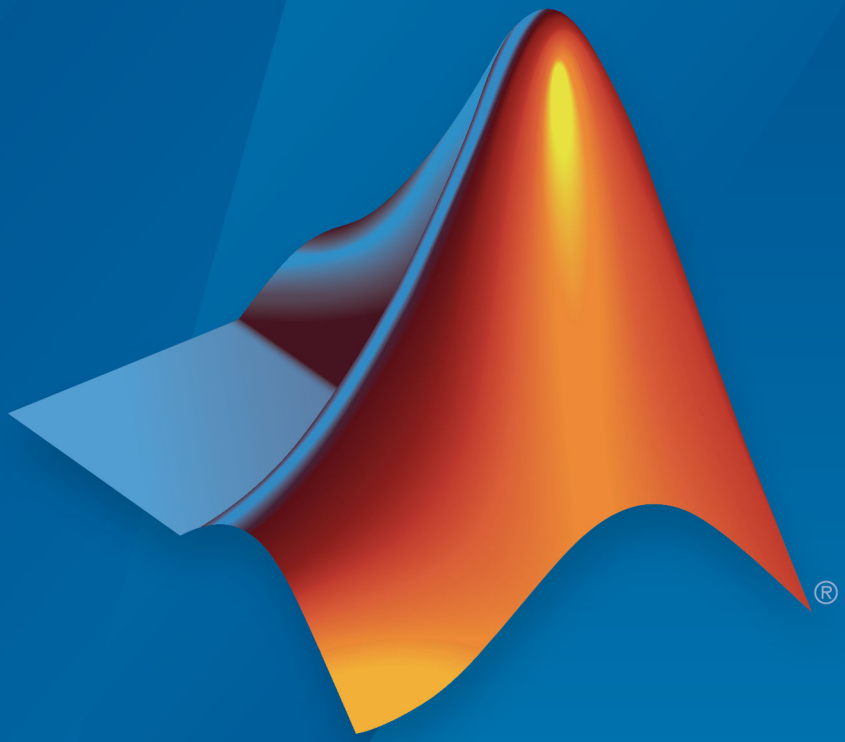
System Composer™

Reference



MATLAB® & SIMULINK®

R2019b

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

Revision History

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

Contents

Functions — Alphabetical List

addChoice

Add a variant choice to a 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.

Input Arguments

`variantComponent` — Architecture component
component

The architecture where the variant choices are added.

Data Types: `systemcomposer.arch.Component`

`choices` — Variant choice names
cell array of strings

Cell array where each element defines the name of a choice component. The length of `choices` must be the same as `labels`.

Data Types: `string`

`labels` — Variant choice labels
cell array of strings

Array of labels where each element is the label for the corresponding choice.. The length of `labels` must be the same as `choices`.

Data Types: `string`

Output Arguments

`compList` — Created components
array of components

Array of created components. This array is the same size as `choices` and `labels`.

See Also

`getActiveChoice` | `getChoices` | `makeVariant`

Topics

“Create Variants”

Introduced in R2019a

addComponent

Add a component to the 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 a Model with two Components

Create model, get root architecture, and create components.

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

Input Arguments

`architecture` — Architecture model element
architecture

Parent architecture to which the component is added.

Data Types: `systemcomposer.arch.Architecture`

`compNames` — Names of components
cell array of strings

Cell array where each element defines the name of a new component. The length of `compNames` must be the same as `stereotypes`.

Data Types: `string`

`stereotypes` — Stereotypes to apply to the components
cell array of stereotypes

Array of stereotypes where each element is the qualified stereotype name for the corresponding component in the form `'<profileName>.<stereotypeName>'`. The length of `stereotypes` must be the same as `compNames`.

Data Types: `string`

Output Arguments

`components` — Created components
array of components

Array of created components. This array is the same size as `compNames` and `stereotypes`.

See Also

`addPort` | `connect`

Topics

“Components”

Introduced in R2019a

addComponent_

Add component to view given path

Syntax

```
compOccur = addComponent(object, compPath, contextView)
```

Description

`compOccur = addComponent(object, compPath, contextView)` adds the component with the specified path to the view given by the parameter ‘contextView’.

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

Input Arguments

`object` — <argument purpose>
<object> (default) | <object>

<argument description>

Data Types: `<object data type>`

`compPath` — <argument purpose>
<argument value> (default) | <argument value>

Path to the component including the name of the top-model.

Data Types: `<argument data type>`

`contextView` — <argument purpose>
<argument value> (default) | <argument value>

Property ‘Parent’ is empty.

Data Types: `systemcomposer.view.ViewArchitecture`

Output Arguments

`parent` — <argument purpose>
<argument value>

<argument description>

Data Types: `<argument data type>`

See Also

Introduced in R2019b

addVariantComponent

Add a component to the 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 a 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` — Architecture model element
architecture

Parent architecture to which the component is added.

Data Types: `systemcomposer.arch.Architecture`

`variantComponents` — Names of variant components
cell array of strings

Cell array where each element defines the name of a variant component.

Data Types: `string`

`position` — four-element vector that specifies location of the top corner of the component
1x4 array

The array denotes the top corner of the component 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

Array of variant components. This array is the same size as `variantComponents`.

See Also

`addPort` | `connect`

Topics

“Components”

Introduced in R2019a

addElement

Add a 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 `newinterface` to the interface dictionary of the model and add an element with type `double` to it.

```
interface = addInterface(archModel.InterfaceDictionary,'newsignal');
element = addElement(interface,'newelement','Type','double)
```

Input Arguments

`interface` — new interface object
signal interface

This is the interface that the new element is to be added.

Data Types: `systemcomposer.interface.SignalInterface`

`name` — Name of the new element
string

The new element name must be a valid variable name.

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` — Type of element
valid data type string

Data type of the element. Must be a valid data type.

Data Types: `char`

`Dimensions` — Dimensions of element
positive integer array

Each element is the size of the element in the corresponding direction. A scalar integer indicates a scalar or vector element, a row vector with two integers indicates a matrix element.

Data Types: `char`

`Complexity` — Complexity of element
real | complex

This describes whether the element is purely real, or if an imaginary part is allowed.

Data Types: `string`

Output Arguments

`element` — new interface element object
signal element

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.

Examples

Add Ports to Architecture

Create model, get root architecture, add 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

`addPort` adds ports to the architecture of a component. Use <*component*>*.Architecture* to access the architecture of a component.

Data Types: `systemcomposer.arch.Architecture`

`portNames` — Names of ports
cell array of strings

Port names must be unique within each component. 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: `string`

`portTypes` — Port directions
cell array of strings

Port directions are given in a cell array. Each element is either `'in'` or `'out'`.

Data Types: `string`

`stereotypes` — Stereotypes to apply to the components
Array of stereotypes

Each stereotype in the array must either be a mixin stereotype or a port stereotype. The size of `portNames`,`portTypes`, and `stereotypes` must be the same.

Data Types: `systemcomposer.profile.Stereotype`

Output Arguments

`ports` — Created ports
Array of ports

See Also

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

Topics

“Ports”

Introduced in R2019a

addInterface

Create a named interface in an interface dictionary

Syntax

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

Description

`interface = addInterface(dictionary,name)` creates a named interface in the interface dictionary.

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

Examples

Add an Interface

Add an interface `newinterface` to the interface dictionary of the model.

```
addInterface(archModel.InterfaceDictionary,'newinterface')
```

Input Arguments

`dictionary` — Data dictionary attached to the architecture model

System Composer dictionary

`dictionary` can be 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.

Data Types: `systemcomposer.interface.Dictionary`

`name` — Name of the new interface
string

The name of the new interface must be a valid variable name.

Data Types: `char`

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

Use this argument when the interface is already defined in a Simulink Bus object.

Data Types: `simulink bus`

Output Arguments

`interface` — new interface object
signal interface

Interface object with properties `Dictionary`, `Name`, and `Elements`.

See Also

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

Topics

"Define Interfaces"

Introduced in R2019a

addProperty

Add a property to a stereotype

Syntax

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

Description

`property = addProperty(stereotype,name,Name,Value)` adds a new property with the specified `Name,Value` attributes.

Examples

Add a Property

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

```
stype = addStereotype(profile,'electricalComponent','AppliesTo','Component')
property = addProperty(stype,'VoltageRating','DefaultValue','5');
```

Input Arguments

`stereotype` — Stereotype to which the property is added
stereotype

`name` — Name of the property
string

Name of the property must be unique within the stereotype.

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: `'Datatype', 'double'`

`Type` — Property data type
valid data type string

Data Types: `char`

`Dimensions` — Dimensions of property
positive integer array

Data Types: `char`

`Min` — Minimum value
numeric value

Data Types: `double`

`Max` — Maximum value
numeric value

Data Types: `double`

`Units` — Property units
string

Data Types: `char`

`DefaultValue` — Default value
numeric value

Data Types: `double`

Output Arguments

`property` — Created property
property

See Also

`getProperty` | `setProperty`

Topics

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

Introduced in R2019a

addStereotype

Add a stereotype to the 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 a Component Stereotype

Add a component stereotype to the profile.

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

Input Arguments

`profile` — Profile object
profile

The profile that contains the new stereotype.

Data Types: `systemcomposer.profile.Profile`

`stereotypeName` — Name of new stereotype
string

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: `'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

See Also

`applyStereotype` | `removeStereotype`

Topics

“Define Profiles and Stereotypes”

Introduced in R2019a

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
architecture model

Data Types: `systemcomposer.arch.Model`

`profileFile` — Profile file
string

Data Types: `string`

See Also

`createProfile` | `removeProfile`

Topics

“Define Profiles and Stereotypes”

Introduced in R2019a

applyStereotype

Apply a stereotype to a model element

Syntax

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

Description

`applyStereotype(element,stereotype)` applies a stereotype to a model element. Adds the specified stereotype if not already applied to a model element. Stereotypes can be applied to Base Architecture, Base Architecture port, and Base Connector model elements.

Input Arguments

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

The stereotype is applied to this component, port, or connector.

Data Types: `systemcomposer.arch.Element`

`stereotype` — Reference stereotype
architecture stereotype

The qualified stereotype name in the form `<profile>.<stereotype>`. The profile must already be applied to the model.

Data Types: `char`

See Also

`batchApplyStereotype` | `removeStereotype`

Topics

“Use Stereotypes and Profiles”

Introduced in R2019a

open

Open System Composermodel

Syntax

```
open(objModel)
```

Description

`open(objModel)` opens the specified model in the System Composer editor if it is not already open.

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

Data Types: `systemcomposer.arch.Model`

See Also

`createModel`

Topics

“Create an Architecture Model”

Introduced in R2019a

batchApplyStereotype

Apply stereotype to all elements in the 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 `elementType` within `architecture`.

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

Examples

Apply a Stereotype to All Connectors

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

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

Input Arguments

`architecture` — Architecture model element

architecture

Parent architecture layer for all components to attach the stereotype.

Data Types: `systemcomposer.arch.Architecture`

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

The element type to apply the stereotype. The stereotype must be applicable for this element type.

Data Types: `string`

`stereotype` — Stereotype to apply
string

Qualified name for the stereotype in the form `'profileName.stereotypeName'` The stereotype must be applicable to components.

Data Types: `string`

`flag` — Apply stereotype recursively
true | false

If this flag is set, the stereotype is applied to the elements in the architecture and its sub-architectures.

Data Types: `logical`

See Also

`removeStereotype`

Topics

“Use Stereotypes and Profiles”

Introduced in R2019a

connect

Connect pairs of components

Syntax

```
connectors = connect(srcComponent,destComponent)
connectors = connect(srcComponent,destComponent,'Stereotype',
stereotype)
connectors = connect(srcComponent,destComponent,'Rule',rule)
connectors = connect(architecture,srcPorts,destPorts,stereotypes,
rule)
```

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(srcComponent,destComponent,'Stereotype', stereotype)` additionally applies the specified stereotype to the connector.

`connectors = connect(srcComponent,destComponent,'Rule',rule)` specifies a rule for establishing connections.

`connectors = connect(architecture,srcPorts,destPorts,stereotypes, rule)` connects pairs of ports in the architecture.

Examples

Connect Components

Create model, get root architecture, add ports, and connect ports.

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

```
names = {'Component1','Component2'};
newcomponents = addComponent(rootArch,names);
outport1 = addPort(newcomponents(1).Architecture,'','OutputPort');
inport1 = addPort(newcomponents(2).Architecture,'InputPort','');
connect(rootArch,outport1, inport1);
```

Input Arguments

`architecture` — Architecture model element

Architecture

Data Types: `systemcomposer.arch.Architecture`

`srcPorts` — Array of source ports

array of ports

`srcPorts` must be the same length as `destPorts` and must consist of all output ports.

Data Types: `systemcomposer.arch.Port`

`destPorts` — Array of destination ports

array of ports

`destPorts` must be the same length as `srcPorts` and must consist of all source ports.

Data Types: `systemcomposer.arch.Port`

`srcComponent` — Source component

architecture component

Data Types: `systemcomposer.arch.Component`

`destComponent` — Destination component

architecture component

Data Types: `systemcomposer.arch.Component`

`stereotypes` — Stereotypes to apply to the connections

Array of stereotypes

Data Types: `systemcomposer.profile.Stereotype`

`rule` — Rule to match ports for connection

`'name'` | `'interface'`

Data Types: `systemcomposer.arch.Component`

Output Arguments

`connectors` — Created connections
Array of connections

See Also

`addPort`

Topics

“Create an Architecture Model”

Introduced in R2019a

systemcomposer.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 return a handle.

Input Arguments

`dictionaryName` — Name of new data dictionary
string

The name must include the `.sldd` extension

Example: 'new_dictionary.sldd'

Data Types: `char`

Output Arguments

`dictionary_id` — Handle to the dictionary
dictionary object

Examples

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

See Also

`addInterface` | `linkDictionary` | `save` | `unlinkDictionary`

Topics

“Save and Link Interfaces”

Introduced in R2019a

createModel

Create a System Composer model

Syntax

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

Description

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

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

Input Arguments

`modelName` — Name of a new model
character vector | string

Model name must be a valid MATLAB variable name.

Data Types: `char` | `string`

Output Arguments

`objModel` — Model handle
Model object

Data Types: `systemcomposer.arch.Model`

Examples

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

```
Model =

  Model with properties:

                    Name: 'model_name'
              ActiveView: []
            Architecture: [1×1 systemcomposer.arch.Architecture]
          SimulinkHandle: 1.2207e-04
                   Views: [0×0 systemcomposer.view.ViewArchitecture]
                Profiles: [0×0 systemcomposer.profile.Profile]
      InterfaceDictionary: [1×1 systemcomposer.interface.Dictionary]
```

See Also

`loadModel` | `open` | `save`

Topics

“Compose Architecture Visually”

Introduced in R2019a

createProfile

Create profile

Syntax

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

Description

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

Input Arguments

`profileName` — Name of new profile
string

Example: 'new_profile'

Data Types: `char` | `string`
Complex Number Support: No

Output Arguments

`profile` — Profile handle
profile object

Examples

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

See Also

`applyProfile` | `removeProfile` | `systemcomposer.loadProfile`

Topics

“Create a Profile and Add Stereotypes”

Introduced in R2019a

createSimulinkBehavior

Create a 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
architecture component

The component must have no children.

Data Types: `systemcomposer.arch.Component`

`modelName` — Model name
string

Name of the Simulink model created by this function.

Data Types: `char`

See Also

`linkToModel`

Topics

“Implement Components in Simulink”

Introduced in R2019a

createViewArchitecture

Create a view

Syntax

```
view = createViewArchitecture(obj, constraint, rootArch,
isRecursive, groupBy, nameValPair)
view = createViewArchitecture(obj, name, constraint, rootArch,
isRecursive, groupBy, nameValPair)
```

Description

`view = createViewArchitecture(obj, constraint, rootArch, isRecursive, groupBy, nameValPair)` creates an empty view with the given name.
`view = createViewArchitecture(obj, name, constraint, rootArch, isRecursive, groupBy, nameValPair)` creates a view with the given name whose contents are populated by finding all components in the given architecture given by the rootArch parameter which satisfies the given constraint. If isRecursive is true, then createViewArchitecture applies the query recursively on all components under the rootArch. The groupBy parameter is the fully qualified property name to group the components in the query.

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

Input Arguments

`obj` — <argument purpose>
<object value> (default)

Data Types: `<object data type>`

`name` — <argument purpose>
<object value> (default)

Data Types: `<object data type>`

`nameValPair` — `<argument purpose>`
<object value> (default)

Data Types: `<object data type>`

`constraint` — `<argument purpose>`
<object value> (default)

Data Types: `<object data type>`

`rootArch` — `<argument purpose>`
<object value> (default)

Data Types: `<object data type>`

`isRecursive` — `<argument purpose>`
<object value> (default)

Data Types: `<object data type>`

See Also

Introduced in R2019b

createViewComponent

Create new view component

Syntax

```
vc = createViewComponent(object, name, contextView)
```

Description

`vc = createViewComponent(object, name, contextView)` creates a new view component with the provided name in the view given by the parameter 'contextView'.

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

Input Arguments

`object` — <argument purpose>
<object> (default) | <object>

<argument description>

Data Types: `<object data type>`

`name` — Name of component
character vector (default)

Name of component

Data Types: `character vector`

`contextView` — <argument purpose>
<argument value> (default) | <argument value>

Property 'Parent' is empty.

Data Types: `systemcomposer.view.ViewArchitecture`

Output Arguments

`parent` — <argument purpose>
<argument value>

<argument description>

Data Types: `<argument data type>`

See Also

Introduced in R2019b

deleteInstance

Delete an architecture instance

Syntax

```
deleteInstance(architectureInstance)
```

Description

`deleteInstance(architectureInstance)` deletes an existing instance.

Input Arguments

`architectureInstance` — The architecture instance
architecture instance

The architecture instance to be deleted.

Data Types: `systemcomposer.analysis.ArchitectureInstance`

See Also

`instantiate`

Topics

“Write Analysis Function”

Introduced in R2019a

destroy

Remove and destroy a 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 | interface element | signal element | property

Data Types: `systemcomposer.arch.Element` | `systemcomposer.interface.SignalInterface` | `systemcomposer.interface.SignalElement` | `systemcomposer.profile.Property`

See Also

`removeElement` | `removeProfile` | `removeProperty`

Introduced in R2019a

systemcomposer.exportModel

Export model information as MATLAB tables

Syntax

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

Description

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

Input Arguments

`modelName` — Name of model to be exported
string | character vector

Name of System Composer model to be exported, specified as a string.

Example: 'exMobileRobot'

Data Types: `char` | `string`

Output Arguments

`exportedSet` — Model tables
struct

Structure containing tables `components`, `ports`, `connections`, and `portInterfaces`.

Examples

Export a System Composer Model

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

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

exportedSet =

  struct with fields:

        components: [11×4 table]
             ports: [22×4 table]
       connections: [16×4 table]
    portInterfaces: [0×9 table]
```

See Also

`systemcomposer.importModel`

Topics

“Importing and Exporting Architecture Models”

Introduced in R2019a

systemcomposer.extractArchitectureFromSimulink

Link component to a model

Syntax

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

Description

`systemcomposer.extractArchitectureFromSimulink(SimulinkModel, architectureModelName)` exports the Simulink model `SimulinkModel` 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

`SimulinkModel` — Model from which to extract the architecture

Simulink model

The model must be on the path.

Data Types: `model`

`architectureModelName` — Architecture model name
string

A new architecture model that shows the architecture of the Simulink model. This model is saved in the current directory.

Data Types: `char`

See Also

`linkToModel`

Topics

“Extract Architecture from Simulink Model”

Introduced in R2019a

find

Find model elements

Syntax

```
[paths, e] = find(obj, constraint, rootArch, nameValPair)
[paths, e] = find(obj, constraint, nameValPair)
```

Description

`[paths, e] = find(obj, constraint, rootArch, nameValPair)`, `[paths, e] = find(obj, constraint, nameValPair)` finds model elements in the specified architecture rootArch using the specified constraint in the model. If rootArch is not provided, then it will find model elements in the root architecture of the model. The output argument paths will contain the fully qualified named path to the element starting from the given root architecture. The following name value pairs are supported:

- 'FlattenReferences': {true, false} – Indicates if the find should search referenced architectures or it should not include referenced architectures. The default is 'false'.
- 'Recurse': {true, false} – Indicates if the find should recursively search through the model or if it should search only the specified layer. The default is 'true'.
- 'ElemType': {'Component', 'Port', 'Connector'} – Specifies what element type to search for in the model. This parameter dictates the return type of "e". The default is 'Component'.

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

Input Arguments

`obj` — <argument purpose>
<argument value> (default)

Data Types: `<argument type>`

`constraint` — <argument purpose>
<argument value> (default)

Data Types: `<argument type>`

`rootArch` — <argument purpose>
<argument value> (default)

Data Types: `<argument type>`

`nameValPair` — <argument purpose>
<argument value> (default)

Data Types: `<argument type>`

Output Arguments

`paths` — <argument purpose>
<argument value>

<argument description>

Data Types: `<argument data type>`

`e` — <argument purpose>
<argument value>

<argument description>

Data Types: `<argument data type>`

See Also

Introduced in R2019b

getActiveChoice

Get the active choice on the variant component

Syntax

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

Description

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

Input Arguments

`variantComponent` — Architecture component
component

The architecture where the variant choices are selected.

Data Types: `systemcomposer.arch.Component`

Output Arguments

`choice` — Handle of chosen variant
component

Handle to the chosen variant.

Data Types: `systemcomposer.arch.Component`

See Also

`addChoice` | `getChoices` | `setActiveChoice`

Topics

“Create Variants”

Introduced in R2019a

getChoices

Get available choices in the variant component

Syntax

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

Description

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

Input Arguments

`variantComponent` — Architecture component
component

Variant component with multiple choices.

Data Types: `systemcomposer.arch.Component`

Output Arguments

`compList` — Choices available for the variant component
array of components

List of possible choices for the variant component.

See Also

`addChoice` | `getActiveChoice` | `setActiveChoice`

Topics

“Create Variants”

Introduced in R2019a

getCondition

Return the variant control on the choice within the variant component

Syntax

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

Description

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

Input Arguments

`variantComponent` — Architecture component
component

Variant component with multiple choices.

Data Types: `systemcomposer.arch.Component`

`choice` — Choice in a variant component
component

The choice whose control string is returned by this function.

Data Types: `systemcomposer.arch.Component`

Output Arguments

`expression` — The control string
string

The control string that controls the selection of the particular `choice`.

See Also

`makeVariant` | `setActiveChoice` | `setCondition`

Topics

“Create Variants”

Introduced in R2019a

getElement

Get the object a 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 the Object for a Named Element

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

```
interface = addInterface(arch.InterfaceDictionary,'newsignal');
addElement(interface,'newelement','Type','double)
element = getElement(interface,'newsignal')
element =
  SignalElement with properties:

      Interface: [1×1 systemcomposer.interface.SignalInterface]
           Name: 'newelement2'
           Type: 'double'
     Dimensions: '1'
          Units: ''
     Complexity: 'real'
        Minimum: '[]'
        Maximum: '[]'
    Description: ''
```

```
        UUID: 'f42c8166-e4ad-4488-926a-293050016e1a'
ExternalUID: ''
```

Input Arguments

`interface` — interface object
signal interface

The object handle to the element to be identified.

Data Types: `systemcomposer.interface.SignalInterface`

`elementName` — Name of the element to be identified
string

Data Types: `char`

Output Arguments

`element` — new interface element object
signal element

See Also

`addElement` | `getInterface` | `removeElement`

Topics

“Define Interfaces”

Introduced in R2019a

getInterface

Get the object for a named interface in an 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 an Interface

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

```
addInterface(arch.InterfaceDictionary,'newsignal')
iface = getInterface(arch.InterfaceDictionary,'newsignal')
iface =
  SignalInterface with properties:
     Dictionary: [1×1 systemcomposer.interface.Dictionary]
           Name: 'newsignal'
       Elements: [0×0 systemcomposer.interface.SignalElement]
           UUID: '438b5004-6cab-40eb-955b-37e0df5a914f'
    ExternalUID: ''
```

Input Arguments

`dictionary` — Data dictionary
System Composer dictionary

This is the data dictionary attached to the model. It could be the local dictionary of the model or an external data dictionary.

Data Types: `systemcomposer.interface.Dictionary`

`name` — Name of the interface
string

Data Types: `char`

Output Arguments

`interface` — object for the interface
signal interface

See Also

`addElement` | `addInterface` | `removeElement`

Topics

"Define Interfaces"

Introduced in R2019a

getInterfaces

Get the object for a named interface in an interface dictionary

Syntax

```
interfaceList = getInterfaces(dictionary)
```

Description

`interfaceList = getInterfaces(dictionary)` gets the list of objects in the interface dictionary.

Examples

Get Interface List

```
ifaceList = getInterfaces(arch.InterfaceDictionary)
```

Input Arguments

`dictionary` — Data dictionary

System Composer dictionary

This is the data dictionary attached to the model. It could be the local dictionary of the model or an external data dictionary.

Data Types: `systemcomposer.interface.Dictionary`

Output Arguments

`interfaceList` — interface object list

array of signal interfaces

See Also

`addInterface` | `getInterface`

Topics

“Define Interfaces”

Introduced in R2019a

getProperty

Get the property value corresponding to a stereotype applied to the 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 a Property from a 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

architecture component | architecture port | architecture connector

This function gets the specified property of this element. A stereotype with the property must be applied to the element.

Data Types: `systemcomposer.arch.Element` | `systemcomposer.arch.Architecture` | `systemcomposer.arch.Component` | `systemcomposer.arch.Port`

`propertyName` — Name of the property
string

The property name must be qualified with the stereotype name, in the form `'<stereotype>.<property>'`.

Data Types: `char`

Output Arguments

`propertyValue` — Value of the property
string | number | enumeration

Data Types: `char`

`propertyUnits` — Unit of the property
string

Data Types: `char`

See Also
`setProperty`

Topics
“Set Tags and Properties for Analysis”

Introduced in R2019a

getStereotypes

Get the stereotypes applied on the element

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

```
stypes = getStereotypes(component_handle)
```

Input Arguments

`element` — Model element
component | port | connector

This is the element of which stereotypes are queried.

Data Types: `systemcomposer.arch.Element`

Output Arguments

`stereotypes` — list of stereotypes
cell array of stereotypes

See Also

`applyStereotype` | `removeStereotype`

Topics

“Use Stereotypes and Profiles”

Introduced in R2019a

getValue

Get value of a property from an 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 the Weight Property

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

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

Input Arguments

`instance` — The element instance

architecture instance | component instance | port instance | connector instance

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.

Data Types: `systemcomposer.analysis.ArchitectureInstance` | `systemcomposer.analysis.ComponentInstance` | `systemcomposer.analysis.PortInstance` | `systemcomposer.analysis.ConnectorInstance`

`property` — The property field
stereotype.property

String in the form `<stereotype>.<property>`.

Data Types: `string`

Output Arguments

`value` — Property value
any variable type

Value of the property. The data type depends on how the property is defined in the profile.

`unit` — Property unit
string

String that describe the unit of the property as defined in the profile.

See Also

`setValue`

Topics

“Write Analysis Function”

Introduced in R2019a

systemcomposer.importModel

Import model information from MATLAB tables

Syntax

```
archModel = systemcomposer.importModel(modelName,components,ports,
connections)
```

Description

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

Input Arguments

`modelName` — Name of model to be created
string

Example: 'importedModel'

Data Types: `char` | `string`

`components` — Component information
MATLAB table

Model components listed in a table created in MATLAB. The component table must include name, unique ID, and parent component ID 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` — Port information
MATLAB table

Model ports listed in a table created in MATLAB. The ports table must include port name, direction, component, and port ID information. Port interface information may also be required to assign ports to components..

Data Types: `table`

`connections` — Connections information
MATLAB table

Model connections listed in a table created in MATLAB. The ports table must include port name, direction, component, and port ID information. Port interface information may also be required to assign ports to components..

Data Types: `table`

Output Arguments

`archModel` — Handle to the architecture model
architecture object

Handle to the architecture model, specified as an architecture 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. 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, and interfaces 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 such as referenced model, stereotype qualifier name and so on. required to construct the architecture hierarchy. 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. This includes, at a minimum, connection ID, source port ID, and destination port ID.

The systemcomposer.importModel(importModelName) API :

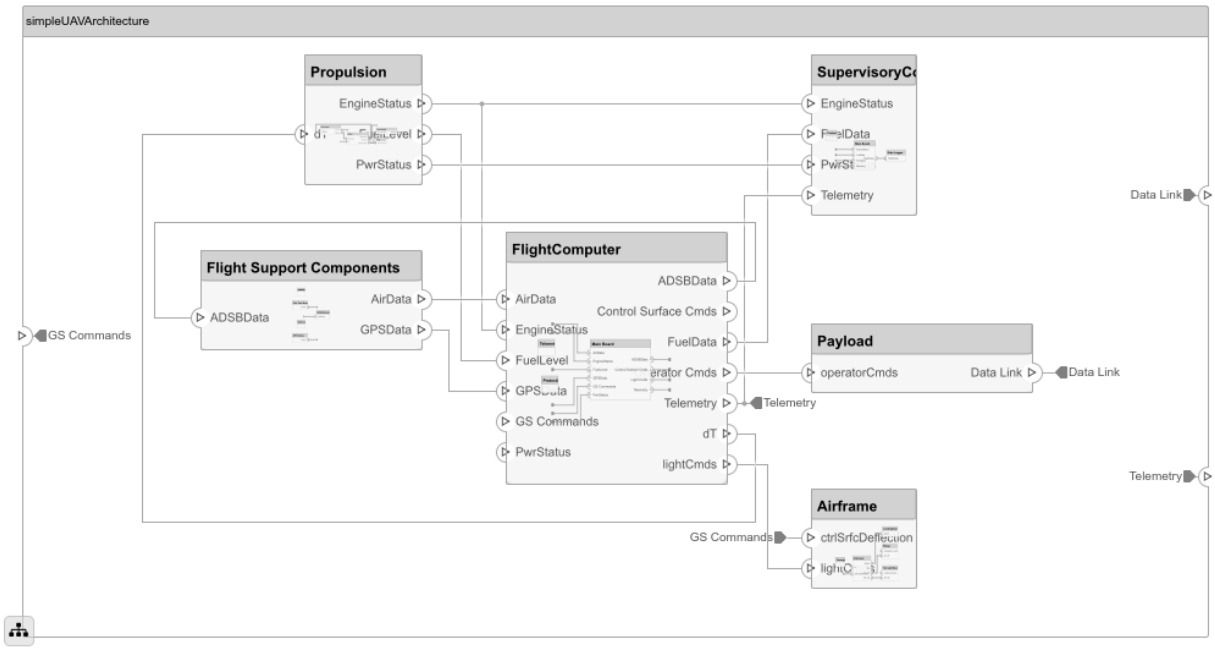
- Reads stereotype names from Component table and load the profiles
- Creates components and attach ports
- Creates connections using the connection map
- Saves referenced models
- Saves the architecture model

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
% 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','Connecti
importAdapter.readTableFromExcel();
```

Import an Architecture

```
model = systemcomposer.importModel(modelName,importAdapter.Components,importAdapter.Por
% 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 tab
% connection table, and the interface table.
% Save the above structure to excel file.
SaveToExcel('ExportedUAVModel',exportedSet);
```

Close Model

```
bdclose(modelName);
```

See Also

`systemcomposer.exportModel`

Topics

“Importing and Exporting Architecture Models”

Introduced in R2019a

inlineComponent

Inline reference architecture into model

Syntax

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

Description

`componentHandle = inlineComponent(component,inlineFlag)` inlines the contents of the architecture model 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 a 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

architecture component

The component must be linked to an architecture model.

Data Types: `systemcomposer.arch.Component`

`inlineFlag` — control the contents of the inlined component
true | false

If `true`, contents of the referenced architecture model are copied to the component architecture. If `false`, the contents are not copied, only ports and interfaces are inlined.

Data Types: `char`

Output Arguments

`componentHandle` — Component object
architecture component

See Also

`saveAsModel`

Topics

“Decompose and Reuse Components”

Introduced in R2019a

instantiate

Create an analysis instance from a specification

Syntax

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

Description

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

Input Arguments

`model` — Handle to the model
model handle

The instance is generated from the model specified in this argument.

`properties` — Stereotype properties which require values in the instance model
instance properties object

Each value for an instance in an instance model can be drawn from any stereotype in any profile on the path. The structure of the property definition parameter accommodates this approach. The definition is a structure with a field for each profile of interest. The name of the field is the name of the profile. Each profile field is itself a structure, which has a field per stereotype whose name is the name of the stereotype. Each stereotype in turn is another structure that contains two fields, one called `properties`, which specifies properties of interest and another called `elementKinds` which indicates the kinds of instance to which the values corresponding to the properties are added. The `properties` field is a structure that lists the required properties as Boolean fields; the name of the field is the name of the property and the value indicates whether the field can be set via the API. The `elementKinds` field is a list of strings whose value must be one of: `'Component'`, `'Port'` or `'Connector'` to indicate the applicable elements.

Data Types: `systemcomposer.analysis.InstanceProperties`

`name` — Name of the instance
string

This is the name given to the instance generated from the model.

Output Arguments

`instance` — The element instance
architecture instance | component instance | port instance | connector instance

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.

Data Types: `systemcomposer.analysis.ArchitectureInstance`

See Also

`deleteInstance` | `loadInstance` | `saveInstance`

Topics

“Write Analysis Function”

Introduced in R2019a

isArchitecture

Find if an instance is a architecture instance

Syntax

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

Description

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

Input Arguments

`instance` — The element instance

architecture instance | component instance | port instance | connector instance

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.

Data Types: `systemcomposer.analysis.ArchitectureInstance` | `systemcomposer.analysis.ComponentInstance` | `systemcomposer.analysis.PortInstance` | `systemcomposer.analysis.ConnectorInstance`

Output Arguments

`flag` — Indicate if the instance is a architecture

boolean

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

See Also

`isComponent` | `isConnector` | `isPort`

Topics

“Write Analysis Function”

Introduced in R2019a

isComponent

Find if an instance is a component instance

Syntax

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

Description

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

Input Arguments

`instance` — The element instance

architecture instance | component instance | port instance | connector instance

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.

Data Types: `systemcomposer.analysis.ArchitectureInstance` | `systemcomposer.analysis.ComponentInstance` | `systemcomposer.analysis.PortInstance` | `systemcomposer.analysis.ConnectorInstance`

Output Arguments

`flag` — Indicate if the instance is a component

boolean

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

See Also

`isArchitecture` | `isConnector` | `isPort`

Topics

“Write Analysis Function”

Introduced in R2019a

isConnector

Find if an instance is a connector instance

Syntax

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

Description

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

Input Arguments

`instance` — The element instance

architecture instance | component instance | port instance | connector instance

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.

Data Types: `systemcomposer.analysis.ArchitectureInstance` | `systemcomposer.analysis.ComponentInstance` | `systemcomposer.analysis.PortInstance` | `systemcomposer.analysis.ConnectorInstance`

Output Arguments

`flag` — Indicate if the instance is a connector

boolean

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

See Also

`isArchitecture` | `isComponent` | `isPort`

Topics

“Write Analysis Function”

Introduced in R2019a

isPort

Find if an instance is a port instance

Syntax

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

Description

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

Input Arguments

`instance` — The element instance

architecture instance | component instance | port instance | connector instance

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.

Data Types: `systemcomposer.analysis.ArchitectureInstance` | `systemcomposer.analysis.ComponentInstance` | `systemcomposer.analysis.PortInstance` | `systemcomposer.analysis.ConnectorInstance`

Output Arguments

`flag` — Indicate if the instance is a port

boolean

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

See Also

`isArchitecture` | `isConnector` | `isConnector`

Topics

“Write Analysis Function”

Introduced in R2019a

iterate

Iterate over model elements

Syntax

```
iterate(architecture,iterType,iterFunction)
iterate(architecture,iterType,iterFunction,'Recurse',false)
iterate(architecture,iterType,iterFunction,'IncludePorts',true)
iterate(architecture,iterType,
iterFunction,'FollowConnectivity',true)
iterate(architecture,iterType,iterFunction,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(architecture,iterType,iterFunction,'Recurse',false)` iterates over components only in this architecture and does not navigate into the architectures of child components.

`iterate(architecture,iterType,iterFunction,'IncludePorts',true)` iterates over components and architecture ports.

`iterate(architecture,iterType,
iterFunction,'FollowConnectivity',true)` ensures components are visited according to how they are connected from source to destination. If this option is specified, iteration type has to be either 'TopDown' or 'BottomUp'. If any other option is specified, iteration defaults to 'TopDown'.

`iterate(architecture,iterType,iterFunction,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 stores
% analysis results.
objcomputeBatterySizing = computeBatterySizing;
% Run the analysis using the iterator
archModel.iterate('Topdown',@computeLoad,objcomputeBatterySizing);
```

Input Arguments

`architecture` — Architecture to iterate over

architecture

The iteration type traverses elements in 'depth-first pre-order', 'depth-first post-order', 'breadth-first top-down', or 'breadth-first bottom-up' order.

Data Types: `systemcomposer.arch.Architecture`

`iterType` — Iteration type

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

The iteration type traverses elements in 'depth-first pre-order', 'depth-first post-order', 'breadth-first top-down', or 'breadth-first bottom-up' order.

Data Types: `char`

`iterFunction` — Iteration function

function handle

Handle to the function to be iterated on each component.

Data Types: `string`

`additionalArgs` — Additional function arguments

function argument

Comma separated list of arguments to be passed to `iterFunction`

See Also

Topics

“Analyze Architecture”

Introduced in R2019a

linkDictionary

Link data dictionary to an architecture model

Syntax

```
linkDictionary(modelObject,dictionaryFile)
```

Description

`linkDictionary(modelObject,dictionaryFile)` associates the specified Simulink Data Dictionary with the model.

Input Arguments

`modelObject` — Architecture model object

Data Types: `systemcomposer.arch.Model`

`dictionaryFile` — Dictionary file name with the `.sldd` extension

string

Data Types: `string`

See Also

`getInterfaces` | `systemcomposer.createDictionary`

Topics

“Save and Link Interfaces”

Introduced in R2019a

linkToModel

Link component to a model

Syntax

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

Description

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

Examples

Reuse a 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

architecture component

The component must have no children.

Data Types: `systemcomposer.arch.Component`

`modelName` — Model name
string

An existing model that define the architecture or behavior of the component.

Data Types: `char`

Output Arguments

`modelHandle` — Handle to the linked model
numeric handle

See Also

`inlineComponent`

Topics

“Decompose and Reuse Components”

Introduced in R2019a

loadInstance

Load an architecture instance

Syntax

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

Description

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

Input Arguments

`fileName` — File that contains an architecture instance
string

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

`overwrite` — Whether to overwrite an instance if it already exists in the workspace
1 | 0

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

See Also

`deleteInstance` | `saveInstance` | `updateInstance`

Topics

“Write Analysis Function”

Introduced in R2019a

loadModel

Load architecture model

Syntax

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

Description

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

Input Arguments

`modelName` — Name of model
string

Model must exist on the MATLAB path.

Example: 'new_arch'

Data Types: `char` | `string`

Output Arguments

`model` — Model handle
Model object

Examples

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

See Also

open | save

Topics

“Create an Architecture Model”

Introduced in R2019a

systemcomposer.loadProfile

Load profile

Syntax

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

Description

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

Input Arguments

`profileName` — Name of new profile
string

Profile must be available on the MATLAB path.

Example: 'new_profile'

Data Types: `char` | `string`

Output Arguments

`profile` — Profile handle
Profile object

Examples

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

See Also

`applyProfile`

Topics

“Define Profiles and Stereotypes”

Introduced in R2019a

lookup

Lookup an architecture element

Syntax

`lookup(modelObject,Name,Value)`

Description

`lookup(modelObject,Name,Value)` finds an architecture element based in its UUID or full path.

Examples

Look up a 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

`modelObject` — Architecture model object

Data Types: `systemcomposer.arch.Model`

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: `'Path', 'RobotSystem/Sensors'`

`UUID` — UUID of the element

character vector

Data Types: `char`

`Path` — Path to the element

character vector

Path to the model element, specified as a character vector.

Data Types: `char`

`SimulinkHandle` — Simulink handle of the element

double

Simulink handle of the element

Data Types: `double`

See Also

`instantiate`

Topics

"Analyze Architecture"

Introduced in R2019a

makeVariant

Convert component to a 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 components

Architecture components to be converted to variants.

Data Types: `systemcomposer.arch.Component`

Output Arguments

`variantComp` — Component containing the variants
component

Component that contains the variants.

`choices` — Variant choice names
cell array of strings

Choices available in the new variant.

Data Types: `string`

See Also

`addChoice` | `getChoices`

Topics

"Create Variants"

Introduced in R2019a

systemcomposer.openModel

Open a 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.

Input Arguments

`modelName` — Name of new model
string

Model must exist on the MATLAB path.

Example: 'new_arch'

Data Types: `char` | `string` | `Model`

Output Arguments

`model` — Model handle
Model object

Examples

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

See Also

`createModel` | `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 a model
Model object (default)

Data Types: `systemcomposer.arch.Model`

See Also

Introduced in R2019b

removeComponent

Remove a component from a view

Syntax

```
removeComponent(object, compObj, contextView)
```

Description

`removeComponent(object, compObj, contextView)` removes the component with the specified path from the view given by the parameter 'contextView'.

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

Input Arguments

`object` — <argument purpose>
systemcomposer.view.ViewArchitecture (default)

<argument description>

`compObj` — <argument purpose>
<argument value> (default) | <argument value>

Path to the component including the name of the top-model.

`contextView` — <argument purpose>
systemcomposer.view.ViewArchitecture (default) | <argument value>

<argument description>

See Also

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,'newsignal');
element = addElement(interface,'newelement','Type','double);
removeElement(interface,'newsignal')
```

Input Arguments

`interface` — interface object
signal interface

Data Types: `systemcomposer.interface.SignalInterface`

`elementName` — Name of the element to be removed
String

Data Types: `char`

See Also

`addElement` | `getElement`

Topics

"Define Interfaces"

Introduced in R2019a

removeInterface

Remove a named interface from an interface dictionary

Syntax

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

Description

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

Examples

Remove an Interface

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

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

Input Arguments

`dictionary` — Data dictionary attached to the architecture model
System Composer dictionary

Data Types: `systemcomposer.interface.Dictionary`

`name` — Name of the new interface
string

Data Types: `char`

See Also

`addInterface` | `getInterface` | `getInterfaces`

Topics

"Define Interfaces"

Introduced in R2019a

removeProfile

Remove profile from a model

Syntax

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

Description

`removeProfile(modelObject,profileFile)` applies the profile to a model and makes all of the constituent stereotypes available.

Examples

Remove a Profile

```
removeProfile(arch,'SystemProfile')
```

Input Arguments

`modelObject` — Architecture model object
architecture model

Data Types: `systemcomposer.arch.Model`

`profileFile` — Profile file
string

Name of a profile attached to the model.

Data Types: `string`

See Also

`applyProfile` | `createProfile`

Topics

“Define Profiles and Stereotypes”

Introduced in R2019a

removeProperty

Remove a property from a 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.

```
stype = addStereotype(profile,'electricalComponent','AppliesTo','Component')
property = addProperty(stype,'VoltageRating','DefaultValue','5');
removeProperty(stype,'VoltageRating');
```

Input Arguments

`stereotype` — Stereotype to which the property is added
stereotype

`propertyName` — Property to be removed
string

See Also

`addProperty`

Topics

“Define Profiles and Stereotypes”

Introduced in R2019a

removeStereotype

Remove a stereotype from a model element

Syntax

```
removeStereotype(element,stereotype)
```

Description

`removeStereotype(element,stereotype)` removes a stereotype from the mode element. Removes the specified stereotype if already applied to a model element.

Input Arguments

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

The stereotype and all its properties are removed from this element.

Data Types: `systemcomposer.arch.Element`

`stereotype` — Reference stereotype
stereotype

The stereotype must be specified in the form `<profile>.<stereotype>`.

Data Types: `systemcomposer.internal.profile.Stereotype`

See Also

`applyStereotype`

Topics

“Remove a Stereotype”

Introduced in R2019a

reparent

Move stereotype

Syntax

```
reparent(stereotype,parentStereotype)
```

Description

`reparent(stereotype,parentStereotype)` reparents the stereotype to the specified stereotype.

Examples

Reparent a Property

Add an architecture stereotype and reparent it to a component.

```
stype = addStereotype(profile,'electricalComponent','systemcomposer.Architecture','Gene
reparent(stype,'systemcomposer.Component')
```

Input Arguments

`stereotype` — Stereotype whose inheritance changes
stereotype

`parentStereotype` — the new stereotype to inherit from
stereotype

See Also

Introduced in R2019a

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` — The architecture model
System Composer architecture

Data Types: `systemcomposer.arch.Model`

`dictionary` — Data dictionary attached to the architecture model
System Composer dictionary

Data Types: `systemcomposer.interface.Dictionary`

See Also

`loadModel`

Topics

“Create an Architecture Model”
“Save and Link Interfaces”

Introduced in R2019a

saveAsModel

Save the Architecture to a 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 a Component

Save the component `robotcomp` in `Robot.slx` and reference the model.

```
saveAsModel(robotcomp,'Robot');
```

Input Arguments

`component` — Architecture component
architecture component

The component must have an architecture with definition type `composition`. For other definition types, this function gives an error.

Data Types: `systemcomposer.arch.Component`

`modelName` — Model name
string

Data Types: `char`

See Also

`inlineComponent` | `linkToModel`

Topics

“Decompose and Reuse Components”

Introduced in R2019a

saveInstance

Save an architecture instance

Syntax

```
saveInstance(architectureInstance,fileName)
```

Description

`saveInstance(architectureInstance,fileName)` saves an architecture instance to a MAT-file.

Input Arguments

`architectureInstance` — The architecture instance
architecture instance

The architecture instance to be saved.

Data Types: `systemcomposer.analysis.ArchitectureInstance`

`fileName` — File to save the instance
string

This is a MAT-file to save the architecture instance.

See Also

`loadInstance`

Topics

“Write Analysis Function”

Introduced in R2019a

setActiveChoice

Set the active choice in the variant component

Syntax

```
setActiveChoice(variantComponent,choice)
```

Description

`setActiveChoice(variantComponent,choice)` sets the active choice on the variant component.

Input Arguments

`variantComponent` — Architecture component
component

Variant component with multiple choices.

Data Types: `systemcomposer.arch.Component`

`choice` — Choice in a variant component
component | string

The choice whose control string is returned by this function. This can be a component object or label of the variant choice.

Data Types: `systemcomposer.arch.Component` | `string`

See Also

`addChoice` | `getActiveChoice` | `getChoices`

Topics

“Create Variants”

Introduced in R2019a

setCondition

Set the condition on the variant choice

Syntax

```
setCondition(variantComponent,choice, expression)
```

Description

`setCondition(variantComponent,choice, expression)` sets the variant control for a choice for the variant component.

Input Arguments

`variantComponent` — Architecture component
component

Variant component with multiple choices.

Data Types: `systemcomposer.arch.Component`

`choice` — Choice in a variant component
component | string

The choice whose control string is set by this function.

Data Types: `systemcomposer.arch.Component`

`expression` — The control string
string

The control string that controls the selection of the `choice`.

See Also

`getCondition` | `makeVariant` | `setActiveChoice`

Topics

“Create Variants”

Introduced in R2019a

setProperty

Set the property value corresponding to a stereotype applied to the 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
architecture component | architecture port | architecture connector

Data Types: `systemcomposer.arch.Element`

`propertyName` — Name of the property
stereotype.property

Qualified name of the property in the form `'<stereotype>.<property>'`.

Data Types: `char`

`propertyValue` — Value of the property
string

Specify numeric values in single quotes. Specify string values in the `sprintf("'%s'",'<property value>')` form. See example on this page.

Data Types: `char`

`propertyUnits` — Units of the property
string

Specify the units to interpret property values.

Data Types: `char`

See Also

`getProperty`

Topics

“Set Tags and Properties for Analysis”

Introduced in R2019a

setValue

Set the value of a property for an 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` — The element instance

architecture instance | component instance | port instance | connector instance

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.

Data Types: `systemcomposer.analysis.ArchitectureInstance` | `systemcomposer.analysis.ComponentInstance` | `systemcomposer.analysis.PortInstance` | `systemcomposer.analysis.ConnectorInstance`

`property` — The property field
stereotype.property

String in the form `<stereotype>.<property>`.

Data Types: `string`

See Also
`getValue`

Topics
“Write Analysis Function”

Introduced in R2019a

unlinkDictionary

Unlink dictionary from a 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

architecture

The model from which the dictionary link is to be removed.

Data Types: `systemcomposer.arch.Model`

See Also

`linkDictionary`

Topics

“Save and Link Interfaces”

Introduced in R2019a

updateInstance

Update an architecture instance

Syntax

```
updateInstance(architectureInstance,updateFlag)
```

Description

`updateInstance(architectureInstance,updateFlag)` updates an instance to mirror the changes in the specification model.

Input Arguments

`architectureInstance` — The architecture instance
architecture instance

The architecture instance to be updated.

Data Types: `systemcomposer.analysis.ArchitectureInstance`

`updateFlag` — whether to update values changed directly in the model
1 | 0

If true, the method reflects changes made directly in the specification model to the instance model.

See Also

`loadInstance` | `saveInstance`

Topics

“Write Analysis Function”

Introduced in R2019a

Classes — Alphabetical List

systemcomposer.analysis.Instance

Class that represents an architecture model element in an 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 the instance
string

This is the name of the instance.

Data Types: `char`

`Specification` — The specification that the instance is created from
architecture | component | port | connector

Every instance has a specification from which it took its form. The kind of the specification depends on the kind of the instance.

Data Types: `systemcomposer.arch.Architecture` | `systemcomposer.arch.Component` | `systemcomposer.arch.Port` | `systemcomposer.arch.Connector`

Architecture Instance Properties

`Components` — Child components of the instance
array of components

The components within the architecture.

Data Types: `systemcomposer.analysis.ComponentInstance`

`Ports` — Ports of the architecture instance
array of ports

These are the architecture ports that belong to the architecture instance.

Data Types: `systemcomposer.analysis.PortInstance`

`Connectors` — Connectors in the architecture instance
array of connectors

These are the connectors within the architecture, connecting child components.

Data Types: `systemcomposer.analysis.Connectors`

Component Instance Properties

`Components` — Child components of the instance
array of components

The components within the architecture.

Data Types: `systemcomposer.analysis.ComponentInstance`

`Ports` — Ports of the architecture instance
array of ports

These are the architecture ports that belong to the architecture instance.

Data Types: `systemcomposer.analysis.PortInstance`

`Connectors` — Connectors in the architecture instance
array of connectors

These are the connectors within the architecture, connecting child components.

Data Types: `systemcomposer.analysis.Connectors`

`Parent` — Parent of the component
component

The architecture that contains the component

Data Types: `systemcomposer.analysis.Architecture`

Port Instance Properties

`Parent` — Parent of the port
component

The component that contains the port

Data Types: `systemcomposer.analysis.Component`

Connector Instance Properties

`Parent` — Parent of the connector
component

The component that contains the connector

Data Types: `systemcomposer.analysis.Component`

`SourcePort` — Source port
port

The port from which the connector originates.

Data Types: `systemcomposer.analysis.Port`

`DestinationPort` — Destination port
port

The port from which the connector ends.

Data Types: `systemcomposer.analysis.Port`

Object Functions

deleteInstance	Delete an architecture instance
getValue	Get value of a property from an element instance
instantiate	Create an analysis instance from a specification
isArchitecture	Find if an instance is a architecture instance
isComponent	Find if an instance is a component instance

isConnector	Find if an instance is a connector instance
isPort	Find if an instance is a port instance
loadInstance	Load an architecture instance
saveInstance	Save an architecture instance
setValue	Set the value of a property for an element instance
updateInstance	Update an architecture instance

See Also

Topics

“Write Analysis Function”

Introduced in R2019a

systemcomposer.arch.Architecture

Class that represents an architecture in an 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 an model and get the root architecture:

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

Properties

`Name` — Name of the architecture
character vector

The architecture name is derived from the parent component or model name to which the architecture belongs.

Example: 'system_architecture'

`Definition` — Definition type of the architecture
Composition | Behavior | View

The definition type can be a composition, a behavior model, or a view.

Example: Composition

Data Types: `ArchitectureDefinition enum`

`Parent` — Handle to the parent component that owns this Architecture
systemcomposer.arch.Component object

`Components` — Array of handles to the set of child components of this architecture
array of systemcomposer.arch.Component objects

`Ports` — Array of architecture ports of this architecture
array of systemcomposer.arch.ArchitecturePort objects

`Connectors` — Array of connectors that either interconnect child components or connect child components to architecture ports
array of systemcomposer.arch.Connector objects

Object Functions

addComponent	Add a component to the architecture
addVariantComponent	Add a component to the architecture
addPort	Add ports to architecture
connect	Connect pairs of components

See Also

`systemcomposer.arch.Component`

Topics

“Create an Architecture Model”

Introduced in R2019a

systemcomposer.arch.ArchitecturePort

Represent an architecture port

Description

This class inherits from systemcomposer.arch.BasePort.

Properties

—
(default) |

Example:

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char` | `string` | `struct` | `table` | `cell` | `function_handle` | `categorical` | `datetime` | `duration` | `calendarDuration` | `fi`
Complex Number Support: Yes

Object Functions

Examples

See Also

Introduced in R2019a

systemcomposer.arch.BaseComponent

Base component for interface

Description

The class inherits from systemcomposer.base.BaseElement and implements the interface systemcomposer.base.BaseComponent.

Properties

`Name` — Name of component
character vector

Get or set name of component.

Example: name = get(obj, 'Name'); set(obj, 'Name', name)

`Parent` — Handle to Architecture
character vector

Get a handle to the Architecture that owns this Component. The returned object is of type systemcomposer.arch.Architectute.

Example: parent= get(obj, 'Parent')

`Architecture` — Architecture of Component
character vector

Get the Architecture of this Component in the composition. For a Component that references a different System Composer model, this will return a handle to the root Architecture of that model. For Variant Components, the Architecture is that of the active Variant,

Example: arch = get(obj, 'Architecture')

`OwnedArchitecture` — Architecture that Component owns
character vector

Get the Architecture that this Component directly owns in the composition. For Components that reference an Architecture, this will be empty. For Variant Components, this will return the Architecture in which the individual Variant Components reside,

Example: arch = get(obj, 'OwnedArchitecture')

`Ports` — Array of Component ports
character vector

Get an array of Component ports for this component in the composition.

Example: ports = get(obj, 'Ports')

`OwnedPorts` — Array of Component ports
character vector

Get an array of Component ports for this component in the composition only if this component is not referencing an architecture.

Example: ports = get(obj, 'OwnedPorts')

Object Functions

Examples

See Also

Introduced in R2019b

systemcomposer.arch.BasePort

Base class of both architecture and component ports

Description

The `BasePort` class is the base class for all ports, both architecture ports and component ports. This class is derived from `systemcomposer.arch.Element`. This class inherits from systemcomposer.base.BaseElement and implements the interface systemcomposer.base.BasePort.

Creation

Create a port.

`addPort`

Properties

`Name` — Name of port
string

`Direction` — Port direction
`'Input'` | `'Output'`

`Interface` — Interface attached to the port
signal interface

Data Types: `systemcomposer.interface.SignalInterface`

Object Functions

connect Connect pairs of components

See Also

`systemcomposer.arch.Element`

Topics

"Ports"

Introduced in R2019a

systemcomposer.arch.Component

Class that represents a 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

`ParentArchitecture` — Handle to the parent component that owns this component
Architecture object

Data Types: `systemcomposer.arch.Architecture`

`Architecture` — Architecture that defines the component structure
Architecture object

For a component that references a different architecture model, this returns a handle to the root architecture of that model. For variant components, the architecture is that of the active variant.

Data Types: `systemcomposer.arch.Architecture`

`OwnedArchitecture` — The architecture that this component directly owns
architecture

For components that reference an architecture, this is be empty. For variant components , this is the architecture in which the individual variant components reside.

Data Types: `systemcomposer.arch.Architecture`

`Ports` — Array of component ports
array of ports

Data Types: `systemcomposer.arch.ComponentPort`

`OwnedPorts` — Array of component ports
array of ports

For all components except Variant View components, this will return the same value as `Ports`. For Variant View components, this returns the aggregate of all ports across all Views in which this component is present.

Data Types: `systemcomposer.arch.ComponentPort`

`ReferenceName` — If linked component, the name of the model that the component references
string

Data Types: `char`

Object Functions

saveAsModel	Save the Architecture to a separate model
createSimulinkBehavior	Create a Simulink model and link component to it
linkToModel	Link component to a model
inlineComponent	Inline reference architecture into model
connect	Connect pairs of components

See Also

`systemcomposer.arch.Architecture`

Topics

“Create an Architecture Model”

Introduced in R2019a

systemcomposer.arch.ComponentPort

Represent a component port

Description

This class inerits from systemcomposer.arch.BasePort.

Properties

—
(default) |

Example:

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char` | `string` | `struct` | `table` | `cell` | `function_handle` | `categorical` | `datetime` | `duration` | `calendarDuration` | `fi`
Complex Number Support: Yes

Object Functions

Examples

See Also

Introduced in R2019a

systemcomposer.arch.Connector

Class that represents a connector between ports

Description

The `connector` class represents a connectore 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

`ParentArchitecture` — Handle to the parent component that owns this component
Architecture object

Data Types: `systemcomposer.arch.Architecture`

`SourcePort` — Source of the connection
architecture port | component port

The source port is an output port.

`DestinationPort` — Destination of the connection
architecture port | component port

The destination port is an input port.

`Direction` — Port direction
`'Input'` | `'Output'`

`Interface` — Interface attached to the port
signal interface

Data Types: `systemcomposer.interface.SignalInterface`

Object Functions

See Also

`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 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` — Unique identifier for a model element
character vector

<property description>

Example: '91d5de2c-b14c-4c76-a5d6-5dd0037c52df'

`ExternalUID` — External identifier
character vector

Set an external ID that is preserved over the lifespan of the element. The external ID is preserved through all operations that preserve the UUID.

Example: 'network_connector_01'

`Model` — Handle to the parent System Composer model of the element
systemcomposer.arch.Model object

<property description>

Example: <property example>

`SimulinkHandle` — Simulink handle for Architecture element
`'SimulinkHandle'`

Simulink handle for Architecture element. This property is necessary for several Simulink related workflows and for using Simulink Requirement APIs.

Example: name = get(object, 'SimulinkHandle')

Object Functions

applyStereotype	Apply a stereotype to a model element
getStereotypes	Get the stereotypes applied on the element
removeStereotype	Remove a stereotype from a model element
setProperty	Set the property value corresponding to a stereotype applied to the element
getProperty	Get the property value corresponding to a stereotype applied to the element
destroy	Remove and destroy a model element

See Also

`systemcomposer.arch.BasePort` | `systemcomposer.arch.Component` | `systemcomposer.arch.Connector`

Topics

“Create an Architecture Model”

Introduced in R2019a

systemcomposer.arch.Model

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

Properties

`Name` — Name of a model
character vector | string

Data Types: `char` | `string`

`Architecture` — Root architecture of a System Composer model
`Architecture` object

Data Types: `systemcomposer.arch.Architecture`

`SimulinkHandle` — Handle
real number

Handle to the Simulink representation of the System Composer model.

Data Types: `double`

`Profiles` — Array of handles to profiles
array of `Profile` objects

Array of handles to profiles attached to the model.

Data Types: `systemcomposer.internal.profile.Profile`

`InterfaceDictionary` — Dictionary object that holds interfaces
`Dictionary` object

`Dictionary` object that holds interfaces. If the model is not linked to an external dictionary, this is a handle to the implicit dictionary

Data Types: `systemcomposer.interface.Dictionary`

`Views` — Array of handles to model views
array of `ViewArchitecture` objects

Array of handles to model views.

Example: `objViewArchitecture = get(objModel, 'Views')`

Data Types: `systemcomposer.view.ViewArchitecture`

Methods

open	Open System Composermodel
save	Save the architecture model or data dictionary
applyProfile	Apply profile to a model
removeProfile	Remove profile from a model
linkDictionary	Link data dictionary to an architecture model
unlinkDictionary	Unlink dictionary from a model
lookup	Lookup an architecture element
openViews	Open architecture views editor
find	Find model elements
createViewArchitecture	Create a view

See Also

Topics

“Create an Architecture Model”

Introduced in R2019a

Represent a variant component

Description

This class inherits from systemcomposer.arch.BaseComponent.

Properties

—
(default) |

Example:

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char` | `string` | `struct` | `table` | `cell` | `function_handle` | `categorical` | `datetime` | `duration` | `calendarDuration` | `fi`
Complex Number Support: Yes

Object Functions

Examples

See Also

Introduced in R2019a

systemcomposer.base.BaseArchitecture

Interface class which defines the common properties and methods

Description

Interface class which defines the common properties and methods of an Architecture. This interface is implemented by arch.Architecture and view.ViewArchitecture. The descriptions of the properties and methods are given in the classes who implement.

Properties

`Name` — <property purpose>
character vector

<property description>

Example: name = get(object, 'Name'); set(object, 'Name', name)

`Parent` — <property purpose>
<property type>

<property description>

Example: parentComponents = get(object, 'Parent')

`Components` — <property purpose>
<property type>

<property description>

Example: childComponents = get(object,'Components')

`Ports` — <property purpose>
<property type>

<property description>

Example: ports = get(object,'Components')

Examples

See Also

Introduced in R2019b

systemcomposer.base.BaseComponent

Interface class which defines the common properties and methods

Description

Interface class which defines the common properties and methods of a Component. This interface is implemented by arch.BaseComponent and view.BaseViewComponent. The descriptions of the properties and methods are given in the classes who implement.

Properties

`Name` — <property purpose>
character vector

<property description>

Example: name = get(object, 'Name'); set(object, 'Name', name)

`Parent` — <property purpose>
<property type>

<property description>

Example: parent = get(object, 'Parent')

`Ports` — <property purpose>
<property type>

<property description>

Example: ports = get(object,'Components')

`Architecture` — <property purpose>
<property type>

<property description>

Example: architecture = get(object,'Architecture')

`OwnedArchitecture` — <property purpose>
<property type>

<property description>

Example: ownedArchitecture = get(object,'OwnedArchitecture')

Examples

See Also

Topics
“Creating Architecture Views Interactively”

Introduced in R2019b

systemcomposer.base.BaseConnector

Interface class which defines the common properties and methods of a Connector

Description

Interface class which defines the common properties and methods of a Connector. This interface is implemented by arch.Connector and view.ViewConnector. The descriptions of the properties and methods are given in the classes who implement.

Properties

`Parent` — <property purpose>
<property type>

<property description>

Example: parent = get(object, 'Parent')

`SourcePort` — <property purpose>
<property type>

<property description>

Example: sourcePort = get(object, 'SourcePort')

`DestinationPort` — <property purpose>
<property type>

<property description>

Example: ports = get(object,'DestinationPort')

Examples

See Also

Topics

“Creating Architecture Views Interactively”

Introduced in R2019b

systemcomposer.base.BaseElement

Base class for all architecture model elements

Description

Base class for all architecture model elements in both systemcomposer.arch and systemcomposer.view packages.

Properties

`UUID` — Unique identifier for the architecture model element

character vector

Get a unique identifier for the architecture model element.

Example: uuid = get(object, 'UUID'

`ExternalUID` — External ID that is preserved

character vector

This is an external ID that can be set on the object that is then preserved over its lifespan. It is guaranteed that this ID will be preserved through all operation that preserve the UUID.

Example: uid = get(object, 'ExternalUID')

Example: set(object, 'ExternalUID', uid)

`Model` — Parent model

character vector

Returns the parent System Composer model of this model element.

Example: m = get(object,'Model')

Object Functions

applyStereotype	Apply a stereotype to a model element
removeStereotype	Remove a stereotype from a model element
getStereotypes	Get the stereotypes applied on the element
getProperty	Get the property value corresponding to a stereotype applied to the element
setProperty	Set the property value corresponding to a stereotype applied to the element
destroy	Remove and destroy a model element

Examples

See Also

Introduced in R2019b

systemcomposer.base.BasePort

Interface class which defines the common properties and methods of a Port

Description

Interface class which defines the common properties and methods of a Port. This interface is implemented by arch.BasePort and view.BaseViewPort. The descriptions of the properties and methods are given in the classes who implement.

Properties

`Name` — <property purpose>
character vector

<property description>

Example: name = get(object, 'Name'); set(object, 'Name', name)

`Parent` — <property purpose>
<property type>

<property description>

Example: parent = get(object, 'Parent')

`Direction` — <property purpose>
<property type>

<property description>

Example: direction = get(object,'Direction')

`Interface` — <property purpose>
<property type>

<property description>

Example: interface = get(object,'Interface')

`Connectors` — <property purpose>
<property type>

<property description>

Example: connectors = get(object,'Connectors')

`Connected` — <property purpose>
<property type>

<property description>

Example: connected = get(object,'Connected')

Object Functions

Examples

See Also

Introduced in R2019b

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 signal element.

```
dictionary = <architecture>.InterfaceDictionary;
```

Properties

`Interfaces` — Interfaces defined in the dictionary
array of signal interfaces

Data Types: `systemcomposer.interface.Dictionary`

`UUID` — Unique identifier
string

Object Functions

addInterface	Create a named interface in an interface dictionary
removeInterface	Remove a named interface from an interface dictionary
getInterface	Get the object for a named interface in an interface dictionary
getInterfaces	Get the object for a named interface in an interface dictionary

See Also

`systemcomposer.interface.SignalElement`

Topics

“Define Interfaces”

Introduced in R2019a

systemcomposer.interface.SignalElement

Class that represents an element in the signal interface

Description

The `SignalElement` class represents a single element in the signal interface

Creation

Create a signal element.

`addElement(interface,elementName)`

Properties

`Interface` — Handle to the parent interface of the element
Interface object

Data Types: `systemcomposer.interface.SignalInterface`

`Name` — Element name
string

`Dimensions` — Dimensions of the element
array of positive integers

`Type` — Data type of the element
string

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

`Units` — Units of the element
string

`Minimum` — Minimum value for the element
double

`Maximum` — Maximum value for the element
double

`Description` — Description text for the element
string

Object Functions

destroy Remove and destroy a model element

See Also

`addInterface`

Topics

“Define Interfaces”

Introduced in R2019a

systemcomposer.interface.SignalInterface

Class that represents the structure of the signal interface

Description

The `SignalInterface` class represents the structure of the signal interface at a given port

Creation

Create an interface.

```
interface = addInterface(architecture, name)
```

Properties

`Dictionary` — Handle to the parent dictionary of the interface
Interface dictionary object

Data Types: `systemcomposer.interface.Dictionary`

`Name` — Interface name
string

`Elements` — Elements in interface
array of interface elements

Object Functions

addElement	Add a signal interface element
removeElement	Remove a signal interface element
getElement	Get the object a signal interface element
destroy	Remove and destroy a model element

See Also

`systemcomposer.interface.SignalInterface`

Topics

“Define Interfaces”

Introduced in R2019a

systemcomposer.io.ModelBuilder

Model builder for System Composer architecture models

Description

Build System Composermodels 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.

Components	Description
`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` `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 specifiedas `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 th
               profile UAVComponent under Stereotype PartDescriptor maps to Domain
# Kind       : Mapped as component property. Property "ModelName" defined in the
               profile UAVComponent under Stereotype PartDescriptor maps to Kind v
# InterfaceName : If class is of port type. InterfaceName maps to name of the inte
# 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.

```
definitionContents = readtable('DataDefinitions.xlsx');

% 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, dimer
    end
end
```

Step 3. Build Architecture Specifications.

Architecture specifications de Create MATLAB tables from source Excel file.

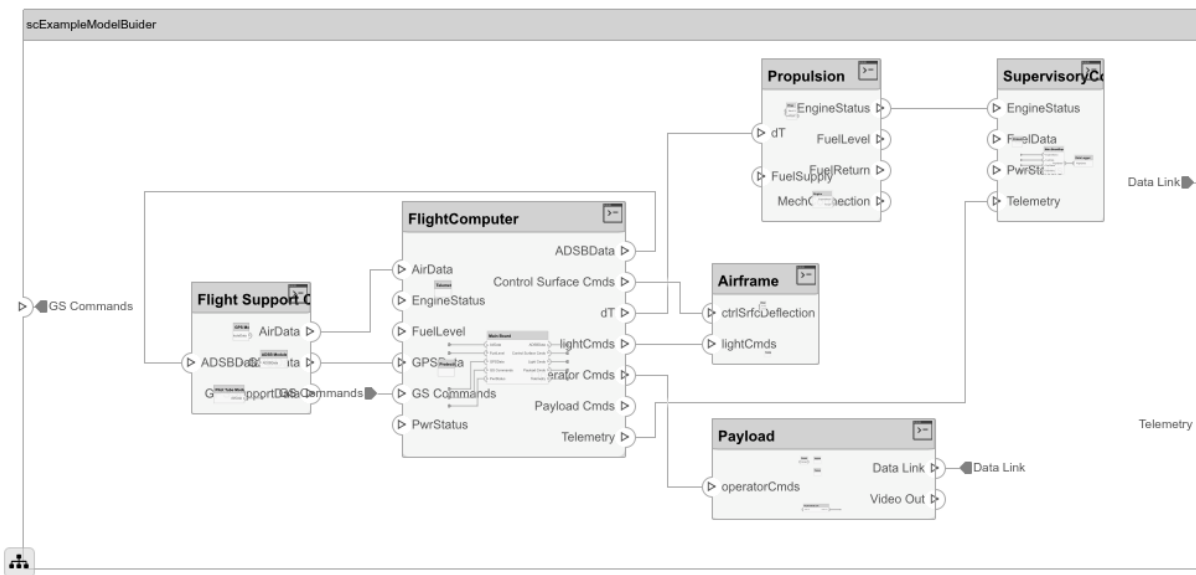
```
excelContents = readtable(architectureFileName);
% Iterate over each row in table.
for rowItr =1:numel(excelContents(:,1))
% Read each row of the excel file and columns.
    class = excelContents.Class(rowItr);
```

```
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
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

“Importing and Exporting Architecture Models”

Introduced in R2019b

systemcomposer.profile.Profile

Class that represents a profile

Description

The `Profile` class represents architecture profiles.

Creation

```
profiles = <architecture>.Profiles;
```

Properties

`Name` — Name of the profile
string

Data Types: `char`

`Description` — Description text for the profile
string

Data Types: `char`

Object Functions

addStereotype	Add a stereotype to the profile
removeStereotype	Remove a stereotype from a model element

See Also

`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 the property
string

Data Types: `char`

`Name` — Property name
string

Data Types: `char`

`Datatype` — Property data type
valid data type string

Data Types: `char`

`Dimensions` — Dimensions of property
positive integer array

Data Types: `char`

`Min` — Minimum value
numeric value

Data Types: `double`

`Max` — Maximum value
numeric value

Data Types: `double`

`Units` — Property units
string

Data Types: `char`

Object Functions

destroy Remove and destroy a model element

See Also

`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 the stereotype
string

Data Types: `char`

`Description` — Description text for the stereotype
string

Data Types: `char`

`Icon` — Icon for the stereotype
string

Data Types: `char`

`Parent` — The stereotype from which this stereotype inherits its properties
stereotype

Data Types: `systemcomposer.profile.Stereotype`

`AppliesTo` — The element type to which this stereotype can be applied
stereotype

Data Types: `systemcomposer.profile.Stereotype`

`Abstract` — Whether the stereotype is abstract
true | false

If true then stereotype cannot be directly applied on model elements, but instead serves as a parent for other stereotypes.

`Properties` — Array of property definitions owned or inherited by this stereotype
stereotype

Data Types: `systemcomposer.profile.Stereotype`

Object Functions

addProperty	Add a property to a stereotype
removeProperty	Remove a property from a stereotype
reparent	Move stereotype

See Also

`systemcomposer.profile.Stereotype`

Topics

“Define Profiles and Stereotypes”

Introduced in R2019a

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 the view component
character vector | string

Name of the view component.

Example: `name = get(objBaseViewComponent, 'Name');`
`set(objBaseViewComponent, 'Name', name)`

`Parent` — Handle to parent view architecture of this component
`ViewArchitecture` object

Handle to the parent view architecture of this component.

Example: `parent = get(objBaseViewComponent, 'Parent')`

`Architecture` — Handle to view architecture of this component
`ViewArchitecture` object

Handle to the view architecture of this component.

Example: `p = get(objBaseViewComponent, 'ViewArchitecture')`

Examples

See Also

Introduced in R2019b

systemcomposer.view.ComponentOccurrence

Shadow of a component from the composition in a view

Description

This class inherits from `systemcomposer.view.BaseViewComponent`.

Properties

`Component` — Handle to the composition
`systemcomposer.arch.BaseComponent` object

Handle to the composition Component of this occurrence.

Example: `get(object, 'Component')`

See Also

Introduced in R2019b

systemcomposer.view.ViewArchitecture

View components in an 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 the architecture
character vector | string

Architecture name derived from the parent component or model name to which the architecture belongs.

Example: `name = get(objViewArchitecture, 'Name')`

`IncludeReferenceModels` — Control inclusion of referenced models
true | false

Control inclusion of referenced models.

Example: `tf = get(objViewArchitecture, 'IncludeReferenceModels')`

`Color` — Color of the view architecture
character vector | string

Color of the view architecture, specified as a character vector or string (for example, `'blue'`, `'black'`, `'green'`) or RGB value encoded in a hexadecimal string (for example, `'#FF00FF'`, `'#DDDDDD'`). An invalid color string results in an error.

Example: `color = get(objViewArchitecture, 'Color')`

`Description` — Description of the view architecture
character vector | string

Description of the view architecture.

Example: `description = get(objViewArchitecture, 'Description');`
`set(objViewArchitecture, 'Description', description)`

`Parent` — Component that owns the view architecture
`systemcomposer.view.BaseViewComponent` object

Handle to the component that owns this view architecture. The returned object is of type `systemcomposer.view.BaseViewComponent`. For a root view architecture, returns an empty handle.

Example: `parentComponent = get(objViewArchitecture, 'Parent')`

`Components` — Array of handles to child components
array of `systemcomposer.base.BaseViewComponents` objects

Array of handles to the set of child components of this view Architecture.

Example: `childComponents = get(objViewArchitecture, 'Components')`

Methods

addComponent	Add component to view given path
createViewComponent	
removeComponent	Remove a component from a view

Examples

See Also

Introduced in R2009b

systemcomposer.view.ViewComponent

View component within an 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

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 (default) | string

Gets the identifier of an object. Used by Simulink Requirements.

Example: identifier = get(objViewElement, 'ZCIdentifier')

Examples

See Also

Introduced in R2009b

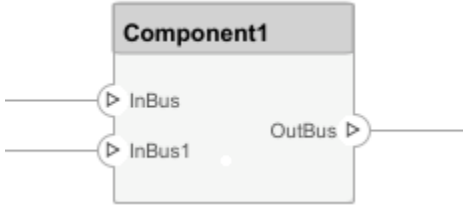
Blocks — Alphabetical List

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”

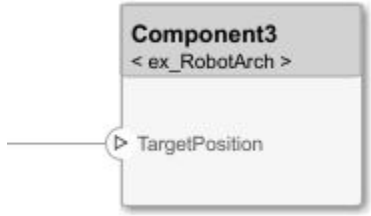
Introduced in R2019a

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”

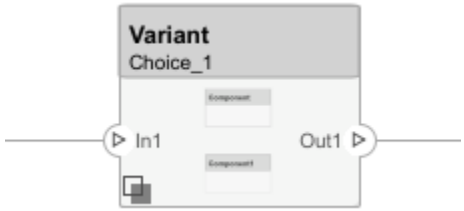
Introduced in R2019a

Variant Component

Add components with alternative designs

Description

Use a Variant Component block to create multiple deign 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

You can have different interface definitions assigned to the source port and destination port of a connection. This could represent an intermediate point in design, where components from different sources are brought together. Use an Adapter block to connect components with different interfaces.

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