Perspective Component Methods

A component method is a function that is defined on a component object. For example, this is how we would call a component method defined on the component object **self**:

```
output = self.myMethod(param1,param2)
```

Perspective has a variety of component methods that are defined on *all* components, and it also offers you the ability to configure your own using custom methods.

On this page

...

Object Traversal

In most Perspective scripts, you are given a reference to a component object (often in the form of a **self** parameter). The object is given in component scripts, but has several methods and properties associated with it to help traverse to the other objects in a Perspective View or get values from the Session.

Object Traversal is limited to a single view. If a script needs to reference a component in a different view, or there is a possibility that the hierarchy of the view will change, then Message Handling should be utilized instead.

Component/Container		
Method /Property	**Description**	**Example**
.parent	Calling this property will move up the component hierarchy, accessing the parent container of the preceding object. Root containers will return the view, and views/sessions will return None.	`self.parent`
.getParent()	This is the same as .parent above.	`self.getParent()`

.view	Calling this from anywhere within a view will return the parent view of the object.	`self.view`
.page	Returns a page object associated with the page the current component is on. This feature is new in Ignition version **8.0.4** Click here to check out the other new features .close() can be called on the page object, and can accept a message string as a parameter.	`page=self.page pageID=page.props.pageId pagePath=page.props.path`

.session	Returns the current perspective session you are in. From this session you can get any of the existing attributes of the Session. Similar to the view.session object. This feature is new in Ignition version **8.0.3** Click here to check out the other new features .close() can be called on the session object, and can accept a message string as a parameter. This feature is new in Ignition version **8.0.5** Click here to check out the other new features .getPages() returns a list of page objects. .getPage(string ID) returns the page associated with the given page ID if it exists. This feature is new in Ignition version **8.0.8** Click here to check out the other new features .getInfo() returns a list of session objects.	`session=self.session sesName=session.props.gateway.address sesProp=session.cu`

		stom.propertyName

.getChild (string)	Method that looks for a change component of a given name. Returns None if not found. String can either be the name of a child object, or a path to an object delimited by a forward slash, allowing you to move through multiple items in the hierarchy in a single call.	`self.getChild('Label_0')self.getChild('Container/Label_0')`

.getSibling(string)	Returns a reference to an object in the same container that the source component is located in. Similar to calling self.parent.getChild('component').	`self.getSibling('Label')`
View Objects		
.rootContainer	Returns the root container of the View.	`view.rootContainer`
.id	Returns the id of the View. This is a string that is similar to the path of the View. All instances of a View will have the same ID, for example: Perspective/Views/path/to/view yields path/to/view@C	`view.id`
.session		`session=view.sessi`

	Returns the current Perspective Session you are in. From this Session you can get any of the existing attributes of the Session. This is similar to the self.session object. **This feature is new in Ignition version 8.0.3** Click here to check out the other new features .close() can be called on the session object, and can accept a message string as a parameter. .getPages() returns a list of page objects. .getPage(string ID) returns the page associated with the given page ID if it exists. **This feature is new in Ignition version 8.0.8** Click here to check out the other new features .getInfo() returns a list of session objects.	`onsesName=session.gateway.addressesProp=session.custom.propertyName`

Object Traversal Examples

If you want to get other component properties in a view while scripting, you can use the above methods and properties to move around the View.

These examples assume you have the following structure/components in a View:

- View
    - Button 1
    - Text Field 1
    - Container 2
        - Text Field 2
    - Container 3
        - Text Field 3
        - Container 4
            - Button 4
            - Text Field 4

Scripting Example: Get component properties from a Button script

```
# this example code exists on the 'Button 1' in the above hierarchy.

# get to the view that the button is in
view = self.view

# get the text from 'Text Field 1'
text1 = self.getSibling('Text Field 1').props.text

# get the text from 'Text Field 2'
text2 = self.getSibling('Container 2').getChild('Text Field 2').props.text

# get the text from 'Text Field 3'
text3 = self.getSibling('Container 3').getChild('Text Field 3').props.text

# get the text from 'Text Field 4'. Either of these will work the same.
text4 = self.getSibling('Container 3').getChild('Container 4/Text Field 4').props.text
text4 = self.getSibling('Container 3').getChild('Container 4').getChild('Text Field 4').props.text
```

Scripting Example: Get component properties from a Button script

```
# this example code exists on the 'Button 4' in the above hierarchy.

# get to the view that the button is in
view = self.view

# get the text from 'Text Field 1'
text1 = self.parent.parent.getSibling('Text Field 1').props.text

# get the text from 'Text Field 2'
text2 = self.parent.parent.getSibling('Container 2').getChild('Text Field 2').props.text

# get the text from 'Text Field 3'
text3 = self.parent.getSibling('Text Field 3').props.text

# get the text from 'Text Field 4'
text4 = self.getSibling('Text Field 4').props.text
```

Built-In Methods

Perspective components contain several shared methods. This section details such methods. Note that some methods are only available to certain types of components. In these cases, the description for the method will state any limitations.

Refreshing Bindings

The `refreshBinding` function can be used to manually fire a binding, and is designed to be used on bindings that can poll (like query and Tag history bindings). In these instances, using refreshBinding in lieu of polling can save Gateway resources. The refreshBinding() function takes a string as a parameter, corresponding to the property that should be refreshed:

```
self.refreshBinding("props.data")
```

It is often useful to use refreshBinding() from a component message handler, since we can then refresh several applicable bindings via a single system.perspective.sendMessage call.

Requesting Focus

The `focus` method can be called by a component to request focus in a view. This is useful if you wish to control where keyboard input is directed after a particular action.

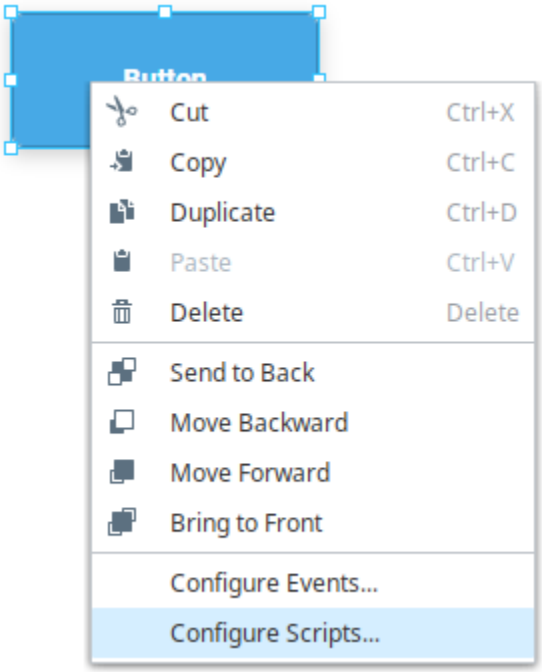
Due to the nature of focus, calling the `focus` method is only effective on components that can have focus. Input components such as the Text Field and Numeric Entry Field components can gain focus, but Display components like Labels and Images can not gain focus.

```
self.focus()
```

Custom Methods

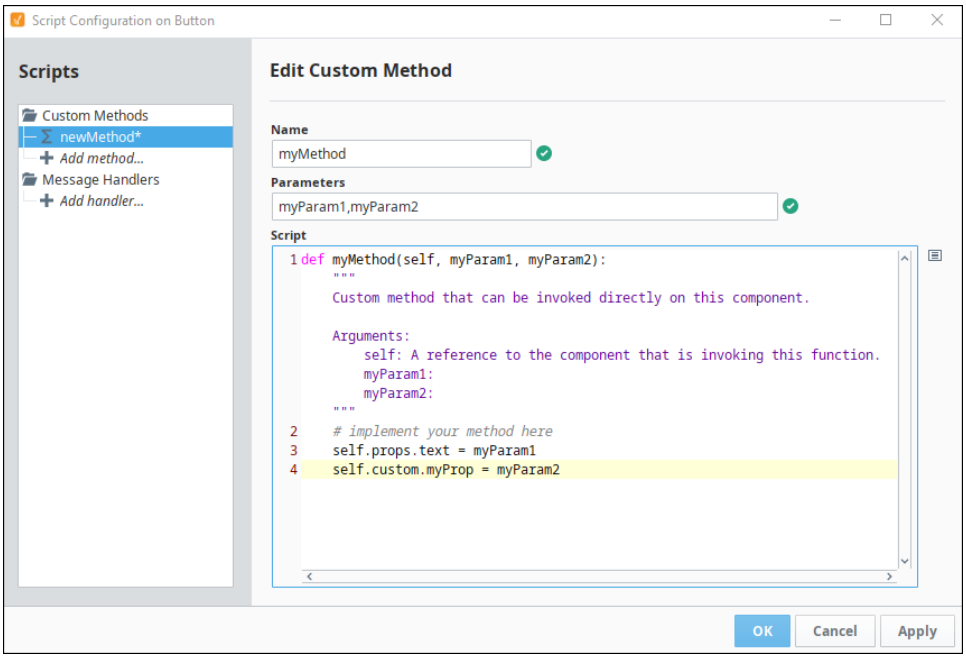
Perspective offers the option of configuring your own methods for a component. To configure a custom method:

1. **Right-click** on the desired component in the Designer, and select **Configure Scripts...**



2. Under **Custom Methods**, double click on **Add method...**
3. Enter a **Name** for your method, which will be used to call the method.
4. Enter any number of **Parameters** your method will need, separated by commas.
5. Add code to implement your method.

> A **self** object is provided in every custom method, but should *not* be provided as a parameter when calling the function.



6. Click **OK** to commit your method.
7. To call this method, use:

```
self.myMethod(param1,param2)
```

For example:

```
self.myMethod("Hi!","This is a test")
```

Related Topics ...

- Component Message Handlers