotype(profile,stereotypeName,Name,Value)` specifies the properties of the stereotype.

Examples

Add Component Stereotype

Add a component stereotype to the profile.

```
addStereotype(profile,'electricalComponent','AppliesTo','Component')
```

Input Arguments

`profile` — Profile object

profile

Profile object, specified as a `systemcomposer.profile.Profile` object.

`stereotypeName` — Name of new stereotype

character vector

Name of new stereotype, specified as a character vector. The name of the stereotype must be unique within the profile.

Data Types: `char`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `addStereotype(profile,'electricalComponent','AppliesTo','Component')`

`Name, Value` — Stereotype properties and values

positive integer array

See `systemcomposer.profile.Stereotype` for stereotype properties and values.

Output Arguments

`stereotype` — Created stereotype
stereotype object

Created stereotype, returned as a `systemcomposer.profile.Stereotype` object.

See Also

`getStereotype`

Topics
“Create a Profile and Add Stereotypes”

Introduced in R2019a

AnyComponent

Package: `systemcomposer.query`

Create query to select all components in model

Syntax

```
query = AnyComponent()
```

Description

`query = AnyComponent()` creates a `query` object that the `find` method and the `createViewArchitecture` method use to select all components in the model.

Examples

Select All Components in Model

Import the package that contains all of the System Composer queries.

```
import systemcomposer.query.*;
```

Open the Simulink project file.

```
scKeylessEntrySystem
```

Open the model.

```
m = systemcomposer.openModel('KeylessEntryArchitecture');
```

Create a query to find all components and list the second.

```
constraint = AnyComponent();
components = find(m,constraint,'Recurse',true,'IncludeReferenceModels',true);
components(2)
```

```
ans =

  1×1 cell array

    {'KeylessEntryArchitecture/Door Lock//Unlock System/Door Lock Controller'}
```

Output Arguments

`query` — Query

query constraint object

Query, returned as a `systemcomposer.query.Constraint` object.

See Also

`createViewArchitecture` | `find` | `systemcomposer.query.Constraint`

Topics
“Creating Architectural Views Programmatically”

Introduced in R2019b

applyProfile

Apply profile to a model

Syntax

```
applyProfile(modelObject,profileFile)
```

Description

`applyProfile(modelObject,profileFile)` applies the profile to an architecture model and makes all of the constituent stereotypes available.

Input Arguments

`modelObject` — Architecture model object

model object

Architecture model object, specified as a `systemcomposer.arch.Model` object.

`profileFile` — Name of profile

character vector

Name of profile, specified as a character vector.

Example: `'SystemProfile'`

Data Types: `char`

See Also

`createProfile` | `removeProfile`

Topics

“Define Profiles and Stereotypes”

Introduced in R2019a

applyStereotype

Apply stereotype to architecture model element

Syntax

```
applyStereotype(element,stereotype)
```

Description

`applyStereotype(element,stereotype)` applies a stereotype to an architecture model element. The function adds the specified stereotype if it is not already applied to a model element. Stereotypes can be applied to architecture, component, port, and connector model elements.

Input Arguments

`element` — Model element

architecture object | component object | port object | connector object

Model element, specified as a `systemcomposer.arch.Architecture`, `systemcomposer.arch.Component`, `systemcomposer.arch.ComponentPort`, `systemcomposer.arch.ArchitecturePort`, or `systemcomposer.arch.Connector` object.

`stereotype` — Fully qualified name of stereotype

character vector

Fully qualified name of stereotype, specified as a character vector in the form `'<profile>.<stereotype>'`. The profile must already be applied to the model.

Data Types: `char`

See Also

`batchApplyStereotype` | `getStereotypes` | `removeStereotype`

Topics

“Use Stereotypes and Profiles”

Introduced in R2019a

batchApplyStereotype

Apply stereotype to all elements in specified architecture

Syntax

```
batchApplyStereotype(architecture,elementType,stereotype)
batchApplyStereotype(architecture,elementType,stereotype,'Recurse',flag)
```

Description

`batchApplyStereotype(architecture,elementType,stereotype)` applies the `stereotype` to all elements that match the `elementType` within the `architecture`.

`batchApplyStereotype(architecture,elementType,stereotype,'Recurse',flag)` applies the `stereotype` to all elements that match the `elementType` within the `architecture` and its sub-architectures.

Examples

Apply a Stereotype to All Connectors

Apply the `standardConn` stereotype in the `GeneralProfile` profile to all connectors within the architecture `arch`.

```
batchApplyStereotype(arch,'Connector','GeneralProfile.standardConn');
```

Input Arguments

`architecture` — Architecture model element
architecture object

Architecture model element, specified as a `systemcomposer.arch.Architecture` object. Parent architecture layer for all components to attach the stereotype.

`elementType` — Type of architecture element
`'Component'` | `'Port'` | `'Connector'` | `'Instance'`

Type of architecture element to apply the stereotype, specified as a character vector of `'Component'`, `'Port'`, `'Connector'`, or `'Instance'`. The stereotype must be applicable for this element type.

Data Types: `char`

`stereotype` — Stereotype to apply
character vector

Stereotype to apply, specified as a character vector in the form `'<profile>.<stereotype>'`. The stereotype must be applicable to components.

Data Types: `char`

`flag` — Apply stereotype recursively
`false` or 0 (default) | `true` or 1

Apply stereotype recursively, specified as a `logical` or numeric value. If `flag` is 1 (`true`), the stereotype is applied to the elements in the architecture and its sub-architectures.

Data Types: `logical`

See Also

`applyStereotype` | `getStereotypes` | `removeStereotype`

Topics
“Use Stereotypes and Profiles”

Introduced in R2019a

close

Close profile

Syntax

```
close(profile,force)
```

Description

`close(profile,force)` closes the profile. If there are unsaved changes, you will receive an error unless the argument `force` is true.

Tip Use `closeAll` to close all loaded profiles.

Input Arguments

`profile` — Profile

profile object

Profile, specified as a `systemcomposer.profile.Profile` object.

`force` — Force the close

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

Force close the profile, specified as a logical or numeric value 1 (`true`) or 0 (`false`).

Data Types: `logical`

See Also

`find` | `load` | `open` | `save` | `systemcomposer.profile.Profile`

Topics

“Define Profiles and Stereotypes”

Introduced in R2019a

close

Close allocation set

Syntax

```
close(allocSet,force)
```

Description

`close(allocSet,force)` closes the allocation set. If there are unsaved changes, you will receive an error unless the argument `force` is true.

Tip Use `closeAll` to close all loaded allocation sets.

Input Arguments

`allocSet` — Allocation set

allocation set object

Allocation set, specified as a `systemcomposer.allocation.AllocationSet` object.

`force` — Force the close

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

Force close the allocation set, specified as a logical or numeric value 1 (`true`) or 0 (`false`).

Data Types: `logical`

See Also

`createScenario` | `deleteScenario` | `getScenario` | `load`

Topics

“Create and Manage Allocations”

Introduced in R2020b

close

Package: `systemcomposer.arch`

Close System Composer model

Syntax

```
close(objModel)
```

Description

`close(objModel)` closes the specified model in System Composer.

Examples

Create, Open, and Close a Model

```
Model = systemcomposer.createModel('modelName');
open(Model)
close(Model)
```

Input Arguments

`objModel` — Model to close in editor

model object

Model to close in the System Composer editor, specified as a `systemcomposer.arch.Model` object.

See Also

`createModel` | `loadModel` | `save`

Topics

“Create an Architecture Model”

Introduced in R2019a

systemcomposer.allocation.Allocation

Allocation between source and target element

Description

The `systemcomposer.allocation.Allocation` defines the allocation between the source element and the target element.

Creation

```
% Create two allocations between four elements in
% the default scenario, Scenario 1.
defaultScenario = allocSet.getScenario('Scenario 1');
defaultScenario.allocate(sourceElement1,sourceElement2);
defaultScenario.allocate(sourceElement3,sourceElement4);
```

Properties

`Source` — Source element
element object

Source element, returned as a `systemcomposer.arch.Element` object.

`Target` — Target element
element object

Target element, returned as a `systemcomposer.arch.Element` object.

`Scenario` — Allocation scenario
allocation scenario object

Allocation scenario, returned as a `systemcomposer.allocation.AllocationScenario` object.

See Also

`allocate` | `getAllocatedFrom` | `getAllocatedTo` | `getAllocation` | `getScenario`

Topics
“Create and Manage Allocations”

Introduced in R2020b

allocate

Create new allocation

Syntax

```
allocation = allocate(sourceElement,targetElement)
```

Description

`allocation = allocate(sourceElement,targetElement)` creates a new allocation between the source element and the target element.

Input Arguments

`sourceElement` — Source element for allocation

element object | character vector

Source element for allocation, specified as a `systemcomposer.arch.Element` object or the name of an element as a character vector.

`targetElement` — Target element for allocation

element object | character vector

Target element for allocation, specified as a `systemcomposer.arch.Element` object or the name of an element as a character vector.

Output Arguments

`allocation` — Allocation between source and target element

allocation object

Allocation between source and target element, returned as a `systemcomposer.allocation.Allocation` object.

See Also

`deallocate` | `getAllocation`

Topics

“Create and Manage Allocations”

Introduced in R2020b

systemcomposer.allocation.AllocationScenario

Manage allocation scenario

Description

The `systemcomposer.allocation.AllocationScenario` class defines a collection of allocations between elements in the source model to elements in the target model.

Creation

```
scenario = createScenario(myAllocationSet);
```

Properties

`Name` — Name of allocation scenario
character vector

Name of allocation scenario, returned as a character vector.

Data Types: `char`

`Allocations` — Allocations in the scenario
cell array of allocation objects

Allocations in the scenario, returned as a cell array of `systemcomposer.allocation.Allocation` objects.

`AllocationSet` — Allocation set that this scenario belongs to
allocation set object

Allocation set that this scenario belongs to, returned as an `systemcomposer.allocation.AllocationSet` object.

`Description` — Description of allocation set
character vector

Description of allocation set, returned as a character vector.

Data Types: `char`

Object Functions

allocate	Create new allocation
destroy	Delete allocation scenario
deallocate	Delete allocation between source and target element
getAllocation	Get allocation between source and target elements
getAllocatedFrom	Get allocation target
getAllocatedTo	Get allocation source

See Also

`createScenario`

Topics

“Create and Manage Allocations”

Introduced in R2020b

deallocate

Delete allocation between source and target element

Syntax

```
deallocate(sourceElement,targetElement)
```

Description

`deallocate(sourceElement,targetElement)` deletes allocation, if one exists, between a source and a target element.

Input Arguments

`sourceElement` — Source element to delete allocation

element object | character vector

Source element to delete allocation, specified as a `systemcomposer.arch.Element` object or the name of an element as a character vector.

`targetElement` — Target element to delete allocation

element object | character vector

Target element to delete allocation, specified as a `systemcomposer.arch.Element` object or the name of an element as a character vector.

See Also

`allocate` | `getAllocatedFrom` | `getAllocatedTo` | `getAllocation`

Topics

“Create and Manage Allocations”

Introduced in R2020b

destroy

Delete allocation scenario

Syntax

`destroy()`

Description

`destroy()` deletes the existing allocation scenario in the allocation set.

See Also

`createScenario` | `deleteScenario` | `getScenario`

Topics

“Create and Manage Allocations”

Introduced in R2020b

getAllocation

Get allocation between source and target elements

Syntax

```
allocation = getAllocation(sourceElement,targetElement)
```

Description

`allocation = getAllocation(sourceElement,targetElement)` get the allocation, if one exists, between the source and target element.

Input Arguments

`sourceElement` — Source element for allocation

element object | character vector

Source element for allocation, specified as a `systemcomposer.arch.Element` object or the name of the element as a character vector.

`targetElement` — Target element for allocation

element object | character vector

Target element for allocation, specified as a `systemcomposer.arch.Element` object or the name of the element as a character vector.

Output Arguments

`allocation` — Allocation between source and target element

allocation object

Allocation between source and target element, returned as a `systemcomposer.allocation.Allocation` object.

See Also

`allocate` | `deallocate` | `getAllocatedFrom` | `getAllocatedTo`

Topics

“Create and Manage Allocations”

Introduced in R2020b

getAllocatedFrom

Get allocation target

Syntax

```
targetElements = getAllocatedFrom(element)
```

Description

`targetElements = getAllocatedFrom(element)` gets all the elements that are allocated from the specified source element.

Input Arguments

`element` — Source element

element object | character vector

Source element, specified as a `systemcomposer.arch.Element` object or an element name as a character vector.

Output Arguments

`targetElements` — Target elements

array of element objects

Target elements that are allocated from the specified element, returned as an array of `systemcomposer.arch.Element` objects.

See Also

`allocate` | `deallocate` | `getAllocatedTo`

Topics

“Create and Manage Allocations”

Introduced in R2020b

getAllocatedTo

Get allocation source

Syntax

```
sourceElements = getAllocatedTo(element)
```

Description

`sourceElements = getAllocatedTo(element)` gets all the source elements allocated to a specified element.

Input Arguments

`element` — Target element
element object | character vector

The element for which you target to find the source elements, specified as a `systemcomposer.arch.Element` object or a name of the element as a character vector.

Output Arguments

`sourceElements` — Source elements
array of element objects

Source elements that are allocated to the specified element, specified as an array of `systemcomposer.arch.Element` objects.

See Also

`allocate` | `deallocate` | `getAllocatedFrom`

Topics
“Create and Manage Allocations”

Introduced in R2020b

closeAll

Close all loaded allocation sets

Syntax

```
systemcomposer.allocation.AllocationSet.closeAll()
```

Description

`systemcomposer.allocation.AllocationSet.closeAll()` closes all allocation sets without saving.

Tip Use `close` to close one allocation set.

See Also

`createScenario` | `deleteScenario` | `getScenario` | `load`

Topics

“Create and Manage Allocations”

Introduced in R2020b

closeAll

Close all open profiles

Syntax

```
systemcomposer.profile.Profile.closeAll()
```

Description

`systemcomposer.profile.Profile.closeAll()` force closes all open profiles.

Tip Use `close` to close one open profile.

See Also

`find` | `load` | `open` | `save` | `systemcomposer.profile.Profile`

Topics

"Define Profiles and Stereotypes"

Introduced in R2019a

connect

Create architecture model connections

Syntax

```
connectors = connect(srcComponent,destComponent)
connectors = connect(srcPort,destPort)
connectors = connect(architecture,[srcComponent,srcComponent,...],[
destComponent,destComponent,...])
connectors = connect(architecture,[],destComponent)
connectors = connect(architecture,srcComponent,[])
connectors = connect( ___ ,Name,Value)
```

Description

`connectors = connect(srcComponent,destComponent)` connects the unconnected output ports of `srcComponent` to the unconnected input ports of `destComponent` based on matching port names, and returns a handle to the connector.

`connectors = connect(srcPort,destPort)` connects a source port and a destination port.

`connectors = connect(architecture,[srcComponent,srcComponent,...],[ destComponent,destComponent,...])` connects arrays of components in the architecture.

`connectors = connect(architecture,[],destComponent)` connects a parent architecture input port to a destination child component.

`connectors = connect(architecture,srcComponent,[])` connects a source child component to a parent architecture output port.

`connectors = connect( ___ ,Name,Value)` specifies options using one or more name-value pair arguments in addition to the input arguments in previous syntaxes.

Examples

Connect System Composer Components

Create and connect two components.

Create top level architecture model.

```
modelName = 'archModel';
arch = systemcomposer.createModel(modelName);
rootArch = get(arch,'Architecture');
```

Create two new components.

```
names = {'Component1','Component2'};
newComponents = addComponent(rootArch,names);
```

Add ports to components.

```
outPort1 = addPort(newComponents(1).Architecture,'testSig','out');
inPort1 = addPort(newComponents(2).Architecture,'testSig','in');
```

Connect components.

```
conns = connect(newComponents(1),newComponents(2));
```

View model.

```
systemcomposer.openModel(modelName);
```

Improve layout.

```
Simulink.BlockDiagram.arrangeSystem(modelName)
```

Connect System Composer Ports

Create and connect two ports.

Create top level architecture model.

```
modelName = 'archModel';
arch = systemcomposer.createModel(modelName);
rootArch = get(arch,'Architecture');
```

Create two new components.

```
names = {'Component1','Component2'};
newcomponents = addComponent(rootArch,names);
```

Add ports to components.

```
outPort1 = addPort(newComponents(1).Architecture,'testSig','out');
inPort1 = addPort(newComponents(2).Architecture,'testSig','in');
```

Extract component ports.

```
srcPort = getPort(newComponents(1),'testSig');
destPort = getPort(newComponents(2),'testSig');
```

Connect ports.

```
conns = connect(srcPort,destPort);
```

View model.

```
systemcomposer.openModel(modelName);
```

Improve layout.

```
Simulink.BlockDiagram.arrangeSystem(modelName)
```

Connect by Selecting Destination Element

Create and connect destination architecture port interface element to component.

Create top level architecture model.

```
modelName = 'archModel';
arch = systemcomposer.createModel(modelName);
rootArch = get(arch,'Architecture');
```

Create new component.

```
newComponent = addComponent(rootArch,'Component1');
```

Add destination architecture ports to component and architecture.

```
outPortComp = addPort(newComponent.Architecture,'testSig','out');
outPortArch = addPort(rootArch,'testSig','out');
```

Extract corresponding port objects.

```
compSrcPort = getPort(newComponent,'testSig');
archDestPort = getPort(rootArch,'testSig');
```

Add interface, interface element, and associate interface with architecture port.

```
interface = arch.InterfaceDictionary.addInterface('interface');
interface.addElement('x');
archDestPort.setInterface(interface);
```

Select element on architecture port and establish connection.

```
conns = connect(compSrcPort,archDestPort,'DestinationElement','x');
```

View model.

```
systemcomposer.openModel(modelName);
```

Improve layout.

```
Simulink.BlockDiagram.arrangeSystem(modelName)
```

Input Arguments

`architecture` — Interface and underlying structural definition of model or component
architecture object

Interface and underlying structural definition of model or component, specified as a `systemcomposer.arch.Architecture` object.

`srcComponent` — Source component
component object

Source component, specified as a `systemcomposer.arch.Component` object.

`destComponent` — Destination component
component object

Destination component, specified as a `systemcomposer.arch.Component` object.

`srcPort` — Source port
port object

Source port to connect, specified as a `systemcomposer.arch.ComponentPort` or `systemcomposer.arch.ArchitecturePort` object.

`destPort` — Destination port
port object

Destination port to connect, specified as a `systemcomposer.arch.ComponentPort` or `systemcomposer.arch.ArchitecturePort` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `connect(archPort,compPort,'SourceElement','a')`

`Stereotype` — Option to apply stereotype to connector
stereotype object

Option to apply stereotype to connector, specified as the comma-separated pair consisting of `'Stereotype'` and a `systemcomposer.profile.Stereotype` object.

`Rule` — Option to specify rule for connections
`'name'` (default) | `'interface'`

Option to specify rule for connections, specified as the comma-separated pair consisting of `'Rule'` and `'name'` based on name of ports or `'interface'` based on interface name on ports.

`MultipleOutputConnectors` — Option to allow multiple destination components
`false` or 0 (default) | `true` or 1

Option for the same source component to connect to multiple destination components, specified as the comma-separated pair consisting of `'MultipleOutputConnectors'` and a numeric or logical 1 (`true`) or 0 (`false`).

`SourceElement` — Option to select source element for connection
character vector

Option to select source element for connection, specified as the comma-separated pair consisting of `'SourceElement'` and a character vector of the name of the signal element.

Data Types: `char`

`DestinationElement` — Option to select destination element for connection
character vector

Option to select destination element for connection, specified as the comma-separated pair consisting of `'DestinationElement'` and a character vector of the name of the signal element.

Data Types: `char`

Output Arguments

`connectors` — Created connections
array of connections

Created connections, returned as an array of `systemcomposer.arch.Connector` objects.

See Also

`addComponent` | `addElement` | `addInterface` | `addPort` | `createModel` | `getDestinationElement` | `getPort` | `getSourceElement` | `openModel` | `setInterface`

Topics
“Create an Architecture Model”

Introduced in R2019a

createAllocationSet

Create a new allocation set

Syntax

```
allocSet = systemcomposer.allocation.createAllocationSet(name, sourceModel,
targetModel)
```

Description

`allocSet = systemcomposer.allocation.createAllocationSet(name, sourceModel, targetModel)` creates a new allocation set with the given name in which the source and target models are provided.

Examples

Create an allocation set and open in Allocation Editor

```
% Create the allocation set with name MyNewAllocation.
systemcomposer.allocation.createAllocationSet('MyNewAllocation',...
     'Source_Model_Allocation','Target_Model_Allocation');

% Open the allocation editor
systemcomposer.allocation.editor()
```

Input Arguments

`name` — Name of allocation set

model object | character vector

Name of allocation set, specified as a `systemcomposer.arch.Model` object or the name of a model as a character vector.

`sourceModel` — Source model for allocation

model object | character vector

Source model for allocation, specified as a `systemcomposer.arch.Model` object or the name of a model as a character vector.

`targetModel` — Target model for allocation

model object | character vector

Target model for allocation, specified as a `systemcomposer.arch.Model` object or the name of a model as a character vector.

Output Arguments

`allocSet` — Allocation set

allocation set object

Allocation set created, returned as a `systemcomposer.allocation.AllocationSet` object.

See Also

`closeAll` | `load` | `open`

Topics

“Create and Manage Allocations”

Introduced in R2020b

createAnonymousInterface

Create and set anonymous interface for port

Syntax

```
interface = createAnonymousInterface(port)
```

Description

`interface = createAnonymousInterface(port)` creates and sets an anonymous interface for a port.

Input Arguments

`port` — Port

port object

Port, specified as a `systemcomposer.arch.ArchitecturePort` or `systemcomposer.arch.ComponentPort` object.

Output Arguments

`interface` — Signal interface

signal interface object

Signal interface, returned as a `systemcomposer.interface.SignalInterface` object.

See Also

`systemcomposer.arch.ArchitecturePort` | `systemcomposer.arch.ComponentPort`

Topics

“Define Interfaces”

Introduced in R2019a

createDictionary

Create data dictionary

Syntax

```
dict_id = systemcomposer.createDictionary(dictionaryName)
```

Description

`dict_id = systemcomposer.createDictionary(dictionaryName)` creates a new Simulink data dictionary to hold interfaces and returns a handle to the `systemcomposer.interface.Dictionary` object.

Examples

Create a New Dictionary

```
dict_id = systemcomposer.createDictionary('new_dictionary.sldd')
```

Input Arguments

`dictionaryName` — Name of new data dictionary

character vector

Name of new data dictionary, specified as a character vector. The name must include the `.sldd` extension.

Example: `'new_dictionary.sldd'`

Data Types: `char`

Output Arguments

`dict_id` — Handle to the dictionary

dictionary object

Handle to the dictionary, returned as a `systemcomposer.interface.Dictionary` object.

See Also

`linkDictionary` | `systemcomposer.openDictionary` | `unlinkDictionary`

Topics

“Save, Link, and Delete Interfaces”

Introduced in R2019a

createModel

Create a System Composer model

Syntax

```
objModel = systemcomposer.createModel(modelName)
```

Description

`objModel = systemcomposer.createModel(modelName)` creates a System Composer model with name `modelName` and returns its handle.

`createModel` is the constructor method for the class `systemcomposer.arch.Model`.

Examples

```
model = systemcomposer.createModel('model_name')
```

```
model =

  model with properties:

                  Name: 'model_name'
          Architecture: [1×1 systemcomposer.arch.Architecture]
        SimulinkHandle: 2.0005
                 Views: [0×0 systemcomposer.view.ViewArchitecture]
              Profiles: [0×0 systemcomposer.profile.Profile]
    InterfaceDictionary: [1×1 systemcomposer.interface.Dictionary]
```

Input Arguments

`modelName` — Name of new model

character vector

Name of new model, specified as a character vector.

Data Types: `char`

Output Arguments

`objModel` — Model handle

model object

Model handle, returned as a `systemcomposer.arch.Model` object.

See Also

`loadModel` | `open` | `save`

Topics

“Compose Architecture Visually”

Introduced in R2019a

systemcomposer.profile.Profile.createProfile

Create profile

Syntax

```
profile = systemcomposer.profile.Profile.createProfile(profileName,dirPath)
profile = systemcomposer.profile.Profile.createProfile(profileName)
```

Description

`profile = systemcomposer.profile.Profile.createProfile(profileName,dirPath)` creates a new profile object of type `systemcomposer.profile.Profile` to setup a set of stereotypes. The `dirPath` argument specifies the directory in which the profile is to be created.

`profile = systemcomposer.profile.Profile.createProfile(profileName)` creates a new profile with name `profileName`.

Example

```
profile = systemcomposer.profile.Profile.createProfile('new_profile')
```

Input Arguments

`profileName` — Name of new profile

character vector

Name of new profile, specified as a character vector.

Example: `'new_profile'`

Data Types: `char`

`dirPath` — Directory path

character vector

Directory path where the profile will be saved, specified as a character vector.

Example: `'C:\Temp\MATLAB'`

Data Types: `char`

Output Arguments

`profile` — Profile handle

profile object

Profile handle, returned as a `systemcomposer.profile.Profile` object.

See Also

`applyProfile` | `find` | `load` | `loadProfile` | `open` | `removeProfile` | `save`

Topics
“Create a Profile and Add Stereotypes”

Introduced in R2019a

createScenario

Create new empty allocation scenario

Syntax

```
scenario = createScenario(name)
```

Description

`scenario = createScenario(name)` creates a new empty allocation scenario in the allocation set with the given name.

Input Arguments

`name` — Name of allocation set

allocation set object | character vector

Name of allocation set, specified as a `systemcomposer.allocation.AllocationSet` object or the name as a character vector.

Output Arguments

`scenario` — New empty allocation scenario

allocation scenario object

New empty allocation scenario, returned as a `systemcomposer.allocation.AllocationScenario` object.

See Also

`deleteScenario` | `getScenario`

Topics

“Create and Manage Allocations”

Introduced in R2020b

createSimulinkBehavior

Create Simulink model and link component to it

Syntax

```
createSimulinkBehavior(component,modelName)
```

Description

`createSimulinkBehavior(component,modelName)` creates a new Simulink model with the same interface as the component and links the component to the new model. This method works only if the component has no children.

Examples

Create a Simulink Model and Link

Create a Simulink behavior model for the component `robotComp` in `Robot.slx` and link the component to the model.

```
createSimulinkBehavior(robotComp,'Robot');
```

Input Arguments

`component` — Architecture component
component object

Architecture component with no children, specified as a `systemcomposer.arch.Component` object.

`modelName` — Model name
character vector

Model name of the Simulink model created by this function, specified as a character vector.

Example: `'Robot'`

Data Types: `char`

See Also

`inlineComponent` | `linkToModel` | `saveAsModel`

Topics
“Implement Components in Simulink”

Introduced in R2019a

createViewArchitecture

Package: `systemcomposer.arch`

Create view

Syntax

```
view = createViewArchitecture(obj,name,Name,Value)
view = createViewArchitecture(obj,name,constraint,Name,Value)
view = createViewArchitecture(obj,name,constraint,groupBy,Name,Value)
```

Description

`view = createViewArchitecture(obj,name,Name,Value)` creates an empty view with the given name.

`view = createViewArchitecture(obj,name,constraint,Name,Value)` creates a view with the given name where the contents are populated by finding all components in the model that satisfy the provided query.

`view = createViewArchitecture(obj,name,constraint,groupBy,Name,Value)` creates a view with the given name where the contents are populated by finding all components in the model that satisfy the provided query. The selected components are then grouped by the fully qualified property name.

Examples

Create a View Based on a Query and Review Status

```
scKeylessEntrySystem;
m = systemcomposer.openModel('KeylessEntryArchitecture');

import systemcomposer.query.*;
myQuery = HasStereotype(IsStereotypeDerivedFrom('AutoProfile.SoftwareComponent'));

view = m.createViewArchitecture('Software Review Status',myQuery,...
'AutoProfile.BaseComponent.ReviewStatus','Color','red');

m.openViews;
```

Input Arguments

`obj` — Model

architecture model object

Model to use to create a view, specified as a `systemcomposer.arch.Model` object.

`name` — Name of view

character vector

Name of the view, specified as a character vector.

Data Types: `char`

`constraint` — Query

query constraint object

Query, specified as a `systemcomposer.query.Constraint` object representing specific conditions. A constraint can contain a sub-constraint that can be joined together with another constraint using `AND` or `OR`. A constraint can also be negated using `NOT`.

Query Objects and Conditions for Constraints

Query Object	Condition
`Property`	A non-evaluated value for the given property or stereotype property.
`PropertyValue`	An evaluated property value from a System Composer object or a stereotype property.
`HasPort`	A component has a port that satisfies the given sub-constraint.
`HasInterface`	A port has an interface that satisfies the given sub-constraint.
`HasInterfaceElement`	An interface has an interface element that satisfies the given sub-constraint.
`HasStereotype`	An architecture element has a stereotype that satisfies the given sub-constraint.
`IsInRange`	A property value is within the given range.
`AnyComponent`	An element is a component and not a port or connector.
`IsStereotypeDerivedFrom`	A stereotype is derived from the given stereotype.

`groupBy` — User-defined property

enumeration

User-defined property, specified as an enumeration by which to group components.

Data Types: `enum`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `createViewArchitecture(model,'Software Review Status',myQuery,'AutoProfile.BaseComponent.ReviewStatus','Color','red','IncludeReferenceModels',true)`

`IncludeReferenceModels` — Option to search for reference architectures

`false` (default) | `true`

Option to search for reference architectures, or to not include referenced architectures, specified as the comma-separated pair consisting of `'IncludeReferenceModels'` and a logical `false` to not include referenced architectures and `true` to search for referenced architectures.

Example: `'IncludeReferenceModels',true`

Data Types: `logical`

`Color` — Color of view
character array

Color of view, specified as the comma-separated pair consisting of `'Color'` and a character array that contains the name of the color or an RGB hexadecimal value.

Example: `'Color','blue'`

Example: `'Color,'#FF00FF'`

Data Types: `char`

Output Arguments

`view` — Model architecture view
view architecture object

Model architecture view created based on the specified query and properties, specified as a `systemcomposer.view.ViewArchitecture` object.

See Also

`find` | `systemcomposer.query.Constraint`

Topics
“Build an Architecture Model from Command Line”
“Creating Architectural Views Programmatically”

Introduced in R2019b

createViewComponent

Create new view component

Syntax

```
viewComp = createViewComponent(object,name)
```

Description

`viewComp = createViewComponent(object,name)` creates a new view component with the provided name.

`createViewComponent` is a method for the class `systemcomposer.view.ViewArchitecture`.

Examples

Create View Component

Create view component with context view.

```
scKeylessEntrySystem
zcModel = systemcomposer.loadModel('KeylessEntryArchitecture');
fobSupplierView = zcModel.createViewArchitecture("FOB Locator System Supplier Breakdown",...
    "Color","lightblue");
supplierD = fobSupplierView.createViewComponent("Supplier D");
```

Input Arguments

`object` — View architecture

view architecture object

View architecture, specified as a `systemcomposer.view.ViewArchitecture` object.

`name` — Name of component

character vector

Name of component, specified as a character vector.

Data Types: `char`

Output Arguments

`viewComp` — View component

view component object

View component, returned as a `systemcomposer.view.ViewComponent` object.

See Also

systemcomposer.view.BaseViewComponent | systemcomposer.view.ComponentOccurrence | systemcomposer.view.ViewArchitecture | systemcomposer.view.ViewComponent | systemcomposer.view.ViewElement

Introduced in R2019b

deleteInstance

Delete architecture instance

Syntax

```
deleteInstance(architectureInstance)
```

Description

`deleteInstance(architectureInstance)` deletes an existing instance.

Input Arguments

`architectureInstance` — Architecture instance
instance object

Architecture instance to be deleted, specified as a `systemcomposer.analysis.ArchitectureInstance` object.

See Also

`instantiate` | `loadInstance` | `saveInstance` | `systemcomposer.analysis.Instance` | `updateInstance`

Topics
“Write Analysis Function”

Introduced in R2019a

deleteScenario

Delete allocation scenario

Syntax

```
deleteScenario(name)
```

Description

`deleteScenario(name)` deletes the allocation scenario in a set with a given name.

Input Arguments

`name` — Name of scenario to be deleted

character vector

Name of scenario to be deleted, specified as a character vector.

Data Types: `char`

See Also

`createScenario` | `getScenario`

Topics

“Create and Manage Allocations”

Introduced in R2020b

destroy

Remove and destroy model element

Syntax

```
destroy(element)
```

Description

`destroy(element)` removes and destroys the model element.

Examples

Destroy a Component

Create a component and then remove it from the model.

```
newcomponent = addComponent(rootArch,'NewComponent');
destroy(newcomponent)
```

Input Arguments

`element` — Architecture model element

architecture element object | interface element object | signal element object | property object

Architecture model element, specified as a `systemcomposer.arch.Element`, `systemcomposer.interface.SignalElement`, `systemcomposer.interface.SignalInterface`, and `systemcomposer.profile.Property` object.

See Also

`removeElement` | `removeProfile` | `removeProperty`

Introduced in R2019a

exportModel

Export model information as MATLAB tables

Syntax

```
[exportedSet] = systemcomposer.exportModel(modelName)
```

Description

`[exportedSet] = systemcomposer.exportModel(modelName)` exports model information for components, ports, connectors, port interfaces, and requirements to be imported into MATLAB® tables. The exported tables have prescribed formats to specify model element relationships, stereotypes, and properties.

Examples

Export System Composer Model

To export a model, pass the model name as an argument to the `exportModel` function. The function returns a structure containing five tables: `components`, `ports`, `connections`, `portInterfaces`, and `requirementLinks`.

```
exportedSet = systemcomposer.exportModel('exMobileRobot')

exportedSet =

  struct with fields:

          components: [3×4 table]
               ports: [3×5 table]
         connections: [1×4 table]
      portInterfaces: [3×9 table]
    requirementLinks: [4×15 table]
```

Input Arguments

`modelName` — Name of model to be exported

character vector

Name of model to be exported, specified as a character vector.

Example: `'exMobileRobot'`

Data Types: `char`

Output Arguments

`exportedSet` — Model tables

structure

Model tables, returned as a structure containing tables `components`, `ports`, `connections`, `portInterfaces`, and `requirementLinks`.

Data Types: `struct`

See Also

`systemcomposer.importModel`

Topics

“Import and Export Architecture Models”

Introduced in R2019a

systemcomposer.extractArchitectureFromSimulink

Extract architecture from Simulink model

Syntax

```
systemcomposer.extractArchitectureFromSimulink(model,architectureModelName)
```

Description

`systemcomposer.extractArchitectureFromSimulink(model,architectureModelName)` exports the Simulink model `model` to an architecture model `architectureModelName` and saves it in the current directory.

Examples

Extract Architecture from Example Model

Extract architecture from a model with subsystem and variant architecture.

```
ex_modeling_variants;
systemcomposer.extractArchitectureFromSimulink('ex_modeling_variants','archModel')
```

Input Arguments

`model` — Simulink model

character vector

Simulink model from which to extract the architecture, specified as a character vector. The model must be on the path.

Example: `'ex_modeling_variants'`

Data Types: `char`

`architectureModelName` — Architecture model name

character vector

Architecture model name, specified as a character vector. This model is saved in the current directory.

Data Types: `char`

See Also

`inlineComponent` | `linkToModel` | `saveAsModel`

Topics

“Extract Architecture from Simulink Model”

Introduced in R2019a

systemcomposer.allocation.editor

Open allocation editor

Syntax

```
systemcomposer.allocation.editor()
```

Description

`systemcomposer.allocation.editor()` opens the allocation editor.

See Also

`createAllocationSet` | `systemcomposer.allocation.AllocationSet`

Topics

“Create and Manage Allocations”

Introduced in R2020b

find

Find loaded allocation set

Syntax

```
allocSet = systemcomposer.allocation.AllocationSet.find(name)
```

Description

`allocSet = systemcomposer.allocation.AllocationSet.find(name)` finds a loaded allocation set in the global name space with the given name.

Input Arguments

`name` — Name of scenario to be found

character vector

Name of scenario to be found, specified as a character vector.

Data Types: `char`

Output Arguments

`allocSet` — Allocation set

allocation set object

Allocation set, returned as a `systemcomposer.allocation.AllocationSet` object.

See Also

`closeAll` | `load` | `save`

Topics

“Create and Manage Allocations”

Introduced in R2020b

find

Package: `systemcomposer.arch`

Find architecture elements using query

Syntax

```
[paths] = find(object,constraint,Name,Value)
[paths, elements] = find( ___ )
[elements] = find( ___ )
[paths] = find(object,constraint,rootArch,Name,Value)
```

Description

`[paths] = find(object,constraint,Name,Value)` finds all element paths starting from the root architecture of the model that satisfy the `constraint` query, with additional options specified by one or more name-value pair arguments.

`[paths, elements] = find( ___ )` returns the component `elements` and their `paths` that satisfy the `constraint` query. If `rootArch` is not provided, then the function finds model elements in the root architecture of the model. The output argument `paths` contains a fully qualified named path for each component in `elements` from the given root architecture.

`[elements] = find( ___ )` finds all component, port, or connector `elements` that satisfy the `constraint` query, with additional options specified by one or more name-value pair arguments, which must include `'Port'` or `'Connector'` for `'ElementType'`.

`[paths] = find(object,constraint,rootArch,Name,Value)` finds all element paths starting from the specified root architecture that satisfy the `constraint` query, with additional options specified by one or more name-value pair arguments.

Examples

Find Model Element Paths that Satisfy Query

Import a model and run a query to select architecture elements that have a stereotype based on the specified sub-constraint.

```
import systemcomposer.query.*;
scKeylessEntrySystem
modelObj = systemcomposer.openModel('KeylessEntryArchitecture');
find(modelObj,HasStereotype(IsStereotypeDerivedFrom('AutoProfile.BaseComponent')),...
 'Recurse',true,'IncludeReferenceModels',true)
```

Create a query to find components that contain the letter `'c'` in their `'Name'` property.

```
constraint = contains(systemcomposer.query.Property('Name'),'c');
find(modelObj,constraint,'Recurse',true,'IncludeReferenceModels',true)
```

Find Elements in an Architecture Model

This example shows how to find elements in an architecture model based on a query.

Create Model

Create an architecture model with two components.

```
m = systemcomposer.createModel('exModel');
comps = m.Architecture.addComponent({'c1','c2'});
```

Create Profile and Stereotypes

Create a profile and stereotypes for your architecture model.

```
pf = systemcomposer.profile.Profile.createProfile('mProfile');
b = pf.addStereotype('BaseComp', 'AppliesTo','Component','Abstract', true);
s = pf.addStereotype('sComp', 'Parent',b);
```

Apply Profile and Stereotypes

Apply the profile and stereotypes to your architecture model.

```
m.Architecture.applyProfile(pf.Name)
comps(1).applyStereotype(s.FullyQualifiedName)
```

Find the Element

Find the element in your architecture model based on a System Composer query.

```
import systemcomposer.query.*;
[p, elem] = find(m, HasStereotype(IsStereotypeDerivedFrom('mProfile.BaseComp')),...
'Recurse', true, 'IncludeReferenceModels', true)
```

```
p = 1x1 cell array
    {'exModel/c1'}


elem =
  Component with properties:

     IsAdapterComponent: 0
           Architecture: [1x1 systemcomposer.arch.Architecture]
                   Name: 'c1'
                 Parent: [1x1 systemcomposer.arch.Architecture]
                  Ports: [0x0 systemcomposer.arch.ComponentPort]
              OwnedPorts: [0x0 systemcomposer.arch.ComponentPort]
      OwnedArchitecture: [1x1 systemcomposer.arch.Architecture]
                Position: [15 15 65 65]
                   Model: [1x1 systemcomposer.arch.Model]
          SimulinkHandle: 2.0004
    SimulinkModelHandle: 3.6621e-04
                    UUID: '8f332ab3-1084-426c-8b02-2d5b55f90e4b'
            ExternalUID: ''
```

Clean Up

Uncomment to remove the model and the profile.

```
% m.close('force');
% systemcomposer.profile.Profile.closeAll;
```

Find Ports in Architecture Model

1. Create a model to query and create two components.

   ```
   m = systemcomposer.createModel('exModel');
   comps = m.Architecture.addComponent({'c1','c2'});
   port = comps(1).Architecture.addPort('cport1','in');
   ```

2. Create a query to find ports that contain the letter `'c'` in their `'Name'` property, that returns only the elements.

   ```
   constraint = contains(systemcomposer.query.Property('Name'),'c');
   find(m,constraint,'Recurse',true,'IncludeReferenceModels',true,'ElementType','Port')
   ```

Find Architecture Element Paths that Satisfy Query

```
import systemcomposer.query.*;
scKeylessEntrySystem
modelObj = systemcomposer.openModel('KeylessEntryArchitecture');
find(modelObj,HasStereotype(IsStereotypeDerivedFrom('AutoProfile.BaseComponent')),...
 modelObj.Architecture,'Recurse',true,'IncludeReferenceModels',true)
```

Input Arguments

`object` — Model

model object

Model, specified as a `systemcomposer.arch.Model` object to query using the `constraint`.

`constraint` — Query

query constraint object

Query, specified as a `systemcomposer.query.Constraint` object representing specific conditions. A constraint can contain a sub-constraint that can be joined together with another constraint using `AND` or `OR`. A constraint can also be negated using `NOT`.

Query Objects and Conditions for Constraints

Query Object	Condition
`Property`	A non-evaluated value for the given property or stereotype property.
`PropertyValue`	An evaluated property value from a System Composer object or a stereotype property.
`HasPort`	A component has a port that satisfies the given sub-constraint.
`HasInterface`	A port has an interface that satisfies the given sub-constraint.
`HasInterfaceElement`	An interface has an interface element that satisfies the given sub-constraint.
`HasStereotype`	An architecture element has a stereotype that satisfies the given sub-constraint.
`IsInRange`	A property value is within the given range.
`AnyComponent`	An element is a component and not a port or connector.
`IsStereotypeDerivedFrom`	A stereotype is derived from the given stereotype.

`rootArch` — Root architecture of the model
character vector

Root architecture of the model, specified as a character vector.

Data Types: `char`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `find(model,constraint,'Recurse',true,'IncludeReferenceModels',true)`

`Recurse` — Option to recursively search through model
`true` or 1 (default) | `false` or 0

Option to recursively search through model, or only search the specific layer, specified as the comma-separated pair consisting of `'Recurse'` and a numeric or logical 1 (`true`) to recursively search or 0 (`false`) to only search the specific layer.

Example: `find(model,constraint,'Recurse',true)`

Data Types: `logical`

`IncludeReferenceModels` — Option to search for reference architectures
`false` or 0 (default) | `true` or 1

Option to search for reference architectures, or to not include referenced architectures, specified as the comma-separated pair consisting of `'IncludeReferenceModels'` and a numeric or logical 0 (`false`) to not include referenced architectures or 1 (`true`) to search for referenced architectures.

Example: `find(model,constraint,'IncludeReferenceModels',true)`

Data Types: `logical`

`ElementType` — Option to search by type
`'Component'` (default) | `'Port'` | `'Connector'`

Option to search by type, specified as the comma-separated pair consisting of `'ElementType'` and `'Component'` to select components to satisfy the query, `'Port'` to select ports to satisfy the query, or `'Connector'` to select connectors to satisfy the query.

Example: `find(model,constraint,'ElementType','Port')`

Data Types: `char`

Output Arguments

`paths` — Element paths
cell array of element paths

Element paths, returned as a cell array of element paths that satisfy `constraint`.

`elements` — Elements
element objects

Elements, returned as `systemcomposer.arch.Element` objects that satisfy `constraint`.

See Also

`createViewArchitecture` | `systemcomposer.query.Constraint`

Topics
“Build an Architecture Model from Command Line”

Introduced in R2019a

find

Find stereotype by name

Syntax

```
stereotype = systemcomposer.profile.Stereotype.find(name)
```

Description

`stereotype = systemcomposer.profile.Stereotype.find(name)` finds a stereotype by name.

Input Arguments

`name` — Name of stereotype

character vector

Name of stereotype, specified as a character vector as a fully-qualified name in the form `'<profile>.<stereotype>'`.

Data Types: `char`

Output Arguments

`stereotype` — Stereotype found

stereotype object

Stereotype found, returned as a `systemcomposer.profile.Stereotype` object.

See Also

`setDefaultComponentStereotype` | `setDefaultConnectorStereotype` | `setDefaultPortStereotype` | `systemcomposer.profile.Stereotype`

Introduced in R2019a

find

Find profile by name

Syntax

```
profile = systemcomposer.profile.Profile.find(name)
```

Description

`profile = systemcomposer.profile.Profile.find(name)` finds a profile by name.

Input Arguments

`name` — Name of profile

character vector

Name of profile, specified as a character vector.

Data Types: `char`

Output Arguments

`profile` — Profile found

profile object

Profile found, returned as a `systemcomposer.profile.Profile` object.

See Also

`close` | `closeAll` | `createProfile` | `load` | `open` | `save` | `systemcomposer.profile.Profile`

Topics

“Define Profiles and Stereotypes”

Introduced in R2019a

getActiveChoice

Get active choice on variant component

Syntax

```
choice = getActiveChoice(variantComponent)
```

Description

`choice = getActiveChoice(variantComponent)` finds which choice is active for the variant component.

Examples

Get Active Choice

Create a model, get the root architecture, create one variant component, add two choices for the variant component, set the active choice, and find the active choice.

```
model = systemcomposer.createModel('archModel');
arch = get(model,'Architecture');
variant = addVariantComponent(arch,'Component1');
compList = addChoice(variant,{'Choice1','Choice2'});
setActiveChoice(variant,compList(2));
comp = getActiveChoice(variant)
```

```
comp =

  Component with properties:

     IsAdapterComponent: 0
           Architecture: [1×1 systemcomposer.arch.Architecture]
                   Name: 'Choice2'
                 Parent: [1×1 systemcomposer.arch.Architecture]
                  Ports: [0×0 systemcomposer.arch.ComponentPort]
              OwnedPorts: [0×0 systemcomposer.arch.ComponentPort]
       OwnedArchitecture: [1×1 systemcomposer.arch.Architecture]
                Position: [15 15 65 65]
                   Model: [1×1 systemcomposer.arch.Model]
          SimulinkHandle: 85.0006
     SimulinkModelHandle: 78.0002
                    UUID: '23b62204-f0e2-48a2-8bd6-4689f003def4'
            ExternalUID: ''
```

Input Arguments

`variantComponent` — Architecture component

variant component object

Architecture component, specified as a `systemcomposer.arch.VariantComponent` object with multiple choices.

Output Arguments

`choice` — Handle of chosen variant

component object

Handle of chosen variant, returned as a `systemcomposer.arch.Component` object.

See Also

`addChoice` | `getChoices` | `setActiveChoice`

Topics

“Create Variants”

Introduced in R2019a

getChoices

Get available choices in variant component

Syntax

```
compList = getChoices(variantComponent)
```

Description

`compList = getChoices(variantComponent)` returns the list of choices available for a variant component.

Examples

Get First Choice

Create a model, get the root architecture, create a one variant component, add two choices for the variant component, and get the first choice.

```
model = systemcomposer.createModel('archModel');
arch = get(model,'Architecture');
variant = addVariantComponent(arch,'Component1');
compList = addChoice(variant,{'Choice1','Choice2'});
choices = getChoices(variant);
choices(1)
```

```
ans =

  Component with properties:

     IsAdapterComponent: 0
           Architecture: [1×1 systemcomposer.arch.Architecture]
                   Name: 'Choice1'
                 Parent: [1×1 systemcomposer.arch.Architecture]
                  Ports: [0×0 systemcomposer.arch.ComponentPort]
              OwnedPorts: [0×0 systemcomposer.arch.ComponentPort]
       OwnedArchitecture: [1×1 systemcomposer.arch.Architecture]
                Position: [15 15 65 65]
                   Model: [1×1 systemcomposer.arch.Model]
          SimulinkHandle: 99.0010
     SimulinkModelHandle: 94.0002
                    UUID: '533d7f63-41e2-40fd-afe8-d081729849f0'
            ExternalUID: ''
```

Input Arguments

`variantComponent` — Architecture component

variant component object

Architecture component, specified as a `systemcomposer.arch.VariantComponent` object with multiple choices.

Output Arguments

`compList` — Choices available for variant component
array of component objects

Choices available for variant component, returned as an array of `systemcomposer.arch.Component` objects.

See Also

`addChoice` | `getActiveChoice` | `setActiveChoice`

Topics
“Create Variants”

Introduced in R2019a

getCondition

Return variant control on choice within variant component

Syntax

```
expression = getCondition(variantComponent,choice)
```

Description

`expression = getCondition(variantComponent,choice)` returns the variant control on the choice within the variant component.

Examples

Get Condition

Create a model, get the root architecture, create on variant component, add two choices for the variant component, set the active variant choice, set a condition, and get the condition.

```
model = systemcomposer.createModel('archModel');
arch = get(model,'Architecture');
mode = 1;
variant = addVariantComponent(arch,'Component1');
compList = addChoice(variant,{'Choice1','Choice2'});
setActiveChoice(variant,compList(2));
setCondition(variant,compList(2),'mode == 2');
exp = getCondition(variant,compList(2))
```

```
exp =

    'mode == 2'
```

Input Arguments

`variantComponent` — Architecture component

variant component object

Architecture component, specified as a `systemcomposer.arch.VariantComponent` object. This component contains multiple choices.

`choice` — Choice in variant component

component object

Choice in variant component whose control string is returned by this function, specified by a `systemcomposer.arch.Component` object.

Output Arguments

`expression` — Control string

character vector

Control string that controls the selection of the particular `choice`, returned as a character vector.

Data Types: `char`

See Also

`addVariantComponent` | `makeVariant` | `setActiveChoice` | `setCondition`

Topics

“Create Variants”

Introduced in R2019a

getDefaultStereotype

Get default stereotype for profile

Syntax

```
stereotype = getDefaultStereotype(profile)
```

Description

`stereotype = getDefaultStereotype(profile)` gets the default stereotype for a profile.

Input Arguments

`profile` — Profile
profile object

Profile, specified as a `systemcomposer.profile.Profile` object.

Output Arguments

`stereotype` — Stereotype
stereotype object

Stereotype, returned as a `systemcomposer.profile.Stereotype` object.

See Also

`addStereotype` | `createProfile` | `getStereotype` | `setDefaultStereotype`

Topics
“Create a Profile and Add Stereotypes”

Introduced in R2019a

getDestinationElement

Gets signal elements selected on destination port for connection

Syntax

```
selectedElems = getDestinationElement(connector)
```

Description

`selectedElems = getDestinationElement(connector)` gets selected signal elements on destination port for connection specified by `connector`.

Examples

Selected Element on Destination Port Connection

Get the selected element on destination port for a connection.

```
modelName = 'archModel';
arch = systemcomposer.createModel(modelName); % Create model
rootArch = get(arch,'Architecture'); % Get architecture

newComponent = addComponent(rootArch,'Component1'); % Add component
outPortComp = addPort(newComponent.Architecture,...
'testSig','out'); % Create out-port on component
outPortArch = addPort(rootArch,'testSig','out'); % Create out-port on architecture
compSrcPort = getPort(newComponent,'testSig'); % Extract component port object
archDestPort = getPort(rootArch,'testSig'); % Extract architecture port object

interface = arch.InterfaceDictionary.addInterface('interface'); % Add interface
interface.addElement('x'); % Create interface element
archDestPort.setInterface(interface); % Set interface on architecture port

conns = connect(compSrcPort,archDestPort,'DestinationElement','x'); % Connect ports
elem = getDestinationElement(conns)
```

```
elem =

  1×1 cell array

    {'x'}
```

Input Arguments

`connector` — Connection between ports

connector object

Connection between ports, specified as a `systemcomposer.arch.Connector` object.

Output Arguments

`selectedElems` — Selected signal element names

character vector

Selected signal element names, returned as a character vector.

Data Types: `char`

See Also

`addComponent` | `addElement` | `addInterface` | `addPort` | `connect` | `createModel` | `getPort` | `getSourceElement` | `setInterface` | `systemcomposer.arch.Connector`

Topics

“Create an Architecture Model”

Introduced in R2020b

getElement

Get object for signal interface element

Syntax

```
element = getElement(interface,elementName)
```

Description

`element = getElement(interface,elementName)` gets the object for an element in a signal interface.

Examples

Get Object for Named Element

Add an interface `newsignal` to the interface dictionary of the model, and add an element `newelement` with type `double`. Then get the object for the element.

```
arch = systemcomposer.createModel('newmodel',0);
interface = addInterface(arch.InterfaceDictionary,'newsignal');
addElement(interface,'newelement','Type','double');
element = getElement(interface,'newelement')
```

```
element =
  SignalElement with properties:

      Interface: [1×1 systemcomposer.interface.SignalInterface]
           Name: 'newelement'
           Type: 'double'
     Dimensions: '1'
          Units: ''
     Complexity: 'real'
        Minimum: '[]'
        Maximum: '[]'
    Description: ''
           UUID: 'f42c8166-e4ad-4488-926a-293050016e1a'
    ExternalUID: ''
```

Input Arguments

`interface` — Interface object

signal interface object

Interface object containing elements to be identified, specified as a `systemcomposer.interface.SignalInterface` object.

`elementName` — Name of element to be identified

character vector

Name of element to be identified, specified as a character vector.

Data Types: `char`

Output Arguments

`element` — New signal element object

signal element object

New signal element object in an interface, returned as a `systemcomposer.interface.SignalElement` object.

See Also

`addElement` | `getInterface` | `removeElement`

Topics

“Define Interfaces”

Introduced in R2019a

getEvaluatedPropertyValue

Get evaluated value of property from component

Syntax

```
[value] = getEvaluatedPropertyValue(compObj,qualifiedPropName)
```

Description

`[value] = getEvaluatedPropertyValue(compObj,qualifiedPropName)` obtains the evaluated value of a property specified on the component.

Input Arguments

`compObj` — Component to get property value from

component object

Component to get property value from, specified as a `systemcomposer.arch.Component` or `systemcomposer.arch.VariantComponent` object.

`qualifiedPropName` — Qualified property name

character vector

Qualified property name, specified as a character vector in the form `'<profile>.<stereotype>.<property>'`.

Data Types: `char`

Output Arguments

`value` — Property value

`double` (default) | `single` | `int64` | `int32` | `int16` | `int8` | `uint64` | `uint32` | `uint8` | `boolean` | `string` | enumeration class name

Property value, returned as a data type that depends on how the property is defined in the profile.

See Also

`getValue` | `setValue`

Topics

“Write Analysis Function”

Introduced in R2019a

getInterface

Get object for named interface in interface dictionary

Syntax

```
interface = getInterface(dictionary,name)
```

Description

`interface = getInterface(dictionary,name)` gets the object for a named interface in the interface dictionary.

Examples

Add Interface

Add an interface `'newInterface'` to the interface dictionary of the model. Obtain the interface object.

```
addInterface(arch.InterfaceDictionary,'newInterface')
interface = getInterface(arch.InterfaceDictionary,'newInterface')
```

```
interface =
  SignalInterface with properties:
     Dictionary: [1×1 systemcomposer.interface.Dictionary]
           Name: 'newInterface'
       Elements: [0×0 systemcomposer.interface.SignalElement]
           UUID: '438b5004-6cab-40eb-955b-37e0df5a914f'
    ExternalUID: ''
```

Input Arguments

`dictionary` — Data dictionary

dictionary object

Data dictionary, specified as a `systemcomposer.interface.Dictionary` object. This is the data dictionary attached to the model. It could be the local dictionary of the model or an external data dictionary.

`name` — Name of interface

character vector

Name of interface, specified as a character vector.

Data Types: `char`

Output Arguments

`interface` — Object for named interface
signal interface object

Object for named interface, returned as a `systemcomposer.interface.SignalInterface` object.

See Also

`addElement` | `addInterface` | `removeElement`

Topics
“Define Interfaces”

Introduced in R2019a

getInterfaceNames

Get names of all interfaces in interface dictionary

Syntax

```
interfaceNames = getInterfaceNames(dictionary)
```

Description

`interfaceNames = getInterfaceNames(dictionary)` gets the names of all interfaces in the interface dictionary.

Examples

Get Interface Names

```
interfaceNames = getInterfaceNames(arch.InterfaceDictionary)
```

Input Arguments

`dictionary` — Data dictionary

dictionary object

Data dictionary attached to the model, specified as a `systemcomposer.interface.Dictionary` object for the local dictionary of the model or an external data dictionary.

Output Arguments

`interfaceNames` — Interface names

array of character vectors

Interface names, specified as an array of character vectors.

Data Types: `char`

See Also

`addInterface` | `getInterface` | `removeInterface`

Topics

“Define Interfaces”

Introduced in R2019a

getPort

Get port from component

Syntax

```
port = getPort(compObj,portName)
```

Description

`port = getPort(compObj,portName)` gets the port on this component with a specified name.

Input Arguments

`compObj` — Component to get port from

component object

Component from which to get the port, specified as a `systemcomposer.arch.Component` or `systemcomposer.arch.VariantComponent` object.

`portName` — Name of port to find

character vector

Name of port to find, specified as a string or character vector.

Data Types: `char`

Output Arguments

`port` — Port of this component

component port

Port of the component, returned as a `systemcomposer.arch.ComponentPort` object.

See Also

`addElement` | `getElement` | `getInterface` | `removeElement`

Introduced in R2019a

getProperty

Get property value corresponding to stereotype applied to element

Syntax

```
[propertyValue,propertyUnits] = getProperty(element,propertyName)
```

Description

`[propertyValue,propertyUnits] = getProperty(element,propertyName)` obtains the value and units of the property specified in the `propertyName` argument. Get the property corresponding to an applied stereotype by qualified name `'<stereotype>.<property> '`.

Examples

Get Property from Component

Get the `weight` property from a component with `sysComponent` stereotype applied.

```
>> [val, units] = getProperty(element,'sysComponent.weight')
val =
    '0'
units =
    'kg'
```

Input Arguments

`element` — Architecture model element
component object | port object | connector object

Architecture model element, specified as a `systemcomposer.arch.Architecture`, `systemcomposer.arch.Component`, `systemcomposer.arch.ComponentPort`, `systemcomposer.arch.ArchitecturePort`, `systemcomposer.arch.Connector`, or `systemcomposer.arch.Element` object.

`propertyName` — Name of property
character vector

Name of property, specified as a character vector as a fully qualified name in the form `'<stereotype>.<property>'`.

Data Types: `char`

Output Arguments

`propertyValue` — Value of property
character vector | numeric | enumeration

Value of property, returned as a character vector, numeric, or enumeration value.

Data Types: `char` | `double` | `enum`

`propertyUnits` — Units of property
character vector

Units of property to interpret property values, returned as a character vector.

Data Types: `char`

See Also

`removeProperty` | `setProperty`

Topics
“Set Tags and Properties for Analysis”

Introduced in R2019a

getScenario

Get allocation scenario

Syntax

```
scenario = getScenario(name)
```

Description

`scenario = getScenario(name)` gets the allocation scenario in this set with the given name, if one exists.

Input Arguments

`name` — Name of scenario

character vector

Name of scenario, specified as a character vector.

Data Types: `char`

Output Arguments

`scenario` — Allocation scenario

allocation scenario object

Allocation scenario, returned as a `systemcomposer.allocation.AllocationScenario` object.

See Also

`createScenario` | `deleteScenario`

Topics

“Create and Manage Allocations”

Introduced in R2020b

getSourceElement

Gets signal elements selected on source port for connection

Syntax

```
selectedElems = getSourceElement(connector)
```

Description

`selectedElems = getSourceElement(connector)` gets selected signal elements on source port for connection specified by `connector`.

Examples

Selected Element on Source Port Connection

Get the selected element on source port for a connection.

```
modelName = 'archModel';
arch = systemcomposer.createModel(modelName); % Create model
rootArch = get(arch,'Architecture'); % Get architecture

newComponent = addComponent(rootArch,'Component1'); % Add component
inPortComp = addPort(newComponent.Architecture,...
'testSig','in'); % Create in-port on component
inPortArch = addPort(rootArch,'testSig','in'); % Create in-port on architecture
compDestPort = getPort(newComponent,'testSig'); % Extract component port object
archSrcPort = getPort(rootArch,'testSig'); % Extract architecture port object

interface = arch.InterfaceDictionary.addInterface('interface'); % Add interface
interface.addElement('x'); % Create interface element
archSrcPort.setInterface(interface); % Set interface on architecture port

conns = connect(archSrcPort,compDestPort,'SourceElement','x'); % Connect ports
elem = getSourceElement(conns)
```

```
elem =

  1×1 cell array

    {'x'}
```

Input Arguments

`connector` — Connection between ports

connector object

Connection between ports, specified as a `systemcomposer.arch.Connector` object.

Output Arguments

`selectedElems` — Selected signal element names

character vector

Selected signal element names, returned as a character vector.

Data Types: `char`

See Also

`addComponent` | `addElement` | `addInterface` | `addPort` | `connect` | `createModel` | `getDestinationElement` | `getPort` | `setInterface` | `systemcomposer.arch.Connector`

Topics

“Create an Architecture Model”

Introduced in R2020b

getStereotype

Find stereotype in profile by name

Syntax

```
stereotype = getStereotype(profile,name)
```

Description

`stereotype = getStereotype(profile,name)` finds a stereotype in a profile by name.

Input Arguments

`profile` — Profile with stereotype

profile object

Profile with stereotype, specified as a `systemcomposer.profile.Profile` object.

`name` — Name of stereotype

character vector

Name of stereotype, specified as a character vector.

Data Types: `char`

Output Arguments

`stereotype` — Found stereotype

stereotype object

Found stereotype, returned as a `systemcomposer.profile.Stereotype` object.

See Also

`addStereotype` | `systemcomposer.profile.Profile`

Topics

“Create a Profile and Add Stereotypes”

Introduced in R2019a

getStereotypes

Get stereotypes applied on element of architecture model

Syntax

```
stereotypes = getStereotypes(element)
```

Description

`stereotypes = getStereotypes(element)` gets an array of fully qualified stereotype names that have been applied on the element.

Examples

Get Stereotypes

```
stereotypes = getStereotypes(component_handle)
```

Input Arguments

`element` — Model element

architecture object | component object | port object | connector object

Model element, specified as a `systemcomposer.arch.Architecture`, `systemcomposer.arch.Component`, `systemcomposer.arch.ComponentPort`, `systemcomposer.arch.ArchitecturePort`, or `systemcomposer.arch.Connector` object.

Output Arguments

`stereotypes` — Fully qualified name list of stereotypes

cell array of character vectors

Fully qualified name list of stereotypes, specified as a cell array of character vectors in the form `'<profile>.<stereotype>'`.

Data Types: `char`

See Also

`applyStereotype` | `batchApplyStereotype` | `removeStereotype`

Topics

“Use Stereotypes and Profiles”

Introduced in R2019a

getValue

Get value of property from element instance

Syntax

```
[value,unit] = getValue(instance,property)
```

Description

`[value,unit] = getValue(instance,property)` obtains the `property` of the `instance` and assigns it to `value`. This function is part of the instance API that you can use to analyze the model iteratively, element by element.`instance` refers to the element instance on which the iteration is being performed.

Examples

Get Weight Property

Assume that a `MechComponent` stereotype is attached to the specification of the instance.

```
weightValue = getValue(instance,'MechComponent.weight');
```

Input Arguments

`instance` — Element instance

architecture instance | component instance | port instance | connector instance

Element instance, specified by a `systemcomposer.analysis.ArchitectureInstance`, `systemcomposer.analysis.ComponentInstance`, `systemcomposer.analysis.PortInstance`, or `systemcomposer.analysis.ConnectorInstance` object. This function is part of the instance API that you can use to analyze the model iteratively, element by element.`instance` refers to the element instance on which the iteration is being performed.

`property` — Property

character vector

Property, specified as a character vector in the form `'<stereotype>.<property>'`.

Data Types: `char`

Output Arguments

`value` — Property value

`double` (default) | `single` | `int64` | `int32` | `int16` | `int8` | `uint64` | `uint32` | `uint8` | `boolean` | `string` | enumeration class name

Property value, returned as a data type that depends on how the property is defined in the profile.

`unit` — Property unit
character vector

Property unit, returned as a character vector that describes the unit of the property as defined in the profile.

See Also

`setValue` | `systemcomposer.analysis.Instance`

Topics
“Write Analysis Function”

Introduced in R2019a

HasInterface

Package: `systemcomposer.query`

Create query to select architecture elements with interface on port based on specified sub-constraint

Syntax

```
query = HasInterface(sub-constraint)
```

Description

`query = HasInterface(sub-constraint)` creates a `query` object that the `find` method and the `createViewArchitecture` method use to select architecture elements with an interface that satisfies the given sub-constraint.

Examples

Construct Query to Select All Port Interfaces

Select all of the port interfaces in an architecture model with matching criteria.

Import the package that contains all of the System Composer queries.

```
import systemcomposer.query.*;
```

Open the Simulink project file.

```
scKeylessEntrySystem
```

Open the model.

```
m = systemcomposer.openModel('KeylessEntryArchitecture');
```

Create a query for all the interfaces in a port with `'KeyFOBPosition'` in the `'Name'` and run the query.

```
constraint = HasPort(HasInterface(contains(Property('Name'),'KeyFOBPosition')));
portInterfaces = find(m,constraint,'Recurse',true,'IncludeReferenceModels',true)
```

```
portInterfaces =

  10×1 cell array

    {'KeylessEntryArchitecture/Door Lock//Unlock System'                     }
    {'KeylessEntryArchitecture/Door Lock//Unlock System/Door Lock Controller' }
    {'KeylessEntryArchitecture/Engine Control System'                        }
    {'KeylessEntryArchitecture/Engine Control System/Keyless Start Controller'}
    {'KeylessEntryArchitecture/FOB Locator System'                           }
    {'KeylessEntryArchitecture/FOB Locator System/FOB Locator Module'        }
    {'KeylessEntryArchitecture/Lighting System'                              }
    {'KeylessEntryArchitecture/Lighting System/Lighting Controller'          }
```

```
    {'KeylessEntryArchitecture/Sound System'                              }
    {'KeylessEntryArchitecture/Sound System/Sound Controller'             }
```

Input Arguments

`sub-constraint` — Condition restricting the query

query constraint object

Condition restricting the query, specified as a `systemcomposer.query.Constraint` object.

Example: `contains(Property('Name'),'KeyFOBPosition')`

Output Arguments

`query` — Query

query constraint object

Query, returned as a `systemcomposer.query.Constraint` object.

See Also

`HasInterfaceElement` | `HasPort` | `createViewArchitecture` | `find` | `systemcomposer.query.Constraint`

Topics

“Creating Architectural Views Programmatically”

Introduced in R2019b

HasInterfaceElement

Package: `systemcomposer.query`

Create query to select architecture elements with interface element on interface based on specified sub-constraint

Syntax

```
query = HasInterfaceElement(sub-constraint)
```

Description

`query = HasInterfaceElement(sub-constraint)` creates a `query` object that the `find` method and the `createViewArchitecture` method use to select architecture elements with an interface element that satisfies the given sub-constraint.

Examples

Construct Query to Select All Interface Elements

Select all of the port interface elements in an architecture model with matching criteria.

Import the package that contains all of the System Composer queries.

```
import systemcomposer.query.*;
```

Open the Simulink project file.

```
scExampleSmallUAV
```

Open the model.

```
m = systemcomposer.openModel('scExampleSmallUAVModel');
```

Create a query for all the interface elements with `'c'` in the `'Name'` and run the query.

```
constraint = HasPort(HasInterface(HasInterfaceElement(contains(Property('Name'),'c'))));
elements = find(m,constraint,'Recurse',true,'IncludeReferenceModels',true)
```

```
elements =

  4×1 cell array

    {'scExampleSmallUAVModel/FlightComputer'           }
    {'scExampleSmallUAVModel/FlightComputer/Main Board'}
    {'scExampleSmallUAVModel/Payload'                  }
    {'scExampleSmallUAVModel/Payload/Payload'          }
```

Input Arguments

`sub-constraint` — Condition restricting the query

query constraint object

Condition restricting the query, specified as a `systemcomposer.query.Constraint` object.

Example: `contains(Property('Name'),'c')`

Output Arguments

`query` — Query
query constraint object

Query, returned as a `systemcomposer.query.Constraint` object.

See Also

`HasInterface` | `HasPort` | `createViewArchitecture` | `find` | `systemcomposer.query.Constraint`

Topics
“Creating Architectural Views Programmatically”

Introduced in R2019b

HasPort

Package: `systemcomposer.query`

Create query to select architecture elements with port on component based on specified sub-constraint

Syntax

```
query = HasPort(sub-constraint)
```

Description

`query = HasPort(sub-constraint)` creates a `query` object that the `find` method and the `createViewArchitecture` method use to select architecture elements with a port that satisfies the given sub-constraint.

Examples

Construct Query to Select All Sensor Component Ports

Select all of the sensor component ports in an architecture model.

Import the package that contains all of the System Composer queries.

```
import systemcomposer.query.*;
```

Open the Simulink project file.

```
scKeylessEntrySystem
```

Open the model.

```
m = systemcomposer.openModel('KeylessEntryArchitecture');
```

Create a query for all the ports in a component with `'Sensor'` in the `'Name'` and run the query.

```
constraint = HasPort(contains(Property('Name'),'Sensor'));
sensorComp = find(m,constraint,'Recurse',true,'IncludeReferenceModels',true)
```

```
sensorComp =

  1×1 cell array

    {'KeylessEntryArchitecture/Door Lock//Unlock System/Door Lock Controller'}
```

Input Arguments

`sub-constraint` — Condition restricting the query

query constraint object

Condition restricting the query, specified as a `systemcomposer.query.Constraint` object.

Example: `contains(Property('Name'),'Sensor')`

Output Arguments

query — Query
query constraint object

Query, returned as a `systemcomposer.query.Constraint` object.

See Also

`HasInterface` | `HasInterfaceElement` | `createViewArchitecture` | `find` | `systemcomposer.query.Constraint`

Topics
“Creating Architectural Views Programmatically”

Introduced in R2019b

HasStereotype

Package: `systemcomposer.query`

Create query to select architecture elements with stereotype based on specified sub-constraint

Syntax

```
query = HasStereotype(sub-constraint)
```

Description

`query = HasStereotype(sub-constraint)` creates a `query` object that the `find` method and the `createViewArchitecture` method use to select architecture elements with a stereotype that satisfies the given sub-constraint.

Examples

Construct Query to Select All Hardware Components

Select all of the hardware components in an architecture model.

Import the package that contains all of the System Composer queries.

```
import systemcomposer.query.*;
```

Open the Simulink project file.

```
scKeylessEntrySystem
```

Open the model.

```
m = systemcomposer.openModel('KeylessEntryArchitecture');
```

Create a query for all the hardware components and run the query, displaying one of them.

```
constraint = HasStereotype(IsStereotypeDerivedFrom('AutoProfile.HardwareComponent'));
hwComp = find(m,constraint,'Recurse',true,'IncludeReferenceModels',true);
hwComp(16)
```

```
ans =

  1×1 cell array

    {'KeylessEntryArchitecture/FOB Locator System/Center Receiver/PWM'}
```

Input Arguments

`sub-constraint` — Condition restricting the query

query constraint object

Condition restricting the query, specified as a `systemcomposer.query.Constraint` object.

Example: `IsStereotypeDerivedFrom('AutoProfile.HardwareComponent')`

Output Arguments

query — Query
query constraint object

Query, returned as a `systemcomposer.query.Constraint` object.

See Also

`IsStereotypeDerivedFrom` | `createViewArchitecture` | `find` | `systemcomposer.query.Constraint`

Topics
“Creating Architectural Views Programmatically”

Introduced in R2019b

importModel

Import model information from MATLAB tables

Syntax

```
archModel = systemcomposer.importModel(modelName,components,ports,
connections,portInterfaces,requirementLinks)
archModel = systemcomposer.importModel(importStruct)
[archModel,idMappingTable,importLog,errorLog] = systemcomposer.importModel(
___ )
```

Description

`archModel = systemcomposer.importModel(modelName,components,ports, connections,portInterfaces,requirementLinks)` creates a new architecture model based on MATLAB tables that specify components, ports, connections, port interfaces, and requirements.

`archModel = systemcomposer.importModel(importStruct)` creates a new architecture model based on a structure of MATLAB tables that specify components, ports, connections, port interfaces, and requirements.

`[archModel,idMappingTable,importLog,errorLog] = systemcomposer.importModel( ___ )` creates a new architecture model with output arguments `idMappingTable` with table information, `importLog` to display import information, and `errorLog` to display import error information.

Input Arguments

`modelName` — Name of model to be created

character vector

Name of model to be created, specified as a character vector.

Example: `'importedModel'`

Data Types: `char`

`components` — Model component information

MATLAB table

Model component information, specified as a MATLAB table. The component table must include name, unique ID, parent component ID, and component type for each component. It can also include other relevant information such as referenced model, stereotype qualifier name, and so on, required to construct the architecture hierarchy.

Data Types: `table`

`ports` — Model port information

MATLAB table

Model port information, specified as a MATLAB table. The ports table must include port name, direction, port ID, and component ID information. `portInterfaces` information may also be required to assign ports to components.

Data Types: `table`

`connections` — Model connections information
MATLAB table

Model connections information, specified as a MATLAB table. The requirement links table must include label, source ID, destination type, and destination ID information.

Data Types: `table`

`portInterfaces` — Model port interfaces information
MATLAB table

Model port interfaces information, specified as a MATLAB table. The port interfaces table must include name, ID, parent ID, data type, dimensions, units, complexity, minimum, and maximum information.

Data Types: `table`

`requirementLinks` — Model requirement links information
MATLAB table

Model requirement links information, specified as a MATLAB table. The requirement links table must include label, source ID, destination type, destination ID, and type information.

Data Types: `table`

`importStruct` — Model tables
structure

Model tables, specified as a structure containing tables `components`, `ports`, `connections`, `portInterfaces`, and `requirementLinks`.

Data Types: `struct`

Output Arguments

`archModel` — Handle to architecture model
architecture object

Handle to architecture model, specified as a `systemcomposer.arch.Architecture` object.

`idMappingTable` — Mapping of custom IDs and internal UUIDs of elements
structure

Mapping of custom IDs and internal UUIDs of elements, returned as a `struct` of MATLAB tables.

Data Types: `struct`

`importLog` — Confirmation that elements were imported
cell array of character vectors

Confirmation that elements were imported, returned as a cell array of character vectors.

Data Types: `char`

`errorLog` — Errors reported during import process
array of message objects

Errors reported during import process, returned as an array of message `MException` objects. You can obtain the error text by calling the `getString` method on each `MException` object.

Examples

Import and Export Architectures

This example shows how to import and export architectures. In System Composer, an architecture is fully defined by three sets of information:

- Component information
- Port information
- Connection information

You can import an architecture into System Composer when this information is defined in, or converted into, MATLAB tables.

In this example, the architecture information of a simple UAV system is defined in an Excel spreadsheet and is used to create a System Composer architecture model. It also links elements to the specified system level requirement. You can modify the files in this example to import architectures defined in external tools, when the data includes the required information. The example also shows how to export this architecture information from System Composer architecture model to an Excel spreadsheet.

Architecture Definition Data

You can characterize the architecture as a network of components and import by defining components, ports, connections, interfaces and requirement links in MATLAB tables. The component table must include name, unique ID, and parent component ID for each component. It can also include other relevant information required to construct the architecture hierarchy for referenced model, and stereotype qualifier names. The port table must include port name, direction, component, and port ID information. Port interface information may also be required to assign ports to components. The connection table includes information to connect ports. At a minimum, this table must include the connection ID, source port ID, and destination port ID.

The systemcomposer.importModel(importModelName) API :

- Reads stereotype names from Component table and load the profiles
- Creates components and attaches ports
- Creates connections using the connection map
- Sets interfaces on ports
- Links elements to specified requirements
- Saves referenced models

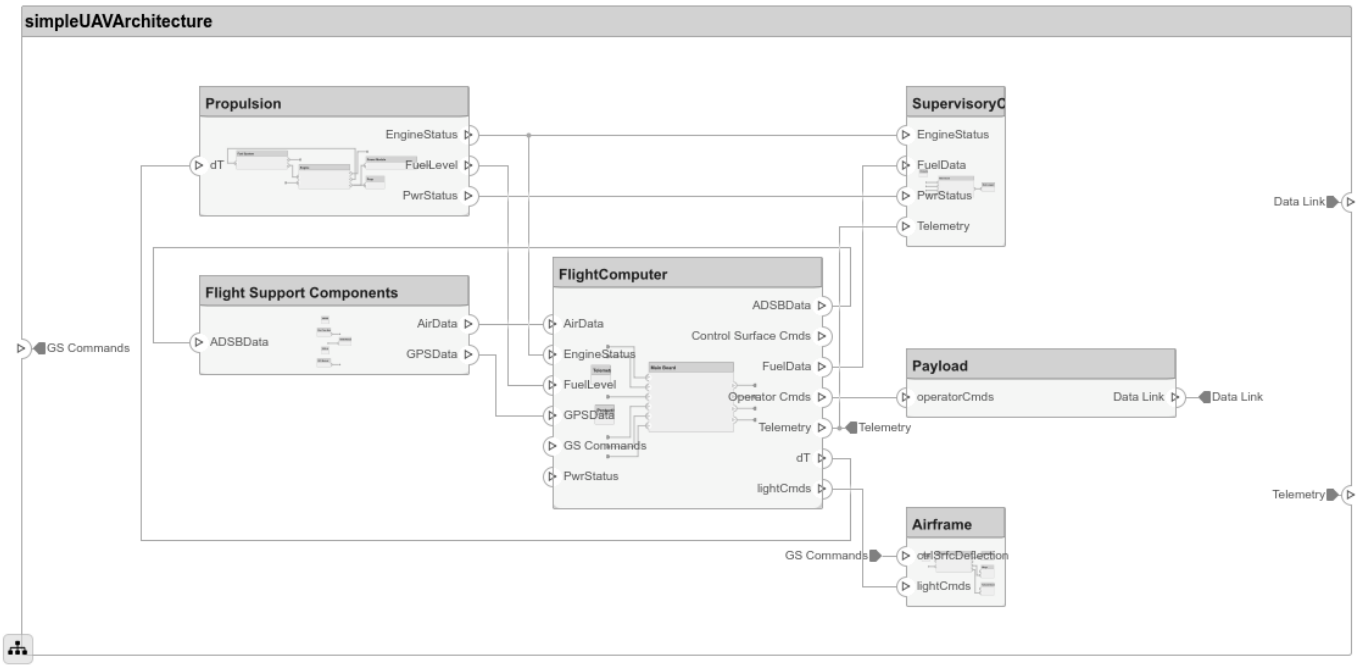
- Saves the architecture model

Make sure the current directory is writable because this example will create files.

```
[stat, fa] = fileattrib(pwd);
if ~fa.UserWrite
    disp('This script must be run in a writable directory');
    return;
end
% Instantiate adapter class to read from Excel.
modelName = 'simpleUAVArchitecture';
% importModelFromExcel function reads the Excel file and creates the MATLAB
% tables.
importAdapter = ImportModelFromExcel('SmallUAVModel.xls','Components','Ports','Connections','Por
importAdapter.readTableFromExcel();
```

Import an Architecture

```
model = systemcomposer.importModel(modelName,importAdapter.Components,importAdapter.Ports,importA
% Auto-arrange blocks in the generated model
Simulink.BlockDiagram.arrangeSystem(modelName);
```



Export an Architecture

You can export an architecture to MATLAB tables and then convert to an external file

```
exportedSet = systemcomposer.exportModel(modelName);
% The output of the function is a structure that contains the component table, port table,
% connection table, the interface table, and the requirement links table.
% Save the above structure to excel file.
SaveToExcel('ExportedUAVModel',exportedSet);
```

Close Model

```
bdclose(modelName);
```

See Also

`systemcomposer.exportModel`

Topics

“Import and Export Architecture Models”

Introduced in R2019a

inlineComponent

Inline reference architecture into model

Syntax

```
componentObj = inlineComponent(component,inlineFlag)
```

Description

`componentObj = inlineComponent(component,inlineFlag)` makes contents of the architecture model inline, referenced by the specified `component` and breaks the link to the reference model. If `inlineFlag` is `false`, then the contents are removed and only interfaces remain.

Examples

Reuse Component

Save the component `robotcomp` in the architecture model `Robot.slx` and reference it from another component, `robotArm` so that `robotArm` uses the architecture of `robotcomp`. Inline `robotcomp` so that its architecture can be edited independently.

```
saveAsModel(robotcomp,'Robot');
linkToModel(robotArm,'Robot');
inlineComponent(robotArm,true);
```

Input Arguments

`component` — Architecture component

component object

Architecture component linked to an architecture model, specified as a `systemcomposer.arch.Component` object.

`inlineFlag` — Control contents of inlined component

`true` or 1 | `false` or 0

Control contents of inlined component, specified as 1 (`true`) if contents of the referenced architecture model are copied to the component architecture, and 0 (`false`) if the contents are not copied and only ports and interfaces are inlined. If the component is a Simulink behavior, `inlineFlag` is ignored and set to `false`.

Data Types: `logical`

Output Arguments

`componentObj` — Architecture component

component object

Architecture component, returned as a `systemcomposer.arch.Component` object.

See Also

`linkToModel` | `saveAsModel`

Topics

“Decompose and Reuse Components”

Introduced in R2019a

instantiate

Create analysis instance from specification

Syntax

```
instance = instantiate(model,properties,name)
instance = instantiate(model,profile,name)
```

Description

`instance = instantiate(model,properties,name)` creates an instance of a model for analysis.

`instance = instantiate(model,profile,name)` creates an instance of a model for analysis with all stereotypes in a profile.

Examples

Instantiate All Properties of a Stereotype

Instantiate all properties of a stereotype that will be applied to specific elements during instantiation.

Create a profile for latency characteristics.

```
profile = systemcomposer.profile.Profile.createProfile('LatencyProfile');

latencybase = profile.addStereotype('LatencyBase');
latencybase.addProperty('latency','Type','double');
latencybase.addProperty('dataRate','Type','double','DefaultValue','10');

connLatency = profile.addStereotype('ConnectorLatency','Parent',...
'LatencyProfile.LatencyBase');
connLatency.addProperty('secure','Type','boolean');
connLatency.addProperty('linkDistance','Type','double');

nodeLatency = profile.addStereotype('NodeLatency','Parent',...
'LatencyProfile.LatencyBase');
nodeLatency.addProperty('resources','Type','double','DefaultValue','1');

portLatency = profile.addStereotype('PortLatency','Parent',...
'LatencyProfile.LatencyBase');
portLatency.addProperty('queueDepth','Type','double');
portLatency.addProperty('dummy','Type','int32');

profile.save;
```

Instantiate all properties of a stereotype.

```
model = systemcomposer.createModel('archModel');
NodeLatency = struct('elementKinds',['Component']);
ConnectorLatency = struct('elementKinds',['Connector']);
LatencyBase = struct('elementKinds',['Connector','Port','Component']);
PortLatency = struct('elementKinds',['Port']);

LatencyAnalysis = struct('NodeLatency',NodeLatency, ...
                'ConnectorLatency',ConnectorLatency, ...
                'PortLatency',PortLatency, ...
```

```
                'LatencyBase',LatencyBase);

properties = struct('LatencyProfile',LatencyAnalysis);
instantiate(model.Architecture,properties,'NewInstance')
```

Instantiate Specific Properties of a Stereotype

Instantiate specific properties of a stereotype that will be applied to specific elements during instantiation.

```
NodeLatency = struct('elementKinds',["Component"], ...
                'properties',struct('resources',true));
ConnectorLatency = struct('elementKinds',["Connector"], ...
                'properties',struct('secure',true,'linkDistance',true));
LatencyBase = struct('elementKinds',[], ...
                'properties',struct('dataRate',true,'latency',false));
PortLatency = struct('elementKinds',["Port"], ...
                'properties',struct('queueDepth',true));

LatencyAnalysis = struct('NodeLatency',NodeLatency, ...
                'ConnectorLatency',ConnectorLatency, ...
                'PortLatency',PortLatency, ...
                'LatencyBase',LatencyBase);

properties = struct('LatencyProfile',LatencyAnalysis);
instantiate(model.Architecture,properties,'NewInstance')
```

Instantiate All Stereotypes in a Profile

Instantiate all stereotypes already in a profile that will be applied to elements during instantiation.

```
instantiate(model.Architecture,'LatencyProfile','NewInstance')
```

Input Arguments

`model` — Model architecture

architecture object

Model architecture from which instance is generated, specified as a `systemcomposer.arch.Architecture` object.

Example: `model.Architecture`

`properties` — Stereotype properties

`struct`

Structure containing profile, stereotype, and property information through which the user can specify which stereotypes and properties need to be instantiated.

`name` — Name of instance

character vector

Name of instance generated from the model, specified as a character vector.

Example: `'NewInstance'`

Data Types: `char`

`profile` — Profile name
character vector

Profile name, specified as a character vector.

Example: `'LatencyProfile'`

Data Types: `char`

Output Arguments

`instance` — Element instance
instance object

Element instance, returned as a `systemcomposer.analysis.ArchitectureInstance`, `systemcomposer.analysis.ComponentInstance`, `systemcomposer.analysis.PortInstance`, or `systemcomposer.analysis.ConnectorInstance` object. This function is part of the instance API that you can use to analyze the model iteratively, element by element. The value refers to the element instance on which the iteration is being performed.

See Also

`deleteInstance` | `loadInstance` | `saveInstance` | `systemcomposer.analysis.Instance`

Topics
“Write Analysis Function”

Introduced in R2019a

isArchitecture

Find if instance is architecture instance

Syntax

```
flag = isArchitecture(instance)
```

Description

`flag = isArchitecture(instance)` finds whether the instance is an architecture instance.

Input Arguments

`instance` — Element instance

architecture instance | component instance | port instance | connector instance

Element instance, specified by a `systemcomposer.analysis.ArchitectureInstance`, `systemcomposer.analysis.ComponentInstance`, `systemcomposer.analysis.PortInstance`, or `systemcomposer.analysis.ConnectorInstance` object. This function is part of the instance API that you can use to analyze the model iteratively, element by element.`instance` refers to the element instance on which the iteration is being performed.

Output Arguments

`flag` — Whether instance is architecture

`true` | `false`

This argument is `true` if the instance is an architecture.

Data Types: `logical`

See Also

`isComponent` | `isConnector` | `isPort`

Topics

“Write Analysis Function”

Introduced in R2019a

isComponent

Find if instance is component instance

Syntax

```
flag = isComponent(instance)
```

Description

`flag = isComponent(instance)` finds whether the instance is a component instance.

Input Arguments

`instance` — Element instance

architecture instance | component instance | port instance | connector instance

Element instance, specified by a `systemcomposer.analysis.ArchitectureInstance`, `systemcomposer.analysis.ComponentInstance`, `systemcomposer.analysis.PortInstance`, or `systemcomposer.analysis.ConnectorInstance` object. This function is part of the instance API that you can use to analyze the model iteratively, element by element.`instance` refers to the element instance on which the iteration is being performed.

Output Arguments

`flag` — Whether instance is component

`true` | `false`

This argument is `true` if the instance is a component.

Data Types: `logical`

See Also

`isArchitecture` | `isConnector` | `isPort`

Topics

“Write Analysis Function”

Introduced in R2019a

isConnector

Find if instance is connector instance

Syntax

```
flag = isConnector(instance)
```

Description

`flag = isConnector(instance)` finds whether the instance is a connector instance.

Input Arguments

`instance` — Element instance

architecture instance | component instance | port instance | connector instance

Element instance, specified by a `systemcomposer.analysis.ArchitectureInstance`, `systemcomposer.analysis.ComponentInstance`, `systemcomposer.analysis.PortInstance`, or `systemcomposer.analysis.ConnectorInstance` object. This function is part of the instance API that you can use to analyze the model iteratively, element by element.`instance` refers to the element instance on which the iteration is being performed.

Output Arguments

`flag` — Whether instance is connector

`true` | `false`

This argument is `true` if the instance is a connector.

Data Types: `logical`

See Also

`isArchitecture` | `isComponent` | `isPort`

Topics

“Write Analysis Function”

Introduced in R2019a

IsInRange

Package: `systemcomposer.query`

Create query to select a range of property values

Syntax

```
query = IsInRange(propertyName,beginRangeValue,endRangeValue)
```

Description

`query = IsInRange(propertyName,beginRangeValue,endRangeValue)` creates a `query` object that the `find` method and the `createViewArchitecture` method use to select a range of values from a specified `propertyName`.

Examples

Find Model Elements that Satisfy Property Range

Import the package that contains all of the System Composer queries.

```
import systemcomposer.query.*;
```

Open the Simulink project file.

```
scKeylessEntrySystem
```

Open the model.

```
m = systemcomposer.openModel('KeylessEntryArchitecture');
```

Create a query to find values from 10ms to 40ms in the `'Latency'` property.

```
constraint = IsInRange(PropertyValue('AutoProfile.BaseComponent.Latency'),...
Value(10,'ms'),Value(40,'ms'));
latency = find(m,constraint,'Recurse',true,'IncludeReferenceModels',true)
```

```
latency =

  5×1 cell array

    {'KeylessEntryArchitecture/Door Lock//Unlock System/Front Driver Door Lock Actuator'}
    {'KeylessEntryArchitecture/Door Lock//Unlock System/Front Pass Door Lock Actuator'  }
    {'KeylessEntryArchitecture/Door Lock//Unlock System/Rear Driver Door Lock Actuator' }
    {'KeylessEntryArchitecture/Door Lock//Unlock System/Rear Pass Door Lock Actuator'   }
    {'KeylessEntryArchitecture/Sound System/Dashboard Speaker'
```

Input Arguments

`propertyName` — Property name

character vector

Property name for model element, specified as a character vector as fully qualified name `'<profile name>.<stereotype name>.<property name>'` or any property on the designated class.

Example: `'Name'`

Example: `'AutoProfile.BaseComponent.Latency'`

Data Types: `char`

`beginRangeValue` — Beginning range value
value object

Beginning range value for `propertyName`, specified as a `systemcomposer.query.Value` object.

Example: `Value(20)`

Example: `Value(5,'ms')`

`endRangeValue` — Ending range value
value object

Ending range value for `propertyName`, specified as a `systemcomposer.query.Value` object.

Example: `Value(100)`

Example: `Value(20,'ms')`

Output Arguments

`query` — Query
query constraint object

Query, returned as a `systemcomposer.query.Constraint` object.

See Also

`createViewArchitecture` | `find` | `systemcomposer.query.Constraint`

Topics
“Creating Architectural Views Programmatically”

Introduced in R2019b

isPort

Find if instance is port instance

Syntax

```
flag = isPort(instance)
```

Description

`flag = isPort(instance)` finds whether the instance is a port instance.

Input Arguments

`instance` — Element instance

architecture instance | component instance | port instance | connector instance

Element instance, specified by a `systemcomposer.analysis.ArchitectureInstance`, `systemcomposer.analysis.ComponentInstance`, `systemcomposer.analysis.PortInstance`, or `systemcomposer.analysis.ConnectorInstance` object. This function is part of the instance API that you can use to analyze the model iteratively, element by element.`instance` refers to the element instance on which the iteration is being performed.

`flag` — Whether instance is port

`true` | `false`

This argument is `true` if the instance is a port.

Data Types: `logical`

See Also

`isArchitecture` | `isComponent` | `isConnector`

Topics

“Write Analysis Function”

Introduced in R2019a

isReference

Find if component is reference to another model

Syntax

```
flag = isReference(compObj)
```

Description

`flag = isReference(compObj)` returns whether or not the component is a reference to another model.

Input Arguments

`compObj` — Component to get port from

base component object | architecture instance | component instance | port instance | connector instance

Component to get port from, specified as a `systemcomposer.arch.BaseComponent`, `systemcomposer.analysis.ArchitectureInstance`, `systemcomposer.analysis.ComponentInstance`, `systemcomposer.analysis.PortInstance`, or `systemcomposer.analysis.ConnectorInstance`object.

Output Arguments

`flag` — Whether component is reference

`true` | `false`

This argument is `true` if the component is a reference.

Data Types: `logical`

See Also

Topics

“Write Analysis Function”

Introduced in R2019a

IsStereotypeDerivedFrom

Package: `systemcomposer.query`

Create query to select stereotype derived from a fully qualified name

Syntax

```
query = IsStereotypeDerivedFrom(name)
```

Description

`query = IsStereotypeDerivedFrom(name)` creates a `query` object that the `find` method and the `createViewArchitecture` method use to select a stereotype from the fully qualified `name`.

Input Arguments

`name` — Fully qualified stereotype name

character vector

Fully qualified stereotype name, specified as a character vector as `'<profile name>.<stereotype name>'`.

Example: `'AutoProfile.BaseComponent'`

Data Types: `char`

Output Arguments

`query` — Query

query constraint object

Query, returned as a `systemcomposer.query.Constraint` object.

See Also

`HasStereotype` | `createViewArchitecture` | `find` | `systemcomposer.query.Constraint`

Topics

“Creating Architectural Views Programmatically”

Introduced in R2019b

iterate

Iterate over model elements

Syntax

```
iterate(architecture,iterType,iterFunction)
iterate( ___ ,Name,Value)
iterate( ___ ,additionalArgs)
```

Description

`iterate(architecture,iterType,iterFunction)` iterates over components in the architecture in the order specified by `iterType` and invokes the function specified by the function handle `iterFunction` on each component.

`iterate( ___ ,Name,Value)` iterates over components in the architecture, with additional options specified by one or more name-value pair arguments.

`iterate( ___ ,additionalArgs)` passes all trailing arguments as arguments to `iterFunction`.

Examples

Battery Capacity Computation

Open the example “Battery Sizing and Automotive Electrical System Analysis”.

```
archModel = systemcomposer.openModel('scExampleAutomotiveElectricalSystemAnalysis');
% Instantiate battery sizing class used by analysis function to store
% analysis results.
objcomputeBatterySizing = computeBatterySizing;
% Run the analysis using the iterator
iterate(archModel,'Topdown',@computeLoad,objcomputeBatterySizing);
```

Input Arguments

`architecture` — Architecture to iterate over

architecture object

Architecture to iterate over, specified as an `systemcomposer.arch.Architecture` object.

`iterType` — Iteration type

`'PreOrder'` | `'PostOrder'` | `'TopDown'` | `'BottomUp'`

Iteration type, specified as `'PreOrder'`, `'PostOrder'`, `'TopDown'`, or `'BottomUp'`.

Data Types: `char`

`iterFunction` — Iteration function

function handle

Iteration function, specified as a function handle to be iterated on each component.

Data Types: `string`

`additionalArgs` — Additional function arguments
function arguments

Additional function arguments, specified as a comma-separated list of arguments to be passed to `iterFunction`.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `iterate(archModel,'Topdown',@computeLoad,objcomputeBatterySizing)`

`Recurse` — Option to recursively iterate through model components
`true` or 1 (default) | `false` or 0

Option to recursively iterate through model components, specified as the comma-separated pair consisting of `'Recurse'` and a numeric or logical 1 (`true`) to recursively iterate or 0 (`false`) to iterate over components only in this architecture and not navigate into the architectures of child components.

Data Types: `logical`

`IncludePorts` — Option to iterate over components and architecture ports
`false` or 0 (default) | `true` or 1

Option to iterate over components and architecture ports, specified as the comma-separated pair consisting of `'IncludePorts'` and a numeric or logical 0 (`false`) to only iterate over components or 1 (`true`) to iterate over components and architecture ports.

Data Types: `logical`

`FollowConnectivity` — Option to ensure components are visited according to how they are connected from source to destination
`false` or 0 (default) | `true` or 1

Option to ensure components are visited according to how they are connected from source to destination, specified as the comma-separated pair consisting of `'FollowConnectivity'` and a numeric or logical 0 (`false`) or 1 (`true`). If this option is specified as 1 (`true`), iteration type has to be either `'TopDown'` or `'BottomUp'`. If any other option is specified, iteration defaults to `'TopDown'`.

Data Types: `logical`

See Also

`instantiate` | `systemcomposer.analysis.Instance`

Topics
“Analyze Architecture”

Introduced in R2019a

linkDictionary

Package: `systemcomposer.arch`

Link data dictionary to architecture model

Syntax

```
linkDictionary(obj,dictionaryFile)
```

Description

`linkDictionary(obj,dictionaryFile)` associates the specified Simulink data dictionary with the model.

Input Arguments

`obj` — Architecture model object

model object

Architecture model object from which the dictionary link is to be added, specified as a `systemcomposer.arch.Model` object.

`dictionaryFile` — Dictionary file name

character vector

Dictionary file name with the `.sldd` extension, specified as a character vector.

Data Types: `char`

See Also

`systemcomposer.createDictionary` | `systemcomposer.openDictionary` | `unlinkDictionary`

Topics

“Save, Link, and Delete Interfaces”

Introduced in R2019a

linkToModel

Link component to a model

Syntax

```
modelHandle = linktoModel(component,modelName)
modelHandle = linktoModel(component,modelFilePath)
```

Description

`modelHandle = linktoModel(component,modelName)` links from the component to a model.

`modelHandle = linktoModel(component,modelFilePath)` links from the component to a model.

Examples

Reuse Component

Save the component `robotComp` in the architecture model `Robot.slx` and reference it from another component, `robotArm` so that `robotArm` uses the architecture of `robotComp`.

```
saveAsModel(robotComp,'Robot');
linkToModel(robotArm,'Robot');
```

Input Arguments

`component` — Architecture component
component object

Architecture component with no children, specified as a `systemcomposer.arch.Component` object.

`modelName` — Model name
character vector

Model name for an existing model that defines the architecture or behavior of the component, specified as a character vector. Models of the same name prioritize protected models.

Example: `'Robot'`

Data Types: `char`

`modelFilePath` — Model file path
character vector

Model file path for an existing model that defines the architecture or behavior of the component, specified as a character vector.

Example: `'Model.slx'`

Example: `'ProtectedModel.slxp'`

Data Types: `char`

Output Arguments

`modelHandle` — Handle to the linked model

numeric value

Handle to the linked model, returned as a numeric value.

Data Types: `double`

See Also

`inlineComponent` | `saveAsModel`

Topics

“Decompose and Reuse Components”

Introduced in R2019a

load

Load allocation set

Syntax

```
allocSet = systemcomposer.allocation.load(name)
```

Description

`allocSet = systemcomposer.allocation.load(name)` loads the allocation set with the given name, if it exists, on the MATLAB path.

Examples

Load Allocation Set and Open in Allocation Editor

```
% Load the allocation set MyNewAllocation.mldatx
allocSet = systemcomposer.allocation.load('MyNewAllocation')

% Open the allocation editor
systemcomposer.allocation.editor()
```

Input Arguments

`name` — Name of allocation set

model object | character vector

Name of allocation set, specified as a `systemcomposer.arch.Model` object or the name of the model as a character vector.

Output Arguments

`allocSet` — Allocation set

allocation set object

Allocation set, returned as a `systemcomposer.allocation.AllocationSet` object.

See Also

`closeAll` | `createAllocationSet` | `open`

Topics

“Create and Manage Allocations”

Introduced in R2020b

load

Load profile from file

Syntax

```
profile = systemcomposer.profile.Profile.load(fileName)
```

Description

`profile = systemcomposer.profile.Profile.load(fileName)` loads a profile from a file name.

Input Arguments

`fileName` — File name for profile

character vector

File name for profile, specified as a character vector. Profile must be available on the MATLAB path.

Example: `'ProfileFile.xml'`

Data Types: `char`

Output Arguments

`profile` — Profile loaded

profile object

Profile loaded, returned as a `systemcomposer.profile.Profile` object.

See Also

`close` | `closeAll` | `createProfile` | `find` | `open` | `save` | `systemcomposer.profile.Profile`

Topics

“Define Profiles and Stereotypes”

Introduced in R2019a

loadInstance

Load architecture instance

Syntax

```
loadInstance(fileName,overwrite)
```

Description

`loadInstance(fileName,overwrite)` loads an architecture instance from a MAT-file.

Input Arguments

`fileName` — File that contains architecture instance

character vector

This is a MAT-file that was previously saved with an architecture instance.

`overwrite` — Whether to overwrite instance if it already exists in workspace

`true` | `false`

If `true`, the load operation overwrites duplicate instances in the workspace.

See Also

`deleteInstance` | `instantiate` | `saveInstance` | `systemcomposer.analysis.Instance` | `updateInstance`

Topics

“Write Analysis Function”

Introduced in R2019a

loadModel

Load architecture model

Syntax

```
model = systemcomposer.loadModel(modelName)
```

Description

`model = systemcomposer.loadModel(modelName)` loads the architecture model with name `modelName` and returns its handle. The loaded model is not displayed.

Examples

```
model = systemcomposer.loadModel('new_arch')
```

Input Arguments

`modelName` — Name of architecture model

character vector

Name of architecture model, specified as a character vector. Architecture model must exist on the MATLAB path.

Example: 'new_arch'

Data Types: `char`

Output Arguments

`model` — Architecture model handle

model object

Architecture model handle, returned as a `systemcomposer.arch.Model` object.

See Also

`open` | `save`

Topics

“Create an Architecture Model”

Introduced in R2019a

loadProfile

Load profile by name

Syntax

```
profile = systemcomposer.loadProfile(profileName)
```

Description

`profile = systemcomposer.loadProfile(profileName)` loads a profile with the specified file name.

Input Arguments

`profileName` — Name of profile

character vector

Name of profile, specified as a character vector. Profile must be available on the MATLAB path.

Example: 'new_profile'

Data Types: `char`

Output Arguments

`profile` — Profile handle

profile object

Profile handle, returned as a `systemcomposer.profile.Profile` object.

Examples

```
systemcomposer.loadProfile('new_profile')
profile = systemcomposer.loadProfile('new_profile')
```

See Also

`applyProfile` | `createProfile` | `systemcomposer.profile.Profile`

Topics

“Define Profiles and Stereotypes”

Introduced in R2019a

lookup

Package: `systemcomposer.arch`

Search for architecture element

Syntax

```
element = lookup(object,Name,Value)
```

Description

`element = lookup(object,Name,Value)` finds an architecture element based on its universal unique identifier (UUID) or full path.

Examples

Look Up Component by Path

```
lookup(arch,'Path','RobotSystem/Sensors')
```

```
ans =

  Component with properties:

                  Name: 'Sensors'
                Parent: [1×1 systemcomposer.arch.Architecture]
                 Ports: [1×2 systemcomposer.arch.ComponentPort]
            OwnedPorts: []
          Architecture: [1×1 systemcomposer.arch.Architecture]
     OwnedArchitecture: []
              Position: [275 75 391 161]
                 Model: [1×1 systemcomposer.arch.Model]
                  UUID: 'f43c9d51-9dc6-43fc-b3af-95d458b81248'
        SimulinkHandle: 9.0002
    SimulinkModelHandle: 2.0002
            ExternalUID: ''
```

Input Arguments

`object` — Architecture model object

model object

Architecture model object to look up using the `UUID`, specified as a `systemcomposer.arch.Model` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `lookup(arch,'Path','RobotSystem/Sensors')`

`UUID` — Search by UUID
character vector

Search by UUID, specified as the comma-separated pair consisting of `'UUID'` and a character vector of the `UUID`.

Example: `lookup(arch,'UUID','f43c9d51-9dc6-43fc-b3af-95d458b81248')`

Data Types: `char`

`SimulinkHandle` — Search by simulink handle
double

Search by Simulink handle, specified as the comma-separated pair consisting of `'SimulinkHandle'` and a double of the `SimulinkHandle` value.

Example: `lookup(arch,'SimulinkHandle',9.0002)`

Data Types: `double`

`Path` — Search by full path
character vector

Search by file path, specified as the comma-separated pair consisting of `'Path'` and a character vector with the path defined.

Example: `lookup(arch,'Path','RobotSystem/Sensors')`

Data Types: `char`

Output Arguments

`element` — Model element
element object

Model element, returned as a `systemcomposer.arch.Architecture`, `systemcomposer.arch.Component`, `systemcomposer.arch.ComponentPort`, `systemcomposer.arch.ArchitecturePort`, or `systemcomposer.arch.Connector` object.

See Also

`find`

Topics
“Analyze Architecture”

Introduced in R2019a

makeVariant

Convert component to variant choice

Syntax

```
[variantComp,choices] = makeVariant(components)
```

Description

`[variantComp,choices] = makeVariant(components)` converts `components` to variant choices and returns the parent component and available choices.

Input Arguments

`components` — Architecture components

array of architecture component objects

Architecture components to be converted to variants, specified as an array of `systemcomposer.arch.Component` objects.

Output Arguments

`variantComp` — Component containing variants

variant component object

Component containing variants, returned as a `systemcomposer.arch.VariantComponent` object.

`choices` — Variant choice names

cell array of character vectors

Variant choice names, returned as a cell array of character vectors.

Data Types: `char`

See Also

`addChoice` | `addVariantComponent` | `getChoices`

Topics

“Create Variants”

Introduced in R2019a

open

Open profile

Syntax

```
open(profile)
```

Description

`open(profile)` opens a profile in the Profile Editor.

Input Arguments

`profile` — Profile

profile object

Profile, specified as a `systemcomposer.profile.Profile` object.

See Also

`close` | `closeAll` | `createProfile` | `find` | `load` | `save`

Topics

“Define Profiles and Stereotypes”

Introduced in R2019a

open

Open allocation set in allocation editor

Syntax

```
allocSet = systemcomposer.allocation.open(name)
```

Description

`allocSet = systemcomposer.allocation.open(name)` opens allocation set in the allocation editor if the allocation set is on the MATLAB path.

Input Arguments

`name` — Name of allocation set

allocation set object | character vector

Name of allocation set, specified as an `systemcomposer.allocation.AllocationSet` object or the name as a character vector.

See Also

`createAllocationSet` | `load`

Topics

“Create and Manage Allocations”

Introduced in R2020b

open

Package: `systemcomposer.arch`

Open architecture model

Syntax

```
open(objModel)
```

Description

`open(objModel)` opens the specified model in System Composer.

`open` is a method for the class `systemcomposer.arch.Model`.

Examples

Create and Open a Model

```
Model = systemcomposer.createModel('modelName');
open(Model)
```

Input Arguments

`objModel` — Model to open in editor

model object

Model to open in editor, specified as a `systemcomposer.arch.Model` object.

See Also

`createModel` | `openModel`

Topics

“Create an Architecture Model”

Introduced in R2019a

systemcomposer.openDictionary

Package: `systemcomposer`

Open data dictionary

Syntax

```
dict_id = systemcomposer.openDictionary(dictionaryName)
```

Description

`dict_id = systemcomposer.openDictionary(dictionaryName)` opens an existing Simulink data dictionary to hold interfaces and returns a handle to the `systemcomposer.interface.Dictionary` object.

Examples

Open an Existing Dictionary

```
dict_id = systemcomposer.openDictionary('my_dictionary.sldd')
```

Input Arguments

`dictionaryName` — Name of existing data dictionary

character vector

Name of existing data dictionary, specified as a character vector. The name must include the `.sldd` extension.

Example: `'my_dictionary.sldd'`

Data Types: `char`

Output Arguments

`dict_id` — Handle to the dictionary

dictionary object

Handle to the dictionary, returned as a `systemcomposer.interface.Dictionary` object.

See Also

`linkDictionary` | `systemcomposer.createDictionary` | `unlinkDictionary`

Topics

“Save, Link, and Delete Interfaces”

Introduced in R2019a

openModel

Open System Composer architecture model

Syntax

```
model = systemcomposer.openModel(modelName)
```

Description

`model = systemcomposer.openModel(modelName)` opens the model with name `modelName` for editing and returns its handle.

Examples

```
model = systemcomposer.openModel('new_arch')
```

Input Arguments

`modelName` — Name of new model

character vector

Name of new model, specified as a character vector. Model must exist on the MATLAB path.

Example: `'new_arch'`

Data Types: `char`

Output Arguments

`model` — Model handle

model object

Model handle, returned as a `systemcomposer.arch.Model` object.

See Also

`close` | `open`

Topics

“Create an Architecture Model”

Introduced in R2019a

openViews

Open architecture views editor

Syntax

```
openViews(objModel)
```

Description

`openViews(objModel)` opens the architecture views editor for the specified model. If the model is already open, `openViews` will bring the views to the front.

The method `openViews` is for the class `systemcomposer.arch.Model`.

Input Arguments

`objModel` — Name of model

model object (default) | character vector

Name of model, specified as a character vector or a `systemcomposer.arch.Model` object.

Data Types: `char`

See Also

Introduced in R2019b

Property

Package: `systemcomposer.query`

Create query to select non-evaluated values for properties or stereotype properties for objects based on specified property name

Syntax

```
query = Property(name)
```

Description

`query = Property(name)` creates a `query` object that the `find` method and the `createViewArchitecture` method use to select non-evaluated values for properties or stereotype properties for objects based on specified property `name`.

Examples

Find Model Elements that Satisfy Property

Import the package that contains all of the System Composer queries.

```
import systemcomposer.query.*;
```

Open the Simulink project file.

```
scKeylessEntrySystem
```

Open the model.

```
m = systemcomposer.openModel('KeylessEntryArchitecture');
```

Create a query to find components that contain the character vector `'Sensor'` in their `'Name'` property and run the query, displaying the first.

```
constraint = contains(Property('Name'),'Sensor');
sensors = find(m,constraint,'Recurse',true,'IncludeReferenceModels',true);
sensors(1)
```

```
ans =

1×1 cell array

    {'KeylessEntryArchitecture/Door Lock//Unlock System/Front Driver Door Lock Sensor'}
```

Input Arguments

`name` — Property name

character vector

Property name for model element, specified as a character vector as fully qualified name `'<profile name>.<stereotype name>.<property name>'` or any property on the designated class.

Example: `'Name'`

Example: `'AutoProfile.BaseComponent.Latency'`

Data Types: `char`

Output Arguments

`query` — Query
query constraint object

Query, returned as a `systemcomposer.query.Constraint` object.

See Also

`PropertyValue` | `createViewArchitecture` | `find` | `systemcomposer.query.Constraint`

Topics
“Creating Architectural Views Programmatically”

Introduced in R2019b

PropertyValue

Package: `systemcomposer.query`

Create query to select property from object or stereotype property and then evaluate property value

Syntax

```
query = PropertyValue(name)
```

Description

`query = PropertyValue(name)` creates a `query` object that the `find` method and the `createViewArchitecture` method use to select properties or stereotype properties for objects based on specified property `name` and then evaluate the property value.

Examples

Find Model Elements that Satisfy Property Value

Import the package that contains all of the System Composer queries.

```
import systemcomposer.query.*;
```

Open the Simulink project file.

```
scKeylessEntrySystem
```

Open the model.

```
m = systemcomposer.openModel('KeylessEntryArchitecture');
```

Create a query to find components that contain the character vector `'Sensor'` in their `'Name'` property and run the query.

```
constraint = PropertyValue('AutoProfile.BaseComponent.Latency')==30;
latency = find(m,constraint,'Recurse',true,'IncludeReferenceModels',true)
```

```
latency =

4×1 cell array

    {'KeylessEntryArchitecture/Door Lock//Unlock System/Front Driver Door Lock Actuator'}
    {'KeylessEntryArchitecture/Door Lock//Unlock System/Front Pass Door Lock Actuator'  }
    {'KeylessEntryArchitecture/Door Lock//Unlock System/Rear Driver Door Lock Actuator' }
    {'KeylessEntryArchitecture/Door Lock//Unlock System/Rear Pass Door Lock Actuator'   }
```

Input Arguments

`name` — Property name

character vector

Property name for model element, specified as a character vector as fully qualified name `'<profile name>.<stereotype name>.<property name>'` or any property on the designated class.

Example: `'Name'`

Example: `'AutoProfile.BaseComponent.Latency'`

Data Types: `char`

Output Arguments

`query` — Query
query constraint object

Query, returned as a `systemcomposer.query.Constraint` object.

See Also

`Property` | `createViewArchitecture` | `find` | `systemcomposer.query.Constraint`

Topics
“Creating Architectural Views Programmatically”

Introduced in R2019b

removeComponent

Package: `systemcomposer.view`

Remove component from view

Syntax

```
removeComponent(object,compPath)
```

Description

`removeComponent(object,compPath)` removes the component with the specified path.

`removeComponent` is a method for the class `systemcomposer.view.ViewArchitecture`.

Input Arguments

`object` — View architecture

view architecture object

View architecture, specified as a `systemcomposer.view.ViewArchitecture` object.

`compPath` — Path to the component

character vector

Path to the component including the name of the top-model, specified as a character vector.

Data Types: `char`

See Also

`addComponent` | `systemcomposer.view.BaseViewComponent` | `systemcomposer.view.ComponentOccurrence` | `systemcomposer.view.ViewArchitecture` | `systemcomposer.view.ViewComponent` | `systemcomposer.view.ViewElement`

Introduced in R2019b

removeElement

Remove a signal interface element

Syntax

```
removeElement(interface,elementName)
```

Description

`removeElement(interface,elementName)` removes an element from a signal interface.

Examples

Add an Interface and an Element

Add an interface `'newInterface'` to the interface dictionary of the model and add an element with type `double` to it, then remove the element.

```
interface = addInterface(arch.InterfaceDictionary,'newInterface');
element = addElement(interface,'newElement','Type',double);
removeElement(interface,'newInterface')
```

Input Arguments

`interface` — Interface object

signal interface object

Interface object, specified as a `systemcomposer.interface.SignalInterface` object.

`elementName` — Name of element

character vector

Name of element to be removed, specified as a character vector.

Data Types: `char`

See Also

`addElement` | `getElement`

Topics

“Define Interfaces”

Introduced in R2019a

removeInterface

Remove named interface from interface dictionary

Syntax

```
removeInterface(dictionary,name)
```

Description

`removeInterface(dictionary,name)` removes a named interface from the interface dictionary.

Examples

Remove Interface

Add an interface `'newInterface'` to the interface dictionary of the model and then remove it.

```
addInterface(arch.InterfaceDictionary,'newInterface')
removeInterface(arch.InterfaceDictionary,'newInterface')
```

Input Arguments

`dictionary` — Data dictionary attached to architecture model

dictionary object

Data dictionary attached to architecture model, specified as a `systemcomposer.interface.Dictionary` object.

`name` — Name of new interface

character vector

Name of new interface, specified as a character vector.

Data Types: `char`

See Also

`addInterface` | `getInterface` | `getInterfaceNames`

Topics

“Define Interfaces”

Introduced in R2019a

removeProfile

Remove profile from model

Syntax

```
removeProfile(modelObject,profileFile)
```

Description

`removeProfile(modelObject,profileFile)` removes the profile from a model.

Examples

Remove a Profile

```
removeProfile(arch,'SystemProfile')
```

Input Arguments

`modelObject` — Architecture model
model object

Architecture model, specified as a `systemcomposer.arch.Model` object.

`profileFile` — Name of profile
character vector

Name of profile, specified as a character vector.

Example: `'SystemProfile'`

Data Types: `char`

See Also

`applyProfile` | `createProfile`

Topics
“Define Profiles and Stereotypes”

Introduced in R2019a

removeProperty

Remove property from stereotype

Syntax

`removeProperty(stereotype,propertyName)`

Description

`removeProperty(stereotype,propertyName)` removes a property from the stereotype.

Examples

Remove a Property

Add a component stereotype and add a `VoltageRating` property with value `5`. Then remove the property.

```
stereotype = addStereotype(profile,'electricalComponent','AppliesTo','Component')
property = addProperty(stereotype,'VoltageRating','DefaultValue','5');
removeProperty(stereotype,'VoltageRating');
```

Input Arguments

`stereotype` — Stereotype to which property is removed

stereotype object

Stereotype to which property is removed, specified as a `systemcomposer.profile.Stereotype` object.

`propertyName` — Name of property

character vector

Name of property to be removed, specified as a character vector.

Data Types: `char`

See Also

`addProperty` | `getProperty`

Topics

“Define Profiles and Stereotypes”

Introduced in R2019a

removeStereotype

Remove stereotype from model element

Syntax

```
removeStereotype(element,stereotype)
```

Description

`removeStereotype(element,stereotype)` removes a stereotype from the mode element. The function removes the specified stereotype if it is already applied to a model element.

Input Arguments

`element` — Model element

architecture object | component object | port object | connector object

Model element, specified as a `systemcomposer.arch.Architecture`, `systemcomposer.arch.Component`, `systemcomposer.arch.ComponentPort`, `systemcomposer.arch.ArchitecturePort`, or `systemcomposer.arch.Connector` object.

`stereotype` — Fully qualified name of stereotype

character vector

Fully qualified name of stereotype, specified as a character vector in the form `'<profile>.<stereotype>'`. The profile must already be applied to the model. The stereotype can also be specified as a `systemcomposer.profile.Stereotype` object.

Data Types: `char`

See Also

`applyStereotype` | `batchApplyStereotype` | `getStereotypes`

Topics

“Remove a Stereotype”

Introduced in R2019a

renameProfile

Rename profile in model

Syntax

```
renameProfile(modelName,oldProfileName,newProfileName)
```

Description

`renameProfile(modelName,oldProfileName,newProfileName)` renames a profile on a model from `oldProfileName` to `newProfileName`.

Input Arguments

`modelName` — Model architecture

model object | character vector

Model architecture, specified as a `systemcomposer.arch.Model` object or a character vector as the name of the model.

Example: `'MyModel'`

Example: `archModel`

Data Types: `char`

`oldProfileName` — Old profile name

character vector

Old profile name, specified as a character vector.

Example: `'MyProfile'`

Data Types: `char`

`newProfileName` — New profile name

character vector

New profile name, specified as a character vector.

Example: `'MyProfileNew'`

Data Types: `char`

See Also

`close` | `open` | `save`

Introduced in R2020b

save

Save profile as file

Syntax

```
filePath = save(profile,dirPath)
```

Description

`filePath = save(profile,dirPath)` saves profile to disk as file specified in its `Name` property with a `.xml` extension. Saves to the current directory if the optional `dirPath` is left blank.

Examples

Save Profile

Create a profile named `'NewProfile'` and save it in the current directory.

```
profile = systemcomposer.profile.Profile.createProfile('NewProfile');
path = save(profile);
```

Input Arguments

`profile` — Profile

profile object

Profile, specified as a `systemcomposer.profile.Profile` object.

`dirPath` — Path to save

character vector

Path to save, specified as a character vector. Current directory is the default if no path is specified.

Example: `'C:\Temp'`

Data Types: `char`

Output Arguments

`filePath` — File path

character vector

File path where profile is saved, returned as a character vector.

See Also

`close` | `closeAll` | `createProfile` | `find` | `load` | `open`

Topics

“Define Profiles and Stereotypes”

Introduced in R2019a

save

Save allocation set

Syntax

`save`

Description

`save` saves the allocation set.

Examples

Create Allocation Set and Save

```
% Create the allocation set with name MyNewAllocation.
allocSet = systemcomposer.allocation.createAllocationSet('MyNewAllocation', ...
     'Source_Model_Allocation', 'Target_Model_Allocation');

% Save the allocation set
allocSet.save;
```

See Also

`createAllocationSet` | `createScenario` | `deleteScenario` | `getScenario` | `systemcomposer.allocation.AllocationSet`

Topics

“Create and Manage Allocations”

Introduced in R2020b

save

Save the architecture model or data dictionary

Syntax

```
save(architecture)
save(dictionary)
```

Description

`save(architecture)` saves the architecture model to the file specified in its `Name` property.

`save(dictionary)` saves the data dictionary.

Examples

Save Model and Data Dictionary

```
save(arch);
save(arch.InterFaceDictionary);
```

Input Arguments

`architecture` — Architecture model
model object

Architecture model, specified as a `systemcomposer.arch.Model` object.

`dictionary` — Data dictionary
dictionary object

Data dictionary attached to the architecture model, specified as a `systemcomposer.interface.Dictionary` object.

See Also

`close` | `loadModel`

Topics
“Create an Architecture Model”
“Save, Link, and Delete Interfaces”

Introduced in R2019a

saveAsModel

Save architecture to separate model

Syntax

```
saveAsModel(component,modelName)
```

Description

`saveAsModel(component,modelName)` saves the architecture of the component to a separate architecture model and references the model from this component.

Examples

Save Component

Save the component `robotComp` in `Robot.slx` and reference the model.

```
saveAsModel(robotComp,'Robot');
```

Input Arguments

`component` — Architecture component
component object

Architecture component, specified as a `systemcomposer.arch.Component` object. The component must have an architecture with definition type `composition`. For other definition types, this function gives an error.

`modelName` — Model name
character vector

Model name, specified as a character vector.

Data Types: `char`

See Also

`inlineComponent` | `linkToModel`

Topics
“Decompose and Reuse Components”

Introduced in R2019a

saveInstance

Save architecture instance

Syntax

```
saveInstance(architectureInstance,fileName)
```

Description

`saveInstance(architectureInstance,fileName)` saves an architecture instance to a MAT-file.

Input Arguments

`architectureInstance` — Architecture instance
instance object

Architecture instance to be saved, specified as a `systemcomposer.analysis.ArchitectureInstance` object.

`fileName` — File to save the instance
character vector

This is a MAT-file to save the architecture instance.

Data Types: `char`

See Also

`deleteInstance` | `instantiate` | `loadInstance` | `systemcomposer.analysis.Instance` | `updateInstance`

Topics
“Write Analysis Function”

Introduced in R2019a

setActiveChoice

Set active choice on variant component

Syntax

`setActiveChoice(variantComponent,choice)`

Description

`setActiveChoice(variantComponent,choice)` sets the active choice on the variant component.

Examples

Set Active Choice

Create a model, get the root architecture, create one variant component, add two choices for the variant component, and set the active choice.

```
model = systemcomposer.createModel('archModel');
arch = get(model,'Architecture');
variant = addVariantComponent(arch,'Component1');
compList = addChoice(variant,{'Choice1','Choice2'});
setActiveChoice(variant,compList(2));
```

Input Arguments

`variantComponent` — Architecture component

variant component object

Architecture component, specified as a `systemcomposer.arch.VariantComponent` object with multiple choices.

`choice` — Active choice in a variant component

component object | label of variant choice

Active choice in a variant component, specified as a `systemcomposer.arch.Component` object or label of the variant choice as a character vector.

See Also

`addChoice` | `addVariantComponent` | `getActiveChoice` | `getChoices`

Topics

“Create Variants”

Introduced in R2019a

setComplexity

Set complexity for signal interface element

Syntax

```
setComplexity(interfaceElem,complexity)
```

Description

`setComplexity(interfaceElem,complexity)` sets the complexity for the designated signal interface element.

Examples

Set Complexity for Interface Element

Create a model named `'archModel'`, add an interface, create an interface element with a name `'x'`, and set the complexity for the interface element `'complex'`.

```
modelName = 'archModel';
arch = systemcomposer.createModel(modelName); % Create model

interface = arch.InterfaceDictionary.addInterface('interface'); % Add interface
elem = interface.addElement('x'); % Create interface element

setComplexity(elem,'complex'); % Set complexity for interface element
```

Input Arguments

`interfaceElem` — Interface element

signal element object

Interface element, specified as a `systemcomposer.interface.SignalElement` object.

`complexity` — Complexity of interface element

`'real'` (default) | `'complex'`

Complexity of interface element, specified as a character vector with values `'real'` or `'complex'`.

Data Types: `char`

See Also

`addElement` | `addInterface` | `createModel` | `systemcomposer.interface.SignalElement`

Topics

“Define Interfaces”

Introduced in R2019a

setCondition

Set condition on variant choice

Syntax

`setCondition(variantComponent,choice,expression)`

Description

`setCondition(variantComponent,choice,expression)` sets the variant control for a choice for the variant component.

Examples

Set Condition

Create a model, get the root architecture, create one variant component, add two choices for the variant component, set the active choice, and set a condition.

```
model = systemcomposer.createModel('archModel');
arch = get(model,'Architecture');
mode = 1;
variant = addVariantComponent(arch,'Component1');
compList = addChoice(variant,{'Choice1','Choice2'});
setActiveChoice(variant,compList(2));
setCondition(variant,compList(2),'mode == 2');
```

Input Arguments

`variantComponent` — Architecture component

variant component object

Architecture component, specified as a `systemcomposer.arch.VariantComponent` object. This component contains multiple choices.

`choice` — Choice in variant component

component object

Choice in variant component whose control string is set by this function, specified by a `systemcomposer.arch.Component` object.

`expression` — Control string

character vector

Control string that controls the selection of `choice`, specified as a character vector.

Data Types: `char`

See Also

`addChoice` | `addVariantComponent` | `getActiveChoice` | `getCondition` | `makeVariant` | `setActiveChoice`

Topics

"Create Variants"

Introduced in R2019a

setDefaultComponentStereotype

Set default stereotype for components

Syntax

```
setDefaultComponentStereotype(stereotype,stereotypeName)
```

Description

`setDefaultComponentStereotype(stereotype,stereotypeName)` specifies the default stereotype `stereotypeName` of the children whose parent component has `stereotype` applied.

Input Arguments

`stereotype` — Stereotype of parent component
stereotype object

Stereotype of parent component, specified as a `systemcomposer.profile.Stereotype` object.

`stereotypeName` — Fully qualified name of default stereotype
character vector

Fully qualified name of default stereotype for child components, specified as a character vector in the form `'<profile>.<stereotype>'`.

Data Types: `char`

See Also

`applyStereotype` | `removeStereotype` | `setDefaultConnectorStereotype` | `setDefaultPortStereotype`

Topics
“Define Profiles and Stereotypes”

Introduced in R2019a

setDefaultConnectorStereotype

Set default stereotype for connectors

Syntax

```
setDefaultConnectorStereotype(stereotype,stereotypeName)
```

Description

`setDefaultConnectorStereotype(stereotype,stereotypeName)` specifies the default stereotype `stereotypeName` of the connectors within the parent component that has `stereotype` applied.

Input Arguments

`stereotype` — Stereotype of parent component
stereotype object

Stereotype of parent component, specified as a `systemcomposer.profile.Stereotype` object.

`stereotypeName` — Fully qualified name of default stereotype
character vector

Fully qualified name of default stereotype for connectors, specified as a character vector in the form `'<profile>.<stereotype>'`.

Data Types: `char`

See Also

`applyStereotype` | `removeStereotype` | `setDefaultComponentStereotype` | `setDefaultPortStereotype`

Topics
“Define Profiles and Stereotypes”

Introduced in R2019a

setDefaultPortStereotype

Set default stereotype for ports

Syntax

```
setDefaultPortStereotype(stereotype,stereotypeName)
```

Description

`setDefaultPortStereotype(stereotype,stereotypeName)` specifies the default stereotype `stereotypeName` of the ports of the parent component that has `stereotype` applied.

Input Arguments

`stereotype` — Stereotype of parent component
stereotype object

Stereotype of parent component, specified as a `systemcomposer.profile.Stereotype` object.

`stereotypeName` — Fully qualified name of default stereotype
character vector

Fully qualified name of default stereotype for ports, specified as a character vector in the form `'<profile>.<stereotype>'`.

Data Types: `char`

See Also

`applyStereotype` | `removeStereotype` | `setDefaultComponentStereotype` | `setDefaultConnectorStereotype`

Topics
“Define Profiles and Stereotypes”

Introduced in R2019a

setDefaultStereotype

Set default stereotype for profile

Syntax

```
setDefaultStereotype(profile,stereotypeName)
```

Description

`setDefaultStereotype(profile,stereotypeName)` sets the default stereotype for a profile.

Input Arguments

`profile` — Profile
profile object

Profile, specified as a `systemcomposer.profile.Profile` object.

`stereotypeName` — Stereotype name
character vector

Stereotype name, specified as a character vector. The stereotype must be present in the profile.

Example: `'ComponentStereotype'`

Data Types: `char`

See Also

`addStereotype` | `createProfile` | `getDefaultStereotype` | `getStereotype`

Topics
“Create a Profile and Add Stereotypes”

Introduced in R2019a

setDescription

Set description for signal interface element

Syntax

```
setDescription(interfaceElem,description)
```

Description

`setDescription(interfaceElem,description)` sets the description for the designated signal interface element.

Examples

Set Description for Interface Element

Create a model named `'archModel'`, add an interface, create an interface element with a name `'x'`, and set the description for the interface element `'Test Description'`.

```
modelName = 'archModel';
arch = systemcomposer.createModel(modelName); % Create model

interface = arch.InterfaceDictionary.addInterface('interface'); % Add interface
elem = interface.addElement('x'); % Create interface element

setDescription(elem,'Test Description'); % Set description for interface element
```

Input Arguments

`interfaceElem` — Interface element

signal element object

Interface element, specified as a `systemcomposer.interface.SignalElement` object.

`description` — Description of interface element

character vector

Description of interface element, specified as a character vector.

Data Types: `char`

See Also

`addElement` | `addInterface` | `createModel` | `systemcomposer.interface.SignalElement`

Topics

“Define Interfaces”

Introduced in R2019a

setDimensions

Set dimensions for signal interface element

Syntax

`setDimensions(interfaceElem,dimensions)`

Description

`setDimensions(interfaceElem,dimensions)` sets the dimensions for the designated signal interface element.

Examples

Set Dimensions for Interface Element

Create a model named `'archModel'`, add an interface, create an interface element with a name `'x'`, and set the dimensions for the interface element `'2'`.

```
modelName = 'archModel';
arch = systemcomposer.createModel(modelName); % Create model

interface = arch.InterfaceDictionary.addInterface('interface'); % Add interface
elem = interface.addElement('x'); % Create interface element

setDimensions(elem,'2'); % Set dimensions for interface element
```

Input Arguments

`interfaceElem` — Interface element

signal element object

Interface element, specified as a `systemcomposer.interface.SignalElement` object.

`dimensions` — Dimensions of interface element

character vector

Dimensions of interface element, specified as a character vector.

Data Types: `char`

See Also

`addElement` | `addInterface` | `createModel` | `systemcomposer.interface.SignalElement`

Topics

“Define Interfaces”

Introduced in R2019a

setMaximum

Set maximum for signal interface element

Syntax

`setMaximum(interfaceElem,maximum)`

Description

`setMaximum(interfaceElem,maximum)` sets the maximum for the designated signal interface element.

Examples

Set Maximum for Interface Element

Create a model named `'archModel'`, add an interface, create an interface element with a name `'x'`, and set the maximum for the interface element `'5.72'`.

```
modelName = 'archModel';
arch = systemcomposer.createModel(modelName); % Create model

interface = arch.InterfaceDictionary.addInterface('interface'); % Add interface
elem = interface.addElement('x'); % Create interface element

setMaximum(elem,'5.72'); % Set maximum for interface element
```

Input Arguments

`interfaceElem` — Interface element

signal element object

Interface element, specified as a `systemcomposer.interface.SignalElement` object.

`maximum` — Maximum of interface element

character vector

Maximum of interface element, specified as a character vector.

Data Types: `char`

See Also

`addElement` | `addInterface` | `createModel` | `systemcomposer.interface.SignalElement`

Topics

“Define Interfaces”

Introduced in R2019a

setMinimum

Set minimum for signal interface element

Syntax

`setMinimum(interfaceElem,minimum)`

Description

`setMinimum(interfaceElem,minimum)` sets the minimum for the designated signal interface element.

Examples

Set Minimum for Interface Element

Create a model named `'archModel'`, add an interface, create an interface element with a name `'x'`, and set the minimum for the interface element `'1.12'`.

```
modelName = 'archModel';
arch = systemcomposer.createModel(modelName); % Create model

interface = arch.InterfaceDictionary.addInterface('interface'); % Add interface
elem = interface.addElement('x'); % Create interface element

setMinimum(elem,'1.12'); % Set minimum for interface element
```

Input Arguments

`interfaceElem` — Interface element

signal element object

Interface element, specified as a `systemcomposer.interface.SignalElement` object.

`minimum` — Minimum of interface element

character vector

Minimum of interface element, specified as a character vector.

Data Types: `char`

See Also

`addElement` | `addInterface` | `createModel` | `systemcomposer.interface.SignalElement`

Topics

“Define Interfaces”

Introduced in R2019a

setName

Set name for signal interface element

Syntax

`setName(interfaceElem,name)`

Description

`setName(interfaceElem,name)` sets the name for the designated signal interface element.

Examples

Set New Name for Interface Element

Create a model named `'archModel'`, add an interface, create an interface element with a name `'x'`, and set a new name for the interface element `'newName'`.

```
modelName = 'archModel';
arch = systemcomposer.createModel(modelName); % Create model

interface = arch.InterfaceDictionary.addInterface('interface'); % Add interface
elem = interface.addElement('x'); % Create interface element

setName(elem,'newName'); % Set new name for interface element
```

Input Arguments

`interfaceElem` — Interface element to be renamed

signal element object

Interface element to be renamed, specified as a `systemcomposer.interface.SignalElement` object.

`name` — Name of interface element

character vector

Name of interface element, specified as a character vector.

Data Types: `char`

See Also

`addElement` | `addInterface` | `createModel` | `systemcomposer.interface.SignalElement`

Topics

“Define Interfaces”

Introduced in R2019a

setName

Set name for port

Syntax

`setName(port,name)`

Description

`setName(port,name)` sets the name for the designated port.

Examples

Set New Name for Port

Create a model, get the root architecture, add a component, add a port, and set a new name for the port.

```
model = systemcomposer.createModel('archModel');
rootArch = get(model,'Architecture');
newcomponent = addComponent(rootArch,'NewComponent');
newport = addPort(newcomponent.Architecture,'NewCompPort','in');
setName(newport,'CompPort');
```

Input Arguments

`port` — Port to be renamed

port object

Port to be renamed, specified as a `systemcomposer.arch.ArchitecturePort` or `systemcomposer.arch.ComponentPort` object.

`name` — Name of port

character vector

Name of port, specified as a character vector.

Data Types: `char`

See Also

`systemcomposer.arch.ArchitecturePort` | `systemcomposer.arch.ComponentPort`

Introduced in R2019a

setInterface

Set interface for port

Syntax

`setInterface(port,interface)`

Description

`setInterface(port,interface)` sets the interface for a port.

Examples

Set Interface for Port

Create a model, get the root architecture, add a component, add a port, add an interface, and set the interface for the port.

```
model = systemcomposer.createModel('archModel');
rootArch = get(model,'Architecture');
newcomponent = addComponent(rootArch,'NewComponent');
newport = addPort(newcomponent.Architecture,'NewCompPort','in');
newinterface = addInterface(model.InterfaceDictionary,'NewInterface');
setInterface(newport,newinterface);
```

Input Arguments

`port` — Port to be edited

port object

Port to be edited, specified as a `systemcomposer.arch.ArchitecturePort` or `systemcomposer.arch.ComponentPort` object.

`interface` — Interface to set

signal interface object

Interface to set, specified as a `systemcomposer.interface.SignalInterface` object.

See Also

`systemcomposer.arch.ArchitecturePort` | `systemcomposer.arch.ComponentPort`

Topics

“Define Interfaces”

Introduced in R2019a

setProperty

Set property value corresponding to stereotype applied to element

Syntax

```
setProperty(element,propertyName,propertyValue,propertyUnits)
```

Description

`setProperty(element,propertyName,propertyValue,propertyUnits)` sets the value and units of the property specified in the `propertyName` argument. Set the property corresponding to an applied stereotype by qualified name `'<stereotype>.<property>'`. This is the verbose approach to setting a property.

Examples

Apply a Stereotype and Set Numeric Property Value

In this example, `weight` is a property of the stereotype `sysComponent`.

```
applyStereotype(element,'sysProfile.sysComponent')
setProperty(element,'sysComponent.weight','5','g')
```

Apply a Stereotype and Set String Property Value

In this example, `description` is a property of the stereotype `sysComponent`.

```
expression = sprintf("'%s'",'component description')
setProperty(element,'sysComponent.description',expression)
```

Input Arguments

`element` — Architecture model element
component object | port object | connector object

Architecture model element, specified as a `systemcomposer.arch.Architecture`, `systemcomposer.arch.Component`, `systemcomposer.arch.ComponentPort`, `systemcomposer.arch.ArchitecturePort`, `systemcomposer.arch.Connector`, or `systemcomposer.arch.Element` object.

`propertyName` — Name of the property
character vector

Qualified name of the property in the form `'<stereotype>.<property>'`.

Data Types: `char`

`propertyValue` — Value of property
character vector | numeric | enumeration

Specify numeric values in single quotes. Specify string values in the `sprintf("'%s'",'<property value>')` form. See example on this page.

Data Types: `char` | `double` | `enum`

`propertyUnits` — Units of property
character vector

Units of property to interpret property values, specified as a character vector.

Data Types: `char`

See Also

`getProperty` | `removeProperty`

Topics
“Set Tags and Properties for Analysis”

Introduced in R2019a

setType

Set type for signal interface element

Syntax

```
setType(interfaceElem,type)
```

Description

`setType(interfaceElem,type)` sets the type for the designated signal interface element.

Examples

Set Type for Interface Element

Create a model named `'archModel'`, add an interface, create an interface element with a name `'x'`, and set the type for the interface element `'single'`.

```
modelName = 'archModel';
arch = systemcomposer.createModel(modelName); % Create model

interface = arch.InterfaceDictionary.addInterface('interface'); % Add interface
elem = interface.addElement('x'); % Create interface element

setType(elem,'single'); % Set type for interface element
```

Input Arguments

`interfaceElem` — Interface element

signal element object

Interface element, specified as a `systemcomposer.interface.SignalElement` object.

`type` — Type of interface element

character vector

Type of interface element, specified as a character vector for a valid MATLAB data type.

Data Types: `char`

See Also

`addElement` | `addInterface` | `createModel` | `systemcomposer.interface.SignalElement`

Topics

“Define Interfaces”

Introduced in R2019a

setUnits

Set units for signal interface element

Syntax

```
setUnits(interfaceElem,units)
```

Description

`setUnits(interfaceElem,units)` sets the units for the designated signal interface element.

Examples

Set Units for Interface Element

Create a model named `'archModel'`, add an interface, create an interface element with a name `'x'`, and set the units for the interface element `'kg'`.

```
modelName = 'archModel';
arch = systemcomposer.createModel(modelName); % Create model

interface = arch.InterfaceDictionary.addInterface('interface'); % Add interface
elem = interface.addElement('x'); % Create interface element

setUnits(elem,'kg'); % Set units for interface element
```

Input Arguments

`interfaceElem` — Interface element

signal element object

Interface element, specified as a `systemcomposer.interface.SignalElement` object.

`units` — Units of interface element

character vector

Units of interface element, specified as a character vector.

Data Types: `char`

See Also

`addElement` | `addInterface` | `createModel` | `systemcomposer.interface.SignalElement`

Topics

“Define Interfaces”

Introduced in R2019a

setValue

Set value of property for element instance

Syntax

```
setValue(instance,property,value)
```

Description

`setValue(instance,property,value)` sets the `property` of the `instance` to `value`. This function is part of the instance API that you can use to analyze the model iteratively, element by element.`instance` refers to the element instance on which the iteration is being performed.

Examples

Set the Weight Property

Assume that a `MechComponent` stereotype is attached to the specification of the instance.

```
setValue(instance,'MechComponent.weight',10);
```

Input Arguments

`instance` — Element instance
architecture instance | component instance | port instance | connector instance

Element instance, specified by a `systemcomposer.analysis.ArchitectureInstance`, `systemcomposer.analysis.ComponentInstance`, `systemcomposer.analysis.PortInstance`, or `systemcomposer.analysis.ConnectorInstance` object. This function is part of the instance API that you can use to analyze the model iteratively, element by element.`instance` refers to the element instance on which the iteration is being performed.

`property` — Property
character vector

Property, specified as a character vector in the form `'<stereotype>.<property>'`.

`value` — Property value
`double` (default) | `single` | `int64` | `int32` | `int16` | `int8` | `uint64` | `uint32` | `uint8` | `boolean` | `string` | enumeration class name

Property value, specified as a data type that depends on how the property is defined in the profile.

See Also

`getValue` | `systemcomposer.analysis.Instance`

Topics
“Write Analysis Function”

Introduced in R2019a

unlinkDictionary

Unlink data dictionary from architecture model

Syntax

```
unlinkDictionary(modelObject)
```

Description

`unlinkDictionary(modelObject)` removes the association of the model from its data dictionary.

Examples

Unlink the Data Dictionary

```
unlinkDictionary(arch);
```

Input Arguments

`modelObject` — Architecture model object

model object

Architecture model object from which the dictionary link is to be removed, specified as a `systemcomposer.arch.Model` object.

See Also

`linkDictionary` | `systemcomposer.createDictionary` | `systemcomposer.openDictionary`

Topics

“Save, Link, and Delete Interfaces”

Introduced in R2019a

updateInstance

Update architecture instance

Syntax

```
updateInstance(architectureInstance,updateFlag)
```

Description

`updateInstance(architectureInstance,updateFlag)` updates an instance to mirror the changes in the specification model.

Input Arguments

`architectureInstance` — Architecture instance
instance object

Architecture instance to be updated, specified as a `systemcomposer.analysis.ArchitectureInstance` object.

`updateFlag` — Whether to update values changed directly in model
`true` | `false`

If `true`, the method reflects changes made directly in the specification model to the instance model.

Data Types: `logical`

See Also

`deleteInstance` | `instantiate` | `loadInstance` | `saveInstance` | `systemcomposer.analysis.Instance`

Topics
“Write Analysis Function”

Introduced in R2019a

Classes

systemcomposer.allocation.AllocationSet

Manage set of allocation scenarios

Description

The `AllocationSet` defines a collection of allocation scenarios between two models.

Creation

```
% Create the allocation set with name MyNewallocation.
systemcomposer.allocation.createAllocationSet('MyNewallocation', ...
     'Source_Model_Allocation', 'Target_Model_Allocation');

% Open the allocation editor
systemcomposer.allocation.editor()
```

Properties

`Name` — Name of allocation set
character vector

Name of allocation set, returned as a character vector.

Data Types: `char`

`SourceModel` — Source model for allocation
model object | character vector

Source model for allocation, returned as a `systemcomposer.arch.Model` object or the name of a model as a character vector.

`TargetModel` — Target model for allocation
model object | character vector

Target model for allocation, returned as a `systemcomposer.arch.Model` object or the name of a model as a character vector.

`Scenarios` — Allocation scenarios
cell array of allocation scenario objects

Allocation scenarios, returned as a cell array of `systemcomposer.allocation.AllocationScenario` objects.

`NeedsRefresh` — Indicates if allocation set is out of date
`true` or 1 | `false` or 0

Indicates if allocation set is out of date with the source and/or target model, returned as a logical or numeric with values 1 (`true`) or 0 (`false`).

Data Types: `logical`

`Dirty` — Indicates if allocation has unsaved changes
`true` or 1 | `false` or 0

Indicates if the allocation set has unsaved changes, returned as a logical or numeric with values 1 (`true`) or 0 (`false`).

Data Types: `logical`

Object Functions

close	Close allocation set
closeAll	Close all loaded allocation sets
createScenario	Create new empty allocation scenario
deleteScenario	Delete allocation scenario
find	Find loaded allocation set
getScenario	Get allocation scenario
save	Save allocation set

See Also

`createAllocationSet` | `systemcomposer.allocation.Allocation` | `systemcomposer.allocation.AllocationScenario` | `systemcomposer.allocation.editor`

Topics
“Create and Manage Allocations”

Introduced in R2020b

systemcomposer.analysis.Instance

Class that represents architecture model element in analysis instance

Description

The `Instance` class represents an instance of an architecture.

Creation

Create an instance of an architecture

```
instance = instantiate(modelHandle,architecture,properties,name)
```

Properties

`Name` — Name of instance
character vector

Name of instance, returned as a character vector.

Data Types: `char`

`Specification` — Specification for creating instance
architecture | component | port | connector

Specification for creating instance, returned as a `systemcomposer.arch.Architecture`, `systemcomposer.arch.Component`, `systemcomposer.arch.ArchitecturePort`, `systemcomposer.arch.ComponentPort`, or `systemcomposer.arch.Connector` object. The specification depends on the kind of instance.

Architecture Instance Properties

`Components` — Child components of instance
array of components

Child components of instance, returned as an array of `systemcomposer.analysis.ComponentInstance` objects.

`Ports` — Ports of architecture instance
array of ports

Ports of architecture instance, returned as an array of `systemcomposer.analysis.PortInstance` objects.

`Connectors` — Connectors in architecture instance
array of connectors

Connectors in architecture instance, returned as an array of `systemcomposer.analysis.ConnectorInstance` objects, connecting child components.

`Specification` — References element in model
architecture object

References element in model, returned as a `systemcomposer.analysis.ArchitectureInstance` object.

Component Instance Properties

`Components` — Child components of instance
array of components

Child components of instance, returned as an array of `systemcomposer.analysis.ComponentInstance` objects within the architecture.

`Ports` — Ports of component instance
array of ports

Ports of component instance, returned as an array of `systemcomposer.analysis.PortInstance` objects.

`Connectors` — Connectors in component instance
array of connectors

Connectors in component instance, connecting child components, returned as an array of `systemcomposer.analysis.ConnectorInstance` objects.

`Parent` — Parent of the component
architecture object

Parent of the component, returned as a `systemcomposer.analysis.ArchitectureInstance` object.

`Specification` — References element in model
architecture object

References element in model, returned as a `systemcomposer.analysis.ArchitectureInstance` object.

Port Instance Properties

`Parent` — Component that contains the port
component instance object

Component that contains the port, returned as a `systemcomposer.analysis.ComponentInstance` object.

Connector Instance Properties

`Parent` — Component that contains connector
component instance object

Component that contains connector, returned as a `systemcomposer.analysis.ComponentInstance` object.

`SourcePort` — Source port instance
port instance object

Source port instance, returned as a `systemcomposer.analysis.PortInstance` object.

`DestinationPort` — Destination port instance
port instance object

Destination port instance, returned as a `systemcomposer.analysis.PortInstance` object.

`Specification` — References element in model
architecture object

References element in model, returned as a `systemcomposer.analysis.ArchitectureInstance` object.

Object Functions

getValue	Get value of property from element instance
setValue	Set value of property for element instance
isArchitecture	Find if instance is architecture instance
isComponent	Find if instance is component instance
isConnector	Find if instance is connector instance
isPort	Find if instance is port instance

See Also

`deleteInstance` | `instantiate` | `loadInstance` | `saveInstance` | `updateInstance`

Topics
“Write Analysis Function”

Introduced in R2019a

systemcomposer.arch.Architecture

Class that represents architecture in architecture model

Description

The `Architecture` class represents an architecture in the model. This class inherits from `systemcomposer.base.BaseElement` and implements the interface `systemcomposer.base.BaseArchitecture`.

Creation

Create a model and get the root architecture:

```
model = systemcomposer.createModel('archModel');
arch = get(model,'Architecture')
```

Properties

`Name` — Name of architecture
character vector

Name of architecture, returned as a character vector. The architecture name is derived from the parent component or model name to which the architecture belongs.

Example: `'system_architecture'`

Data Types: `char`

`Definition` — Definition type of architecture
composition | behavior | view

Definition type of architecture, returned as a composition, a behavior model, or a view.

Data Types: `ArchitectureDefinition enum`

`Parent` — Handle to parent component
component object

Handle to parent component that owns architecture, returned as a `systemcomposer.arch.Component` object.

`Components` — Array of handles to set of child components
array of component objects

Array of handles to set of child components of architecture, returned as an array of `systemcomposer.arch.Component` objects.

`Ports` — Array of architecture ports
array of architecture port objects

Array of architecture ports of architecture, returned as an array of `systemcomposer.arch.ArchitecturePort` objects.

`Connectors` — Array of connectors that connect child components of this architecture
array of connector objects

Array of connectors that connect child components of this architecture, returned as an array of `systemcomposer.arch.Connector` objects.

Object Functions

addComponent	Add components to architecture
addVariantComponent	Add variant components to architecture
addPort	Add ports to architecture
connect	Create architecture model connections
applyStereotype	Apply stereotype to architecture model element
getStereotypes	Get stereotypes applied on element of architecture model
removeStereotype	Remove stereotype from model element
batchApplyStereotype	Apply stereotype to all elements in specified architecture
iterate	Iterate over model elements
instantiate	Create analysis instance from specification
setProperty	Set property value corresponding to stereotype applied to element
getProperty	Get property value corresponding to stereotype applied to element
removeProfile	Remove profile from model
applyProfile	Apply profile to a model
getEvaluatedPropertyValue	Get evaluated value of property from component

See Also

`systemcomposer.arch.Component` | `systemcomposer.arch.Element`

Topics
“Create an Architecture Model”

Introduced in R2019a

systemcomposer.arch.ArchitecturePort

Represent input and output ports of architecture

Description

This class inherits from `systemcomposer.arch.BasePort`.

Creation

```
port = addPort(archObj,'in')
```

The `addPort` method is the constructor for the `systemcomposer.arch.ArchitecturePort` class.

Properties

`Name` — Name of port
character vector

Name of port, returned as a character vector.

Data Types: `char`

`Direction` — Port direction
`'Input'` | `'Output'`

Port direction, returned as a character array with values `'Input'` and `'Output'`.

Data Types: `char`

`InterfaceName` — Name of interface associated with port
character vector

Name of interface associated with port, returned as a character vector.

Data Types: `char`

`Interface` — Interface associated with port
signal interface object

Interface associated with port, returned as a `systemcomposer.interface.SignalInterface` object.

`Connectors` — Port connectors
connector object

Port connectors, returned as a `systemcomposer.arch.Connector` object.

`Connected` — Whether port has connections
`true` or 1 | `false` or 0

Whether port has connections, returned as a logical or numeric value 1 (`true`) or 0 (`false`).

Data Types: `logical`

`Parent` — Architecture that owns port
architecture object

Architecture that owns port, returned as a `systemcomposer.arch.Architecture` object.

Object Functions

connect	Create architecture model connections
setName	Set name for port
setInterface	Set interface for port
createAnonymousInterface	Create and set anonymous interface for port
applyStereotype	Apply stereotype to architecture model element
getStereotypes	Get stereotypes applied on element of architecture model
removeStereotype	Remove stereotype from model element
setProperty	Set property value corresponding to stereotype applied to element
getProperty	Get property value corresponding to stereotype applied to element
destroy	Remove and destroy model element
getEvaluatedPropertyValue	Get evaluated value of property from component

See Also

`addPort` | `systemcomposer.arch.BasePort` | `systemcomposer.arch.ComponentPort` | `systemcomposer.arch.Element`

Topics
“Create an Architecture Model”

Introduced in R2019a

systemcomposer.arch.BaseComponent

Common base class for all components in architecture model

Description

A `systemcomposer.arch.BaseComponent` cannot be constructed. Either create a `systemcomposer.arch.Component` or `systemcomposer.arch.VariantComponent`.

Properties

`Parent` — Architecture that owns component
architecture object

Architecture that owns component, returned as a `systemcomposer.arch.Architecture` object.

`Ports` — Input and output ports of component
component port object

Input and output ports of component, returned as a `systemcomposer.arch.ComponentPort` object.

`OwnedArchitecture` — Architecture owned by component
architecture object

Architecture owned by component, returned as a `systemcomposer.arch.Architecture` object.

`Position` — Position of component on canvas
vector of coordinates in pixels

Position of component on canvas, returned as a vector of coordinates, in pixels `[left top right bottom]`.

Object Functions

getStereotypes	Get stereotypes applied on element of architecture model
getProperty	Get property value corresponding to stereotype applied to element
setProperty	Set property value corresponding to stereotype applied to element
getEvaluatedPropertyValue	Get evaluated value of property from component
getPort	Get object for signal interface element
applyStereotype	Apply stereotype to architecture model element
connect	Create architecture model connections
destroy	Remove and destroy model element
isReference	Find if component is reference to another model
removeStereotype	Remove stereotype from model element

See Also

Introduced in R2019b

systemcomposer.arch.BasePort

Common base class for all ports in architecture model

Description

A `systemcomposer.arch.Baseport` cannot be constructed. Create a `systemcomposer.arch.ArchitecturePort`.

Properties

`Name` — Name of port
character vector

Name of port, returned as a character vector.

Data Types: `char`

`Direction` — Port direction
`'Input'` | `'Output'`

Port direction, returned as a character array with values `'Input'` and `'Output'`.

Data Types: `char`

`InterfaceName` — Name of interface associated with port
character vector

Name of interface associated with port, returned as a character vector.

Data Types: `char`

`Interface` — Interface associated with port
signal interface object

Interface associated with port, returned as a `systemcomposer.interface.SignalInterface` object.

`Connectors` — Port connectors
connector object

Port connectors, returned as a `systemcomposer.arch.Connector` object.

`Connected` — Whether port has connections
`true` or 1 | `false` or 0

Whether port has connections, returned as a logical or numeric value 1 (`true`) or 0 (`false`).

Data Types: `logical`

Object Functions

getProperty	Get property value corresponding to stereotype applied to element

setProperty	Set property value corresponding to stereotype applied to element
getEvaluatedPropertyValue	Get evaluated value of property from component
applyStereotype	Apply stereotype to architecture model element
getStereotypes	Get stereotypes applied on element of architecture model
removeStereotype	Remove stereotype from model element
destroy	Remove and destroy model element

See Also

`systemcomposer.arch.ArchitecturePort` | `systemcomposer.arch.ComponentPort` | `systemcomposer.arch.Element`

Topics

“Ports”

Introduced in R2019a

systemcomposer.arch.Component

Class that represents component or view component

Description

The `Component` class represents a component in the architecture model. This class inherits from `systemcomposer.arch.BaseComponent`.

Creation

Create a component in an architecture model:

```
model = systemcomposer.createModel('archModel');
arch = get(model,'Architecture');
component = addComponent(arch,'NewComponent');
```

Properties

`Name` — Name of component
character vector

Name of component, returned as a character vector.

Data Types: `char`

`Parent` — Handle to parent architecture that owns component
architecture object

Handle to parent architecture that owns component, returned as a `systemcomposer.arch.Architecture` object.

`Architecture` — Architecture that defines component structure
architecture object

Architecture that defines component structure, returned as a `systemcomposer.arch.Architecture` object. For a component that references a different architecture model, this property returns a handle to the root architecture of that model. For variant components, the architecture is that of the active variant.

`OwnedArchitecture` — Architecture that component owns
architecture object

Architecture that component owns, returned as a `systemcomposer.arch.Architecture` object. For components that reference an architecture, this property is empty. For variant components, this property is the architecture in which the individual variant components reside.

`Ports` — Array of component ports
array of component port objects

Array of component ports, returned as an array of `systemcomposer.arch.ComponentPort` objects.

`OwnedPorts` — Array of component ports
array of component port objects

Array of component ports, returned as an array of `systemcomposer.arch.ComponentPort` objects. For reference components, this property is empty.

`ReferenceName` — If linked component, name of model that component references
character vector

If linked component, name of model that component references, returned as a character vector.

Data Types: `char`

Object Functions

saveAsModel	Save architecture to separate model
createSimulinkBehavior	Create Simulink model and link component to it
linkToModel	Link component to a model
inlineComponent	Inline reference architecture into model
makeVariant	Convert component to variant choice
isReference	Find if component is reference to another model
connect	Create architecture model connections
applyStereotype	Apply stereotype to architecture model element
getStereotypes	Get stereotypes applied on element of architecture model
removeStereotype	Remove stereotype from model element
setProperty	Set property value corresponding to stereotype applied to element
getProperty	Get property value corresponding to stereotype applied to element
destroy	Remove and destroy model element
getPort	Get port from component
getEvaluatedPropertyValue	Get evaluated value of property from component

See Also

`addComponent` | `createModel` | `systemcomposer.arch.Architecture` | `systemcomposer.arch.Element`

Topics
“Create an Architecture Model”

Introduced in R2019a

systemcomposer.arch.ComponentPort

Represents input and output ports of component

Description

This class inherits from `systemcomposer.arch.BasePort`.

Creation

A component port is constructed by creating an architecture port on the architecture of the component.

```
addPort(compObj.Architecture,portName,'in')
```

```
compPortObj = getPort(compObj,portName)
```

Properties

`Name` — Name of port
character vector

Name of port, returned as a character vector.

Data Types: `char`

`Direction` — Port direction
`'Input'` | `'Output'`

Port direction, returned as a character array with values `'Input'` and `'Output'`.

Data Types: `char`

`InterfaceName` — Name of interface
character vector

Name of interface associated with port, returned as a character vector.

Data Types: `char`

`Interface` — Interface associated with port
signal interface object

Interface associated with port, returned as a `systemcomposer.interface.SignalInterface` object.

`Connectors` — Port connectors
connector object

Port connectors, returned as a `systemcomposer.arch.Connector` object.

`Connected` — Whether port has connections
`true` or 1 | `false` or 0

Whether port has connections, returned as a logical or numeric value 1 (`true`) or 0 (`false`).

Data Types: `logical`

`Parent` — Component that owns port
architecture object

Component that owns port, returned as a `systemcomposer.arch.Architecture` object.

`ArchitecturePort` — Architecture port
architecture port object

Architecture port within the component that maps to port, returned as a `systemcomposer.arch.ArchitecturePort` object.

Object Functions

connect	Create architecture model connections
setName	Set name for port
setInterface	Set interface for port
createAnonymousInterface	Create and set anonymous interface for port
applyStereotype	Apply stereotype to architecture model element
getStereotypes	Get stereotypes applied on element of architecture model
removeStereotype	Remove stereotype from model element
setProperty	Set property value corresponding to stereotype applied to element
getProperty	Get property value corresponding to stereotype applied to element
destroy	Remove and destroy model element
getEvaluatedPropertyValue	Get evaluated value of property from component

See Also

`addPort` | `getPort` | `systemcomposer.arch.ArchitecturePort` | `systemcomposer.arch.BasePort` | `systemcomposer.arch.Element`

Introduced in R2019a

systemcomposer.arch.Connector

Class that represents connector between ports

Description

The `connector` class represents a connector between ports. This class is derived from `systemcomposer.arch.Element`. This class inherits from `systemcomposer.base.BaseElement` and implements the interface `systemcomposer.base.BaseConnector`.

Creation

Create a connector.

```
connector = connect(architecture,outports,inports)
```

Properties

`Parent` — Handle to parent architecture that owns connector
architecture object

Handle to parent architecture that owns connector, returned as a `systemcomposer.arch.Architecture` object.

`Name` — Name of connector
character vector

Name of connector, returned as a character vector.

Data Types: `char`

`SourcePort` — Source of connection
architecture port object | component port object

Source of connection as an output port, returned as a `systemcomposer.arch.ArchitecturePort` or `systemcomposer.arch.ComponentPort` object.

`DestinationPort` — Destination of connection
architecture port object | component port object

Destination of connection as an input port, returned as a `systemcomposer.arch.ArchitecturePort` or `systemcomposer.arch.ComponentPort` object.

Object Functions

applyStereotype	Apply stereotype to architecture model element
getStereotypes	Get stereotypes applied on element of architecture model
removeStereotype	Remove stereotype from model element
setProperty	Set property value corresponding to stereotype applied to element

getProperty	Get property value corresponding to stereotype applied to element
destroy	Remove and destroy model element
getEvaluatedPropertyValue	Get evaluated value of property from component
getSourceElement	Gets signal elements selected on source port for connection
getDestinationElement	Gets signal elements selected on destination port for connection

See Also

`connect` | `systemcomposer.arch.Element`

Topics

“Create an Architecture Model”

Introduced in R2019a

systemcomposer.arch.Element

Base class of all model elements

Description

The `Element` class is the base class for all System Composer model elements — architecture, component, port, and connector. This class inherits from `systemcomposer.base.BaseElement`.

Creation

Create an architecture, component, port, or connector: `addComponent`, `addPort`, `connect`.

Properties

`UUID` — Universal unique identifier for model element
character vector

Universal unique identifier for model element, returned as a character vector.

Example: '91d5de2c-b14c-4c76-a5d6-5dd0037c52df'

Data Types: `char`

`ExternalUID` — Unique external identifier
character vector

Unique external identifier, returned as a character vector. The external ID is preserved over the lifespan of the element and through all operations that preserve the UUID.

Example: 'network_connector_01'

Data Types: `char`

`Model` — Handle to parent System Composer model of element
model object

Handle to parent model of element, returned as a `systemcomposer.arch.Model` object.

`SimulinkHandle` — Simulink handle for element
numeric value

Simulink handle for element, returned as a numeric value. This property is necessary for several Simulink related work flows and for using Simulink Requirement APIs.

Example: `handle = get(object,'SimulinkHandle')`

Data Types: `double`

Object Functions

applyStereotype	Apply stereotype to architecture model element

getStereotypes	Get stereotypes applied on element of architecture model
removeStereotype	Remove stereotype from model element
setProperty	Set property value corresponding to stereotype applied to element
getProperty	Get property value corresponding to stereotype applied to element
destroy	Remove and destroy model element
getEvaluatedPropertyValue	Get evaluated value of property from component

See Also

`systemcomposer.arch.Architecture` | `systemcomposer.arch.ArchitecturePort` | `systemcomposer.arch.BasePort` | `systemcomposer.arch.Component` | `systemcomposer.arch.ComponentPort` | `systemcomposer.arch.Connector`

Topics

“Create an Architecture Model”

Introduced in R2019a

systemcomposer.arch.Model

Represent System Composer model

Description

Use the `Model` class to create and manage architecture objects in a System Composer model.

Creation

```
objModel = systemcomposer.createModel(modelName)
```

The `createModel` method is the constructor for the `systemcomposer.arch.Model` class.

Properties

`Name` — Name of model
character vector

Name of model, returned as a character vector.

Data Types: `char`

`Architecture` — Root architecture of model
architecture object

Root architecture of model, returned as a `systemcomposer.arch.Architecture` object.

`SimulinkHandle` — Simulink handle
numeric value

Simulink handle, returned as a numeric value.

Data Types: `double`

`Profiles` — Array of handles to profiles
array of profile objects

Array of handles to profiles attached to the model, returned as `systemcomposer.profile.Profile` objects.

`InterfaceDictionary` — Dictionary object that holds interfaces
dictionary object

Dictionary object that holds interfaces, returned as a `systemcomposer.interface.Dictionary` object. If the model is not linked to an external dictionary, this is a handle to the implicit dictionary

`Views` — Array of handles to model views
array of view architecture objects

Array of handles to model views, returned as an array of `systemcomposer.view.ViewArchitecture` objects.

Example: `objViewArchitecture = get(objModel,'Views')`

Object Functions

open	Open architecture model
close	Close System Composer model
save	Save the architecture model or data dictionary
find	Find architecture elements using query
lookup	Search for architecture element
createViewArchitecture	Create view
openViews	Open architecture views editor
applyProfile	Apply profile to a model
removeProfile	Remove profile from model
linkDictionary	Link data dictionary to architecture model
unlinkDictionary	Unlink data dictionary from architecture model
renameProfile	Rename profile in model
iterate	Iterate over model elements

See Also

Topics

“Create an Architecture Model”

Introduced in R2019a

systemcomposer.arch.VariantComponent

Represent variant component in System Composer model

Description

This class inherits from `systemcomposer.arch.BaseComponent`. A variant component allows you to create multiple design alternatives for a component.

Creation

```
varComp = addVariantComponent(archObj,compName)
```

The `addVariantComponent` method creates an object method on the `systemcomposer.arch.Architecture` class, and then creates a `systemcomposer.arch.VariantComponent` object.

Properties

`Parent` — Architecture that owns variant component
architecture object

Architecture that owns variant component, returned as a `systemcomposer.arch.Architecture` object.

`Ports` — Input and output ports
component port objects

Input and output ports of variant component, returned as `systemcomposer.arch.ComponentPort` objects.

`OwnedArchitecture` — Architecture owned by variant component
architecture object

Architecture owned by variant component, returned as a `systemcomposer.arch.Architecture` object.

`Architecture` — Architecture of active variant choice
architecture object

Architecture of the active variant choice, returned as a `systemcomposer.arch.Architecture` object.

Object Functions

addChoice	Add variant choices to variant component
setCondition	Set condition on variant choice
setActiveChoice	Set active choice on variant component
getChoices	Get available choices in variant component

getActiveChoice	Get active choice on variant component
getCondition	Return variant control on choice within variant component
getStereotypes	Get stereotypes applied on element of architecture model
removeStereotype	Remove stereotype from model element
applyStereotype	Apply stereotype to architecture model element
destroy	Remove and destroy model element
getEvaluatedPropertyValue	Get evaluated value of property from component
getPort	Get port from component
getProperty	Get property value corresponding to stereotype applied to element
setProperty	Set property value corresponding to stereotype applied to element
isReference	Find if component is reference to another model

See Also

Topics
“Decompose and Reuse Components”

Introduced in R2019a

systemcomposer.interface.Dictionary

Class that represents an element in the signal interface

Description

The `systemcomposer.interface.Dictionary` class represents the interface dictionary of an architecture model.

Creation

Create a dictionary.

```
dict_id = systemcomposer.createDictionary('NewDictionary');
```

Properties

`Interfaces` — Interfaces defined in dictionary
array of signal interfaces

Interfaces defined in dictionary, returned as an array of `systemcomposer.interface.SignalInterface` objects.

`UUID` — Universal unique identifier
character vector

Universal unique identifier for an interface dictionary, returned as a character vector.

Example: '91d5de2c-b14c-4c76-a5d6-5dd0037c52df'

Data Types: `char`

Object Functions

addInterface	Create named interface in interface dictionary
save	Save the architecture model or data dictionary
applyProfile	Apply profile to a model
removeProfile	Remove profile from model
removeInterface	Remove named interface from interface dictionary
getInterface	Get object for named interface in interface dictionary
getInterfaceNames	Get names of all interfaces in interface dictionary
destroy	Remove and destroy model element

See Also

`systemcomposer.createDictionary` | `systemcomposer.interface.SignalElement` | `systemcomposer.interface.SignalInterface` | `systemcomposer.openDictionary`

Topics
“Define Interfaces”

Introduced in R2019a

systemcomposer.interface.SignalElement

Class that represents element in signal interface

Description

The `SignalElement` class represents a single element in the signal interface.

Creation

Create a signal element.

```
elem = addElement(interface,'NewElement')
```

Properties

`Interface` — Handle to parent interface of element
signal interface object

Handle to parent interface of element, returned as a `systemcomposer.interface.SignalInterface` object.

`Name` — Element name
character vector

Element name, returned as a character vector.

Data Types: `char`

`Dimensions` — Dimensions of element
array of positive integers

Dimensions of element, returned as an array of positive integers.

Data Types: `integer`

`Type` — Data type of element
character vector

Data type of element, returned as a character vector.

Data Types: `char`

`Complexity` — Complexity of element
`'real'` | `'complex'`

Complexity of element, returned as `'real'` or `'complex'` character vectors.

Data Types: `char`

`Units` — Units of element
character vector

Units of element, returned as a character vector.

Data Types: `char`

`Minimum` — Minimum value for element
double

Minimum value for element, returned as a `double`.

Data Types: `double`

`Maximum` — Maximum value for element
double

Maximum value for element, returned as a `double`.

Data Types: `double`

`Description` — Description text for element
character vector

Description text for element, returned as a character vector.

Data Types: `char`

Object Functions

setName	Set name for signal interface element
setType	Set type for signal interface element
setDimensions	Set dimensions for signal interface element
setUnits	Set units for signal interface element
setComplexity	Set complexity for signal interface element
setMinimum	Set minimum for signal interface element
setMaximum	Set maximum for signal interface element
setDescription	Set description for signal interface element
destroy	Remove and destroy model element

See Also

`addElement` | `addInterface` | `getElement` | `getInterface` | `getInterfaceNames` | `removeElement` | `removeInterface` | `systemcomposer.interface.SignalInterface`

Topics
“Define Interfaces”

Introduced in R2019a

systemcomposer.interface.SignalInterface

Class that represents structure of signal interface

Description

The `SignalInterface` class represents the structure of the signal interface at a given port.

Creation

Create an interface.

`interface = addInterface(dictionary,name)`

Properties

`Dictionary` — Handle to parent dictionary of interface
interface dictionary object

Handle to parent dictionary of interface, returned as a `systemcomposer.interface.Dictionary` object.

`Name` — Interface name
character vector

Interface name, returned as a character vector.

Data Types: `char`

`Elements` — Elements in interface
array of interface element objects

Elements in interface, returned as an array of `systemcomposer.interface.SignalElement` objects.

Object Functions

addElement	Add signal interface element
removeElement	Remove a signal interface element
getElement	Get object for signal interface element
destroy	Remove and destroy model element
applyStereotype	Apply stereotype to architecture model element
removeStereotype	Remove stereotype from model element
getStereotypes	Get stereotypes applied on element of architecture model
getProperty	Get property value corresponding to stereotype applied to element
setProperty	Set property value corresponding to stereotype applied to element
getEvaluatedPropertyValue	Get evaluated value of property from component

See Also

`addInterface` | `systemcomposer.interface.SignalElement`

Topics
“Define Interfaces”

Introduced in R2019a

systemcomposer.io.ModelBuilder

Model builder for System Composer architecture models

Description

Build System Composer models using the model builder utility class. Build System Composer models with these sets of information: components and their position in architecture hierarchy, ports and their mappings to components, connections between the components through ports, and interfaces in architecture models and their mappings to ports.

Creation

Syntax

```
builder = systemcomposer.io.ModelBuilder(profile)
```

Description

`builder = systemcomposer.io.ModelBuilder(profile)` creates the `ModelBuilder` object.

Input Arguments

`profile` — Metadata XML file
character vector

File that contains a set of properties for any model element.

Output Arguments

`builder` — Model builder instantiation
`ModelBuilder` object

`ModelBuilder` object used to build a System Composer model.

Properties

`Components` — Component information
table

Table containing the hierarchical information of components, type of component (for example, reference, variant, or adapter), stereotypes applied on component, and ability to set property values of component.

`Ports` — Ports information
table

Table containing the information about ports, their mappings to components and interfaces, as well as stereotypes applied on them.

`Connections` — Connections information
table

Table containing information about the connections between the ports defined in ports table also stereotypes applied on connections.

`Interfaces` — Interfaces information
table

Table containing the definitions of various interfaces and their elements.

Utility Functions

Components	Description
`addComponent(compName, ID, ParentID)`	Add component with name and ID as a child of component with ID as `ParentID`. In case of root, `ParentID` is 0.
`setComponentProperty(ID, varargin)`	Set stereotype on component with ID. Key value pair of property name and value defined in the stereotype can be passed as input. In this example `builder.setComponentProperty(ID, 'StereotypeName',... 'UAVComponent.PartDescriptor','ModelName',kind,'Manufacturer',doma` `ModelName` and `Manufacturer` are properties under stereotype `PartDescriptor`.

Ports	Description
`addPort(portName, direction, ID, compID)`	Add port with name and ID with direction (either Input or Output) to component with ID as `compID`.
`setPropertyOnPort(ID, varargin)`	Set stereotype on port with ID. Key value pair of the property name and the value defined in the stereotype can be passed as input.

Connections	Description
`addConnection(connName, ID, sourcePortID,destPortID)`	Add connection with name and ID between ports with `sourcePortID` (direction: Output) and `destPortID` (direction: Input) defined in the ports table.
`setPropertyOnConnection(ID, varargin)`	Set stereotype on connection with ID. Key value pair of the property name and the value defined in the stereotype can be passed as input.

Interfaces	Description
`addInterface(interfaceName, ID)`	Add interface with name and ID to a data dictionary.

Interfaces	Description
`addElementInInterface(elementName, ID, interfaceID, datatype, dimensions, units, complexity, Maximum, Minimum)`	Add element with name and ID under an interface with ID as `interfaceID`. Data types, dimensions, units, complexity, and maximum and minimum are properties of an element. These properties are specified as strings.
`addAnonymousInterface(ID, datatype, dimensions, units, complexity, Maximum, Minimum)`	Add anonymous interface with ID and element properties like data type, dimensions, units, complexity, maximum and minimum. Data type of an anonymous interface cannot be another interface name. Anonymous interfaces do not have elements like other interfaces.

Interfaces and Ports	Description
`addInterfaceToPort(interfaceID, portID)`	Link an interface with ID specified as `InterfaceID` to a port with ID specified as `PortID`.

Models	Description
`build(modelName)`	Build model with model name passed as input.

Logging and Reporting	Description
`getImportErrorLog()`	Get `ErrorLogs` generated while importing the model . Called after the `build()` function
`getImportReport()`	Get a report of the import. Called after the `build()` function.

Examples

Import System Composer Architecture using Model Builder.

This example shows how to import architecture specifications into System Composer using the systemcomposer.io.modelBuilder() utility class. These architecture specifications can be defined in external source such as Excel file.

In system composer, an architecture is fully defined by three sets of information:

- Components and its position in architecture hierarchy
- Ports and its mapping to components
- Connections between the components through ports In this example, we also import interface data definitions from external source.
- Interfaces in architecture models and its mapping to ports

This example uses systemcomposer.modelBuilder class to pass all of the above architecture information and import a System Composer model.

In this example, architecture information of a small UAV system is defined in an Excel spreadsheet and is used to create a System Composer architecture model.

External Source Files

- Architecture.xlsx : This Excel file contains hierarchical information of the architecture model. This example maps the external source data to System Composer model elements. Below is the mapping of information in column names to System Composer model elements.

```
# Element    : Name of the element. Either can be component or port name.
# Parent     : Name of the parent element.
# Class      : Can be either component or port(Input/Output direction of the port).
# Domain     : Mapped as component property. Property "Manufacturer" defined in the
               profile UAVComponent under Stereotype PartDescriptor maps to Domain values ir
# Kind       : Mapped as component property. Property "ModelName" defined in the
               profile UAVComponent under Stereotype PartDescriptor maps to Kind values in e
# InterfaceName : If class is of port type. InterfaceName maps to name of the interface linl
# ConnectedTo : In case of port type, it specifies the connection to
                other port defined in format "ComponentName::PortName".
```

- DataDefinitions.xlsx : This excel file contains interface data definitions of the model. This example assumes the below mapping between the data definitions in the source excel file and interfaces hierarchy in System Composer :

```
# Name        : Name of the interface or element.
# Parent      : Name of the parent interface Name(Applicable only for elements) .
# Datatype    : Datatype of element. Can be another interface in format
                Bus: InterfaceName
# Dimensions  : Dimensions of the element.
# Units       : Unit property of the element.
# Minimum     : Minimum value of the element.
# Maximum     : Maximum value of the element.
```

Step 1. Instantiate the model builder class

You can instantiate the model builder class with a profile name.

Make sure the current directory is writable because this example will be creating files.

```
[stat, fa] = fileattrib(pwd);
if ~fa.UserWrite
    disp('This script must be run in a writable directory');
    return;
end
% Name of the model to build.
modelName = 'scExampleModelBuider';
% Name of the profile.
profile = 'UAVComponent';
% Name of the source file to read architecture information.
architectureFileName = 'Architecture.xlsx';

% Instantiate the ModelBuilder
builder = systemcomposer.io.ModelBuilder(profile);
```

Step 2. Build Interface Data Definitions.

Reading the information in external source file DataDefinitions.xlsx, we build the interface data model.

Create MATLAB tables from source Excel file.

```
opts = detectImportOptions('DataDefinitions.xlsx');
opts.DataRange = 'A2'; % force readtable to start reading from the second row.
definitionContents = readtable('DataDefinitions.xlsx', opts);

% systemcomposer.io.IdService class generates unique ID for a
% given key
idService = systemcomposer.io.IdService();

for rowItr =1:numel(definitionContents(:,1))
    parentInterface = definitionContents.Parent{rowItr};
    if isempty(parentInterface)
        % In case of interfaces adding the interface name to model builder.
        interfaceName = definitionContents.Name{rowItr};
        % Get unique interface ID. getID(container,key) generates
        % or returns(if key is already present) same value for input key
        % within the container.
        interfaceID = idService.getID('interfaces',interfaceName);
        % Builder utility function to add interface to data
        % dictionary.
        builder.addInterface(interfaceName,interfaceID);
    else
        % In case of element read element properties and add the element to
        % parent interface.
        elementName  = definitionContents.Name{rowItr};
        interfaceID = idService.getID('interfaces',parentInterface);
        % ElementID is unique within a interface.
        % Appending 'E' at start of ID for uniformity. The generated ID for
        % input element is unique within parent interface name as container.
        elemID = idService.getID(parentInterface,elementName,'E');
        % Datatype, dimensions, units, minimum and maximum properties of
        % element.
        datatype = definitionContents.DataType{rowItr};
        dimensions = string(definitionContents.Dimensions(rowItr));
        units = definitionContents.Units(rowItr);
        % Make sure that input to builder utility function is always a
        % string.
        if ~ischar(units)
            units = '';
        end
        minimum = definitionContents.Minimum{rowItr};
        maximum = definitionContents.Maximum{rowItr};
        % Builder function to add element with properties in interface.
        builder.addElementInInterface(elementName, elemID, interfaceID, datatype, dimensions, uni
    end
end
```

Step 3. Build Architecture Specifications.

Architecture specifications de Create MATLAB tables from source Excel file.

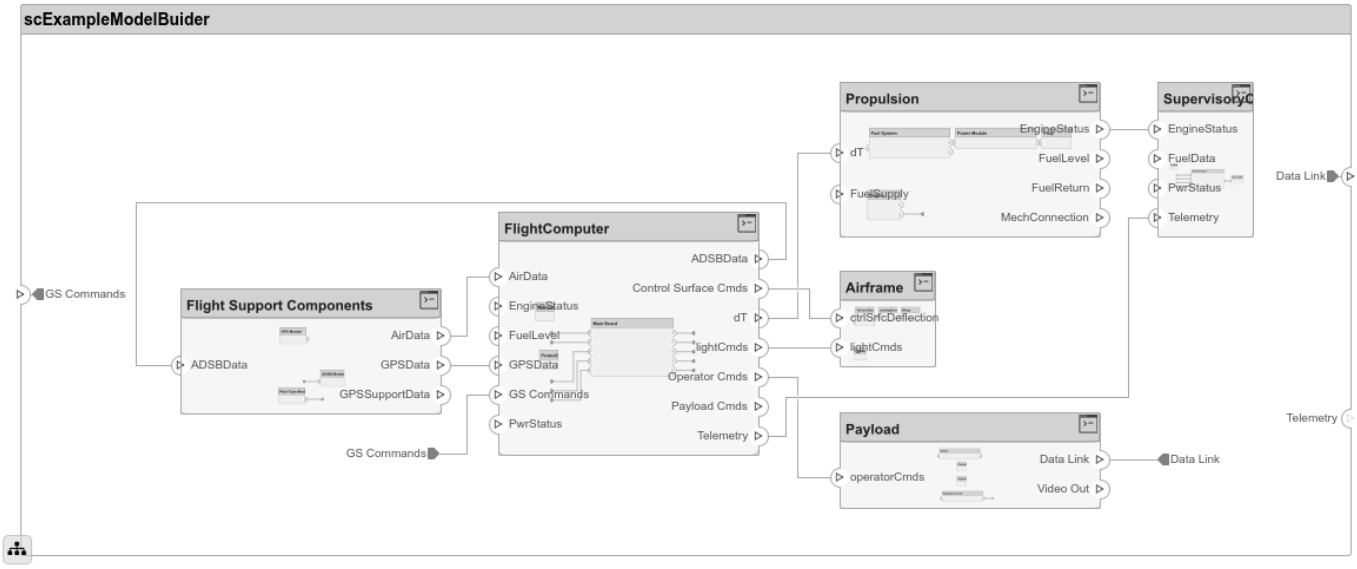
```
excelContents = readtable(architectureFileName);
% Iterate over each row in table.
for rowItr =1:numel(excelContents(:,1))
% Read each row of the excel file and columns.
    class = excelContents.Class(rowItr);
    Parent = excelContents.Parent(rowItr);
    Name = excelContents.Element{rowItr};
    % Populating the contents of table using the builder.
```

```
if strcmp(class,'component')
    ID = idService.getID('comp',Name);
    % Root ID is by default set as zero.
    if strcmp(Parent,'scExampleSmallUAV')
        parentID = "0";
    else
        parentID = idService.getID('comp', Parent);
    end
    % Builder utility function to add component.
    builder.addComponent(Name,ID,parentID);
    % Reading the property values
    kind = excelContents.Kind{rowItr};
    domain = excelContents.Domain{rowItr};
    % *Builder to set stereotype and property values*
    builder.setComponentProperty(ID, 'StereotypeName','UAVComponent.PartDescriptor','ModelNar
else
    % In this example, concatenation of port name and parent component name
    % is used as key to generate unique IDs for ports.
    portID = idService.getID('port',strcat(Name,Parent));
    % For ports on root architecture. compID is assumed as "0".
    if strcmp(Parent,'scExampleSmallUAV')
        compID = "0";
    else
        compID = idService.getID('comp',Parent);
    end
    % Builder utility function to add port.
    builder.addPort(Name, class, portID, compID );

    % InterfaceName specifies the name of the interface linked to port.
    interfaceName = excelContents.InterfaceName{rowItr};

    % Get interface ID. getID() will return the same IDs already
    % generated while adding interface in Step 2.
    interfaceID = idService.getID('interfaces',interfaceName);
    % Builder to map interface to port.
    builder.addInterfaceToPort(interfaceID, portID);

    % Reading the connectedTo information to build connections between
    % components.
    connectedTo = excelContents.ConnectedTo{rowItr};
    % connectedTo is in format -:
    % (DestinationComponentName::DestinationPortName).
    % For this example, considering the current port as source of the connection.
    if ~isempty(connectedTo)
        connID = idService.getID('connection',connectedTo);
        splits = split(connectedTo,'::');
        % Get the port ID of the connected port.
        % In this example, port ID is generated by concatenating
        % port name and parent component name. If port id is already
        % generated getID() function returns the same id for input key.
        connectedPortID = idService.getID('port',strcat(splits(2),splits(1)));
        % Using builder to populate connection table.
        sourcePortID = portID;
        destPortID = connectedPortID;
        % Builder to add connections.
        builder.addConnection(connectedTo,connID,sourcePortID,destPortID);
    end
```

```
    end
end
```

Step 3. Builder build method imports model from populated tables.

```
[model, importReport] = builder.build(modelName);
```



Close Model

```
bdclose(modelName);
```

See Also

Topics
“Import and Export Architecture Models”

Introduced in R2019b

systemcomposer.profile.Profile

Class that represents a profile

Description

The `Profile` class represents architecture profiles.

Creation

```
profile = systemcomposer.profile.Profile.createProfile(profileName);
```

Properties

`Name` — Name of profile
character vector

Name of profile, returned as a character vector.

Data Types: `char`

`Description` — Description text for profile
character vector

Description text for profile, returned as a character vector.

Data Types: `char`

Object Functions

createProfile	Create profile
addStereotype	Add stereotype to profile
getStereotype	Find stereotype in profile by name
getDefaultStereotype	Get default stereotype for profile
setDefaultStereotype	Set default stereotype for profile
find	Find profile by name
open	Open profile
load	Load profile from file
save	Save profile as file
close	Close profile
closeAll	Close all open profiles
destroy	Remove and destroy model element

See Also

`loadProfile` | `systemcomposer.profile.Stereotype`

Topics
“Define Profiles and Stereotypes”

Introduced in R2019a

systemcomposer.profile.Property

Class that represents a property

Description

The `Property` class represents properties in a stereotype.

Creation

`addProperty(stereotype,AttributeName,AttributeValue)`

Properties

`Name` — Name of property
character vector

Name of property, returned as a character vector.

Data Types: `char`

`Type` — Property data type
character vector

Property data type, returned as a character array with a valid data type.

Data Types: `char`

`Dimensions` — Dimensions of property
positive integer array

Dimensions of property, returned as a positive integer array.

Data Types: `double`

`Min` — Minimum value
numeric

Minimum value, returned as a numeric value.

Data Types: `double`

`Max` — Maximum value
numeric

Maximum value, returned as a numeric value.

Data Types: `double`

`Units` — Property units
character vector

Property units, returned as a character vector.

Data Types: `char`

Object Functions

destroy Remove and destroy model element

See Also

`addProperty` | `removeProperty` | `systemcomposer.profile.Profile` | `systemcomposer.profile.Stereotype`

Topics

“Define Profiles and Stereotypes”

Introduced in R2019a

systemcomposer.profile.Stereotype

Class that represents a stereotype

Description

The `Stereotype` class represents architecture stereotypes in a profile.

Creation

`addStereotype(profile,name,type)`

Properties

`Name` — Name of stereotype
character vector

Name of stereotype, returned as a character vector.

Data Types: `char`

`Description` — Description text for stereotype
character vector

Description text for stereotype, returned as a character vector.

Data Types: `char`

`Icon` — Icon for stereotype
character vector

Icon for stereotype, returned as a character vector.

Data Types: `char`

`Parent` — Stereotype from which stereotype inherits properties
stereotype object

Stereotype from which stereotype inherits properties, returned as a `systemcomposer.profile.Stereotype` object.

`AppliesTo` — Element type to which stereotype can be applied
`'Component'` | `'Port'` | `'Connector'` | `'Interface'`

Element type to which stereotype can be applied, returned as a character vector of the following options: `'Component'`, `'Port'`, `'Connector'`, or `'Interface'`.

Data Types: `char`

`Abstract` — Whether stereotype is abstract
`true` or 1 | `false` or 0

Whether stereotype is abstract, returned as a logical of numeric 1 (`true`) or 0(`false`). If `true`, then stereotype cannot be directly applied on model elements, but instead serves as a parent for other stereotypes.

Data Types: `logical`

Object Functions

addProperty	Define a custom property for a stereotype
removeProperty	Remove property from stereotype
find	Find stereotype by name
setDefaultComponentStereotype	Set default stereotype for components
setDefaultConnectorStereotype	Set default stereotype for connectors
setDefaultPortStereotype	Set default stereotype for ports
destroy	Remove and destroy model element

See Also

`addStereotype` | `getStereotype` | `removeStereotype` | `systemcomposer.profile.Profile`

Topics

“Define Profiles and Stereotypes”

Introduced in R2019a

systemcomposer.query.Constraint

Represent query constraint

Description

The `systemcomposer.query.Constraint` class is a base class for all System Composer query constraints.

Object Functions

HasStereotype	Create query to select architecture elements with stereotype based on specified sub-constraint
Property	Create query to select non-evaluated values for properties or stereotype properties for objects based on specified property name
PropertyValue	Create query to select property from object or stereotype property and then evaluate property value
HasPort	Create query to select architecture elements with port on component based on specified sub-constraint
HasInterface	Create query to select architecture elements with interface on port based on specified sub-constraint
HasInterfaceElement	Create query to select architecture elements with interface element on interface based on specified sub-constraint
IsInRange	Create query to select a range of property values
AnyComponent	Create query to select all components in model
IsStereotypeDerivedFrom	Create query to select stereotype derived from a fully qualified name

See Also

`createViewArchitecture` | `find`

Topics

“Creating Architectural Views Programmatically”

Introduced in R2019b

systemcomposer.view.BaseViewComponent

Base class for view components

Description

This class inherits from `systemcomposer.view.ViewElement` and implements the interface `systemcomposer.base.BaseComponent`.

Properties

`Name` — Name of view component
character vector

Name of view component, returned as a character vector.

Example: `name = get(objBaseViewComponent,'Name')`

Example: `set(objBaseViewComponent,'Name',name)`

`Parent` — Handle to parent view architecture of component
view architecture object

Handle to the parent view architecture of component, returned as a `systemcomposer.view.ViewArchitecture` object.

Example: `parent = get(objBaseViewComponent,'Parent')`

`Architecture` — Handle to view architecture of component
view architecture object

Handle to the view architecture of component, returned as a `systemcomposer.view.ViewArchitecture` object.

Example: `viewArch = get(objBaseViewComponent,'ViewArchitecture')`

See Also

`systemcomposer.view.ComponentOccurrence` | `systemcomposer.view.ViewArchitecture` | `systemcomposer.view.ViewComponent` | `systemcomposer.view.ViewElement`

Topics
“Create Architecture Views Interactively”
“Creating Architectural Views Programmatically”

Introduced in R2019b

systemcomposer.view.ComponentOccurrence

Shadow of component from composition in view

Description

This class inherits from `systemcomposer.view.BaseViewComponent`.

Properties

`Component` — Handle to composition
base component object

Handle to composition component of this occurrence, returned as a `systemcomposer.arch.BaseComponent` object.

Example: `handle = get(object,'Component')`

See Also

`systemcomposer.view.BaseViewComponent` | `systemcomposer.view.ViewArchitecture` | `systemcomposer.view.ViewComponent` | `systemcomposer.view.ViewElement`

Topics
“Create Architecture Views Interactively”
“Creating Architectural Views Programmatically”

Introduced in R2019b

systemcomposer.view.ViewArchitecture

View components in architecture view

Description

A view architecture describes a set of view components that make up a view. This class inherits from the `systemcomposer.view.ViewElement` class and implements the `systemcomposer.base.BaseArchitecture` interface.

Properties

`Name` — Name of architecture
character vector

Name of architecture derived from the parent component or model name to which the architecture belongs, returned as a character vector.

Example: `name = get(objViewArchitecture,'Name')`

Data Types: `char`

`IncludeReferenceModels` — Control inclusion of referenced models
`true` or 1 | `false` or 0

Control inclusion of referenced models, returned as a numeric or logical with values 1 (`true`) or 0 (`false`).

Example: `included = get(objViewArchitecture,'IncludeReferenceModels')`

Data Types: `logical`

`Color` — Color of view architecture
character vector

Color of view architecture, returned as a character vector as a name `'blue'`, `'black'`, or `'green'` or as a RGB value encoded in a hexadecimal string `'#FF00FF'` or `'#DDDDDD'`. An invalid color string results in an error.

Example: `color = get(objViewArchitecture,'Color')`

`Description` — Description of view architecture
character vector

Description of view architecture, returned as a character vector.

Example: `description = get(objViewArchitecture,'Description')`

Example: `set(objViewArchitecture,'Description',description)`

Data Types: `char`

`Parent` — Component that owns view architecture
base view component object

Component that owns view architecture, returned as a `systemcomposer.view.BaseViewComponent` object. For a root view architecture, returns an empty handle.

Example: `parentComponent = get(objViewArchitecture,'Parent')`

`Components` — Array of handles to child components
array of base view component objects

Array of handles to the set of child components of this view architecture, returned as an array of `systemcomposer.view.BaseViewComponent` objects.

Example: `childComponents = get(objViewArchitecture,'Components')`

Methods

addComponent	Add component to view given path
removeComponent	Remove component from view
createViewComponent	Create new view component

See Also

`systemcomposer.view.BaseViewComponent` | `systemcomposer.view.ComponentOccurrence` | `systemcomposer.view.ViewComponent` | `systemcomposer.view.ViewElement`

Topics
“Create Architecture Views Interactively”
“Creating Architectural Views Programmatically”

Introduced in R2019b

systemcomposer.view.ViewComponent

View component within architecture view

Description

A view component is a component that exist only in the view it is created in. These components do not exist in the composition. This class inherits from `systemcomposer.view.BaseViewComponent`.

See Also

`systemcomposer.view.BaseViewComponent` | `systemcomposer.view.ComponentOccurrence` | `systemcomposer.view.ViewArchitecture` | `systemcomposer.view.ViewElement`

Topics

“Create Architecture Views Interactively”
“Creating Architectural Views Programmatically”

Introduced in R2019b

systemcomposer.view.ViewElement

Base class of all view elements

Description

Base class of all view elements. This class inherits from `systemcomposer.base.BaseElement`.

Properties

`ZCIdentifier` — Identifier of object
character vector

Gets the identifier of an object. Used by Simulink Requirements.

Example: `identifier = get(objViewElement,'ZCIdentifier')`

Data Types: `char`

See Also

`systemcomposer.view.BaseViewComponent` | `systemcomposer.view.ComponentOccurrence` | `systemcomposer.view.ViewArchitecture` | `systemcomposer.view.ViewComponent`

Topics
“Create Architecture Views Interactively”
“Creating Architectural Views Programmatically”

Introduced in R2009b

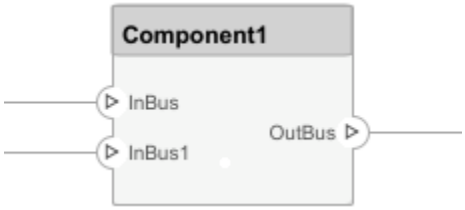
Blocks

Component

Add component to an architecture model

Description

Use a Component block to represent a structural or behavioral element at any level of an architecture model hierarchy. Add ports to the block for connecting to other components. Define an interface for the ports and add properties using stereotypes.



Ports

Input Port

`Source` — Provide connection from another component

Output Port

`Destination` — Provide connection to another component

See Also

Blocks
Adapter | Reference Component | Variant Component

Topics
“Implement Components in Simulink”

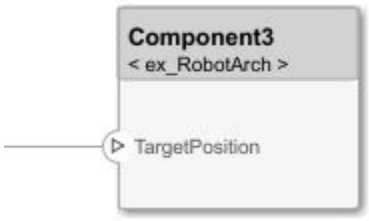
Introduced in R2019a

Reference Component

Link to an architectural definition or Simulink behavior

Description

Use a Reference Component block to link an architectural definition of a component or a Simulink behavior.



Ports

Input Port

`Source` — Provide connection from another component

Output Port

`Destination` — Provide connection to another component

See Also

Blocks
Adapter | Component | Variant Component

Topics
“Implement Components in Simulink”

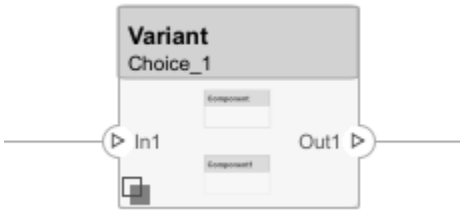
Introduced in R2019a

Variant Component

Add components with alternative designs

Description

Use a Variant Component block to create multiple design alternatives for a component.



Ports

Input Port

`Source` — Provide connection from another component

Output Port

`Destination` — Provide connection to another component

See Also

Blocks
Adapter | Component | Reference Component | Subsystem

Topics
“Decompose and Reuse Components”

Introduced in R2019a

Adapter

Connect components with different interfaces

Description

The Adapter block allows you to adapt dissimilar interfaces. Connect the source and destination ports of components that have different interface definitions.

Limitations

- When used for structural interface adaptations, the Adapter block uses bus element ports internally and, subsequently, only supports virtual buses.

Ports

Input Port

`Source` — Provide connection from a component

Output Port

`Destination` — Provide connection to a component

See Also

Blocks
Component | Reference Component | Variant Component

Topics
“Assign Interfaces to Ports”
“Interface Adapter”

Introduced in R2019a

Sequence Viewer

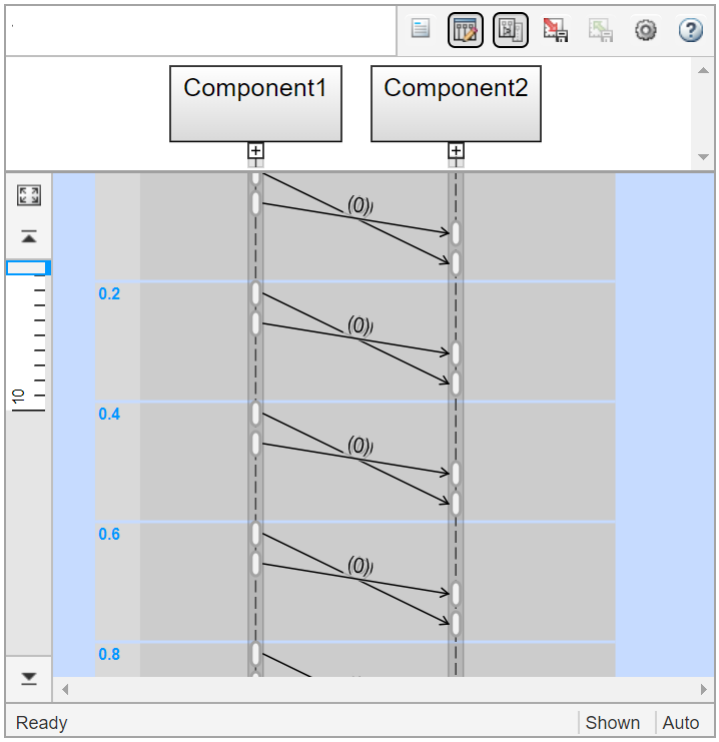
Visualize messages, events, states, transitions, and functions

Description

The Sequence Viewer visualizes message flow, function calls, and state transitions.

Use the Sequence Viewer to see the interchange of messages, events, function calls in Simulink behavior models in System Composer and between Stateflow® charts in Simulink models.

In the Sequence Viewer window, you can view event data related to Stateflow chart execution and the exchange of messages between Stateflow charts. The Sequence Viewer window shows messages as they are created, sent, forwarded, received, and destroyed at different times during model execution. The Sequence Viewer window also displays state activity, transitions, and function calls to Stateflow graphical functions, Simulink functions, and MATLAB functions. For more information, see “Use the Sequence Viewer Block to Visualize Messages, Events, and Entities”.



Open the Sequence Viewer

- Simulink Toolstrip: On the **Simulation** tab, in the **Review Results** section, click **Sequence Viewer**.

Examples

Using the Sequence Viewer Tool

1. To activate logging events, in the Simulink Toolstrip, under the **Simulation** tab, in the **Prepare** section, click **Log Events**.
2. Simulate your model.
3. To open the tool, in the Simulink Toolstrip, under the **Simulation** tab, in the **Review Results** section, click **Sequence Viewer**.

- "Use the Sequence Viewer Block to Visualize Messages, Events, and Entities"
- "Simulink Messages Overview"

Parameters

`Sequence Viewer Time Precision` — Digits for time increment precision

3 (default) | scalar

Number of digits for time increment precision. When using a variable step solver, change this parameter to adjust the time precision for the sequence viewer. By default the block supports `3` digits of precision. Minimum and maximum precision are `1` and `16`, respectively.

Suppose the block displays two events that occur at times `0.1215` and `0.1219`. Displaying these two events precisely requires `4` digits of precision. If the precision is `3`, then the block displays two events at time `0.121`.

Programmatic Use

Block Parameter: `SequenceViewerTimePrecision`
Type: character vector
Values: `'3'` | scalar
Default: `'3'`

`Sequence Viewer History` — Maximum number of previous events to display

`1000` (default) | scalar

Total number of events before the last event to display. Minimum and maximum number of events are `0` and `25000`, respectively.

For example, if **History** is `5` and there are `10` events in your simulation, then the block displays `6` events, including the last event and the five events prior the last event. Earlier events are not displayed. The time ruler is greyed to indicate the time between the beginning of the simulation and the time of the first displayed event.

Each send, receive, drop, or function call event is counted as one event, even if they occur at the same simulation time.

Programmatic Use

Block Parameter: `SequenceViewerHistory`
Type: character vector
Values: `'1000'` | scalar
Default: `'1000'`

See Also

Topics
“Use the Sequence Viewer Block to Visualize Messages, Events, and Entities”
“Simulink Messages Overview”

Introduced in R2020b