Focal Point®

Custom Business Rule Development and Usage Guide

Release 7.5.1

Publication information

FPNB-7510-00 (May 2022)

Information in this publication is subject to change. Changes will be published in new editions or technical newsletters.

Contents

1 Custom business rule development and usage

Focal Point helps you make value-based decisions for project, product, and portfolio management. Focal Point reduces the complexity of managing information from various sources such as emails, documents and spreadsheets and provides a collaborative, centralized environment for requirements, products, and project data.

The following topics are covered in this manual:

Business rules in Focal Point

Expressions and business rules are used to provide business intelligence to your data in Focal Point. Expressions are usually simple mathematical operators or functions whereas business rules are business logic written as Java modules that take several parameters to do complex tasks. Examples of such complex tasks are:

- implementing a complete workflow
- aggregating information through links
- rolling up information from a lower level to a higher level
- triggering automatic emails to stakeholders for approval

All Focal Point installations include a set of standard expressions and business rules. Administrators can use these when they configure and set up new workspaces. Apart from these standard business rules, some installations run custom business rules, which are developed and supported by the product development team.

Note This manual does not cover the standard business rules or custom business rules that are developed by the product development team.

Business rule API

In Focal Point 7.4 and later, you can add custom functionality to your Focal Point installation by using the business rule API to create your own business rules. Newly-developed business rules can be deployed into an existing Focal Point server by using the 'Business rule Administration' page in Focal Point.

This manual explains the basic concepts behind the business rules and provides a comprehensive guide for the development of new custom business rules in Focal Point using the business rule API.

Note Custom business rules that you create are not supported. Product support is limited to the business rule API and the business rules that are included in the standard installation.

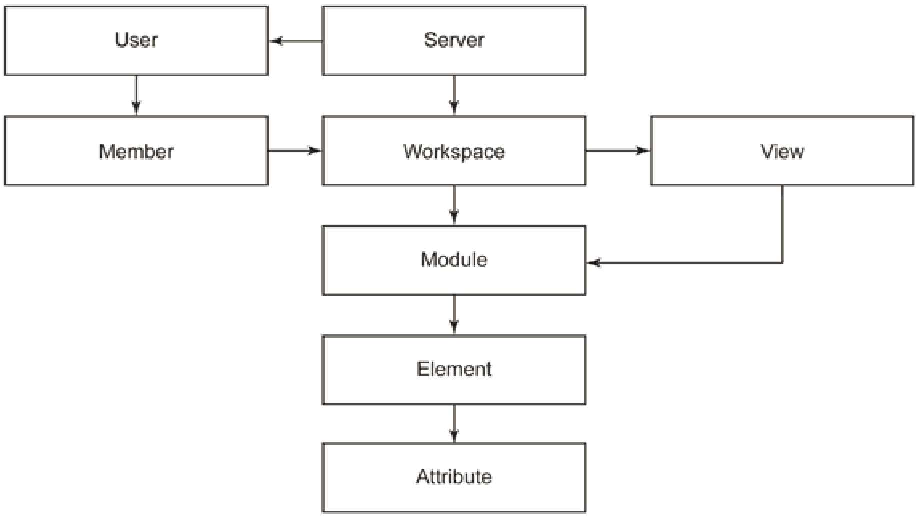
Focal Point data model

The Focal Point data model contains the following components:

- Focal Point server instance, which contains a set of workspaces.
- Workspaces, which contain a collection of related modules. One of the workspaces is system-defined and is used to store information about the users in Focal Point.
- Modules, which are used to define the elements as attribute definitions. Each module contains a set of elements.
- Elements, which contain a set of values for the attributes defined in a module.
- Views, which are a subset of elements and attributes within a module.
- Members, which are instances of global users in a workspace.

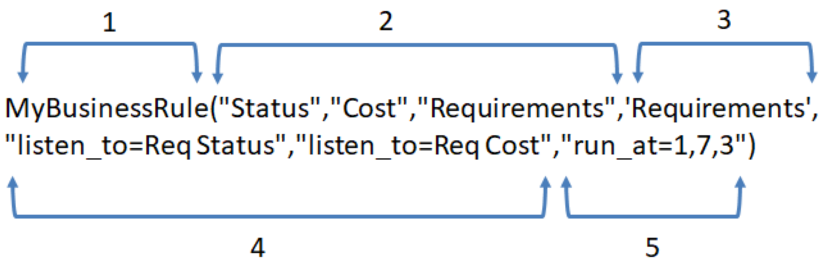
Users, views, and members are stored internally as elements in their respective modules.

The following diagram shows the relationships between the components:

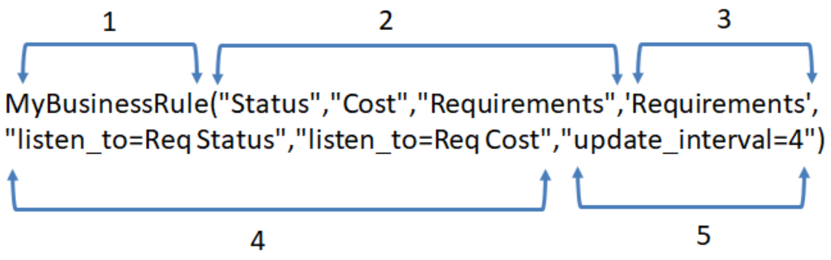


Structure of Business Rules

There are five components to a business rule and they must be specified in a particular order:



```
MyBusinessRule("Status","Cost","Requirements",'Requirements',
"listen_to=Req Status","listen_to=Req Cost","run_at=1,7,3")
```



```
MyBusinessRule("Status","Cost","Requirements",'Requirements',
"listen_to=Req Status","listen_to=Req Cost","update_interval=4")
```

1 A name for the business rule. This should be unique in a Focal Point instance.

2 Positional arguments that are enclosed in double quotes. There can be zero or more positional arguments each separated by a comma. These arguments are sent to the business rule during its execution and you can refer to them by using their position index.

3 Business rule listeners that are enclosed in single quotes. These are the names of the attributes on an element that should trigger the business rule whenever their values change. There can be zero or more listener arguments each separated by a comma. The list of listener attributes is sent to the business rule during its execution.

4 `listen_to` parameters that are enclosed in double quotes. If any of the listener attributes is a 'link' or 'linklist' attribute, then the business rule can also listen to attributes on the linked element(s). You specify the `listen_to` parameter by using the following format:

"listen_to=*attributename*"

where:

attributename is the name of the attribute on the link or linklist target.

Note There can be zero or more `listen_to` parameters each separated by a comma.

5 Timer parameters that are enclosed in double quotes. These parameters schedule the triggering of the business rule. If the business rule is triggered by a timer rather than changes in attribute values, then you can use timer parameters. There are two timer parameters:

- `run_at` parameter. This parameter is used to create a timer that runs the business rule at a specified time by using the following format:

  "run_at=*hour*,*interval*,*first_run*"

  where:

  hour is the hour of the day at which the business rule runs

  interval is the number of days between runs of the business rule. If this entry is omitted, the business rule runs every day.

  `first_run` is the number of days before the first run of the business rule. If this entry is omitted, the business rule runs on the next configured hour.

  For example:

  "run_at=1,7,3"means run the business rule at 1 am every week and the first run is scheduled 3 days from now

  "run_at=1,7" means run the business rule at 1 am every week

  “run_at=1" means run the business rule at 1 am every day

Note You can specify only one `run_at` parameter.

- `update_interval` parameter. This parameter is used to set the interval between two runs of the business rule by using the following format:

  "update_interval=*hours*"

  where *hours* is the number of hours between runs of the business rule.

  For example, "update_interval=4" means run this business rule every 4 hours.

  Note You can specify only one `update_interval` parameter.

Notes

1 You cannot use both "update_interval=" and "run_at=" at the same time.

2. You can specify only a particular matrix cell or a range of cells as a listener attribute and not an entire matrix attribute.

3. An incoming link that is configured with a view should not be specified as a listener or `listen_to` parameter. The behavior of a business rule that listens to an incoming link that is configured with views cannot be guaranteed and needs special handling.

Triggering of Business Rules

A business rule is triggered in the following situations:

- a formula is saved

  The most common scenario is when an attribute is edited and a formula is entered. This can also happen when a new element is created or a new attribute with a default formula is created.

- one of the listener attributes changes
- one of the `listen_to` attributes changes
- a timer is triggered through the use of a `run_at` or `update_interval` parameter

Storing the Business Rule Text

Business rules are stored inside a container attribute. Results from the business rules are usually used to update the same attribute. One example of such a business rule is the `List` business rule, which is stored inside an integer or float attribute and is used to do aggregate functions on the linked elements of a linklist attribute. Here the same attribute that hosts the business rule stores the result of the aggregate function.

But, in some situations, the business rule might update another attribute. Only the status of the last run is stored in the container attribute. One example of such a business rule is the `SetChoice` business rule. This business rule is stored inside a text attribute, but updates the status of a choice attribute by listening to an integer attribute. The status of only the last run of this business rule is stored in the container attribute.

Business Rule Design Considerations

Consider the following points when designing a new business rule:

- Fully understand the business case to be solved.
- Determine if the business rule will update the container attribute or another attribute.
- Determine the positional arguments to send to the business rule. Usually these are the names of the attributes that the business rule should read data from or write data to. You can also use these arguments to send flags to the business rule.

- Determine the listener attributes for the business rule. These are the attributes on the same element that can trigger the business rule.
- Determine the attributes on the linked elements that can trigger the business rule. You can send them to the business rule by using the `listen_to` parameter.
- Determine if the business rule is triggered by a timer and if so whether to use the `run_at` parameter or the `update_interval` parameter.
- Determine the value the business rule should return. It could be the value of the business logic or the status of the run.
- Determine whether to control the execution of the business rule. For example, you could use a checkbox attribute as a flag. The name of the checkbox attribute can be sent to the business rule as a positional argument. A business rule should stop execution if the checkbox is turned off.

  Note A control attribute should not be sent as a listener attribute, but as a positional argument only.

Business Rule Development

Business rules are written in Java, based on the business rule API that is provided in Focal Point. The API along with the Javadoc can be downloaded from the business rule administration page in Focal Point. You must add the API jar file to your class path. Note the following points before proceeding:

- All new business rules are implementations of an abstract class `BusinessRule` that is provided in the API. A business rule must be a direct descendant of this class; you cannot inherit another business rule.
- New implementing classes should have the following class annotations in the class file:

```
@VersionInfo(APIVERSION = Version.APIVERSION,
  APIFPVERSION = Version.APIFPVERSION,
  REQUIREDFPVERSION = Version.REQUIREDFPVERSION)
```

- There should be no package declarations in the implementing class.
- Business rule Java files must be compiled by using Java version 8.0.
- Implementing classes should provide implementations for the following abstract methods:
  - `perform()` - This is where the real business logic is executed through a call back. The method should return a Java string object that represents either the value of the evaluation or the status of the run (success or failure).
  - `getHelp()` - Provides descriptive message for a new person to set up this business rule.
  - `getCopyrightMessage()` - Provides the copyright statement for the business rule.
  - `getAuthor()` - Provides the name or id of the business rule developer.

- The following methods provide access to the runtime environment of the business rule and data model of the Focal Point instance.
  - `getRuntimeContext()` - Returns the `RuntimeContext` object, which provides the details on the runtime context of the business rule. This includes details like the container element, container attribute, positional and listener arguments passed to the business rule, change attribute, type of change event, and the user who triggered the business rule.
  - `getGlobalContext()` - Returns the `GlobalContext` object, which provides a method to access workspaces, modules, and elements in Focal Point by using ids, aliases and names. It also provides the ability to store the attribute changes in the database or create a new element in any module.

Example 1- Copy Attribute from Link Target

Note Custom business rules that you create based on this example are not supported. Product support is limited to the business rule API and the business rules that are included in the standard installation.

Business Case

There might be situations when you want to copy a specific attribute from a linked element (source attribute) to an attribute in the current element (target attribute). Focal Point provides an in-built attribute called `Mirror` for this purpose. The `Mirror` attribute listens to a link attribute and copies a specific attribute from the link target into the mirror attribute. `Mirror` attributes have some limitations; they do not support copying all kinds of attributes. For example, you cannot use a `Mirror` attribute to copy another `Mirror` attribute; the type of the `Mirror` attribute is different from the type of the attribute being copied, and `Mirror` attributes cannot be used in expressions or filtering. This example shows how to create a business rule that overcomes these limitations of `Mirror` attributes.

Structure

Use the following format for the business rule. In this example, it is assumed that the names of the source attribute and the target attribute are the same.

```
LinkTargetAttributeCopy("Link_AttributeName","AttributeName",
"CheckboxName",'Link_AttributeName',
"listen_to=AttributeName")
```

where:

`Link_AttributeName` is the name of link attribute

`AttributeName` is the name of the attribute to be copied,

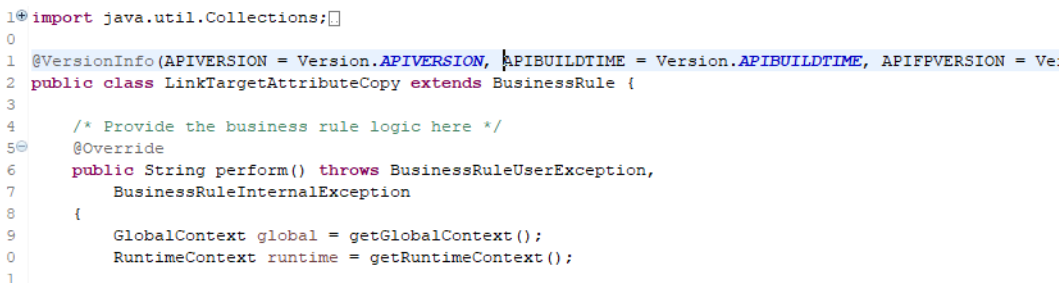
`CheckboxName` is the name of the checkbox attribute for turning off the business rule execution.

The parameter `Link_AttributeName` is provided so that the business rule listens to this attribute for changes.

The parameter `listen_to=AttributeName` is provided so that any change in the source attribute on the linked element triggers the business rule.

Class Definition

Create a `LinkTargetAttributeCopy` class that extends the abstract class `BusinessRule` (see the class annotations that are used). Then override the perform method and obtain the `GlobalContext` and `RuntimeContext` objects as shown in the following image:

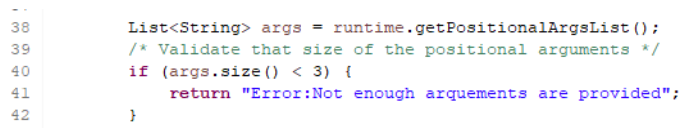


```
import java.util.Collections;

@VersionInfo(APIVERSION = Version.APIVERSION, APIBUILDTIME = Version.APIBUILDTIME, APIFPVERSION = Ve
public class LinkTargetAttributeCopy extends BusinessRule {

    /* Provide the business rule logic here */
    @Override
    public String perform() throws BusinessRuleUserException,
        BusinessRuleInternalException
    {
        GlobalContext global = getGlobalContext();
        RuntimeContext runtime = getRuntimeContext();
```

Validation of Inputs

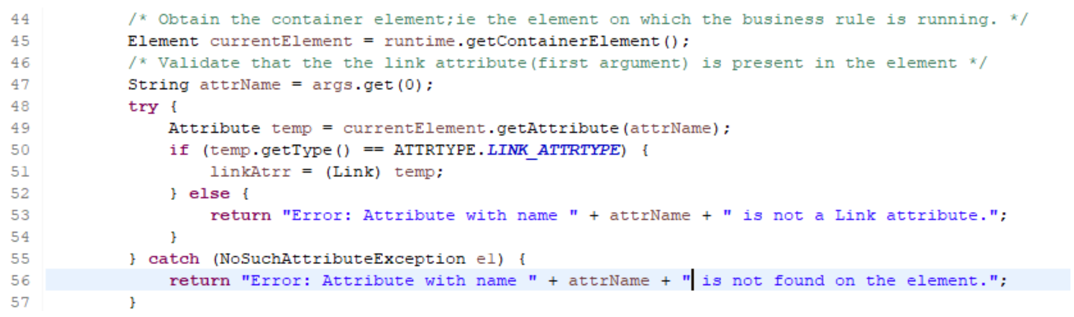
Validate the inputs given to the business rule.

1 Make sure that enough positional arguments are passed into the business rule:



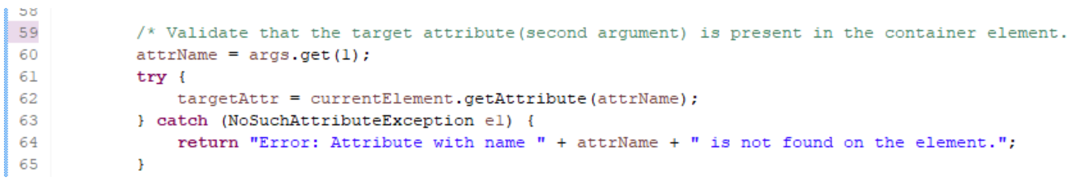
```
        List<String> args = runtime.getPositionalArgsList();
        /* Validate that size of the positional arguments */
        if (args.size() < 3) {
            return "Error:Not enough arguements are provided";
        }
```

2 Ensure that the first argument passed is a link attribute that is in the business rule container element:



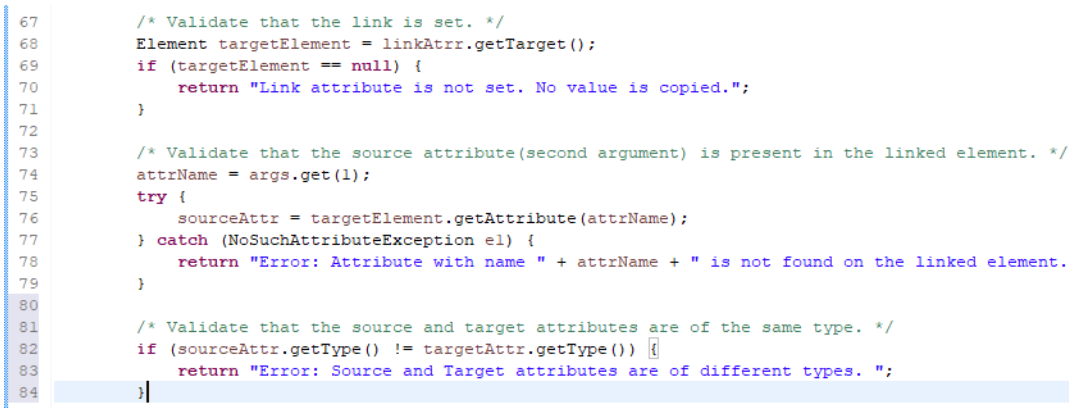
```
        /* Obtain the container element;ie the element on which the business rule is running. */
        Element currentElement = runtime.getContainerElement();
        /* Validate that the the link attribute(first argument) is present in the element */
        String attrName = args.get(0);
        try {
            Attribute temp = currentElement.getAttribute(attrName);
            if (temp.getType() == ATTRTYPE.LINK_ATTRTYPE) {
                linkAtrr = (Link) temp;
            } else {
                return "Error: Attribute with name " + attrName + " is not a Link attribute.";
            }
        } catch (NoSuchAttributeException e1) {
            return "Error: Attribute with name " + attrName + " is not found on the element.";
        }
```

3 Validate the second argument and make sure that it identifies an attribute in the container element.



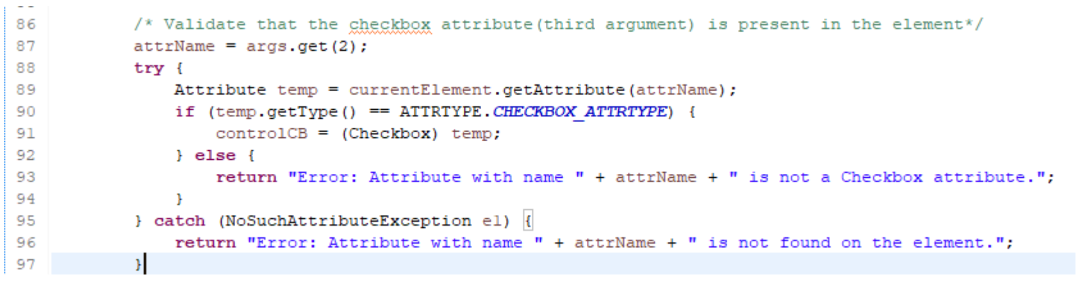
```java
        /* Validate that the target attribute(second argument) is present in the container element.
        attrName = args.get(1);
        try {
            targetAttr = currentElement.getAttribute(attrName);
        } catch (NoSuchAttributeException e1) {
            return "Error: Attribute with name " + attrName + " is not found on the element.";
        }
```

4 Ensure that the link attribute is set. If it is not set, then stop the execution there. Otherwise, ensure that the attribute to be copied is available on the linked element and both the source attribute and target attribute are of the same type:



```java
        /* Validate that the link is set. */
        Element targetElement = linkAtrr.getTarget();
        if (targetElement == null) {
            return "Link attribute is not set. No value is copied.";
        }

        /* Validate that the source attribute(second argument) is present in the linked element. */
        attrName = args.get(1);
        try {
            sourceAttr = targetElement.getAttribute(attrName);
        } catch (NoSuchAttributeException e1) {
            return "Error: Attribute with name " + attrName + " is not found on the linked element.
        }

        /* Validate that the source and target attributes are of the same type. */
        if (sourceAttr.getType() != targetAttr.getType()) {
            return "Error: Source and Target attributes are of different types. ";
        }
```

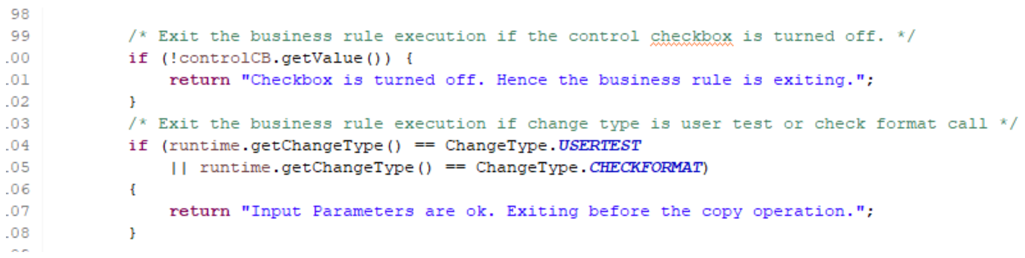
5 Verify the checkbox parameter:



```java
        /* Validate that the checkbox attribute(third argument) is present in the element*/
        attrName = args.get(2);
        try {
            Attribute temp = currentElement.getAttribute(attrName);
            if (temp.getType() == ATTRTYPE.CHECKBOX_ATTRTYPE) {
                controlCB = (Checkbox) temp;
            } else {
                return "Error: Attribute with name " + attrName + " is not a Checkbox attribute.";
            }
        } catch (NoSuchAttributeException e1) {
            return "Error: Attribute with name " + attrName + " is not found on the element.";
        }
```

Business Rule Exit Strategy

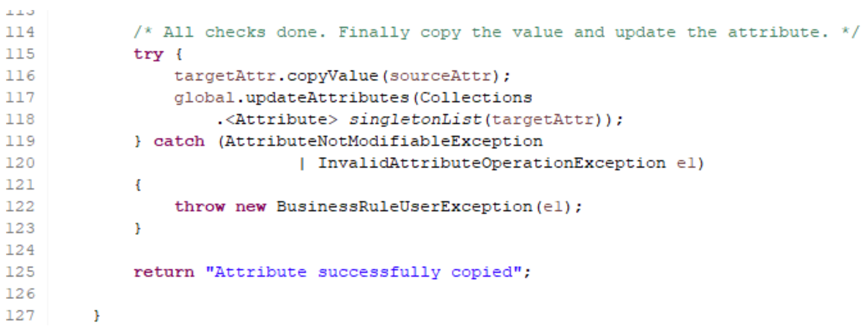
In certain situations, you might want to stop the execution of a business rule. A common scenario is to exit the business rule if the control checkbox attribute is turned off. Another scenario is, when a business rule is evaluated (in the user interface) by a user, the user might not want the attribute to be copied; but simply want to verify the inputs.



```java
        /* Exit the business rule execution if the control checkbox is turned off. */
        if (!controlCB.getValue()) {
            return "Checkbox is turned off. Hence the business rule is exiting.";
        }
        /* Exit the business rule execution if change type is user test or check format call */
        if (runtime.getChangeType() == ChangeType.USERTEST
            || runtime.getChangeType() == ChangeType.CHECKFORMAT)
        {
            return "Input Parameters are ok. Exiting before the copy operation.";
        }
```

Final Execution

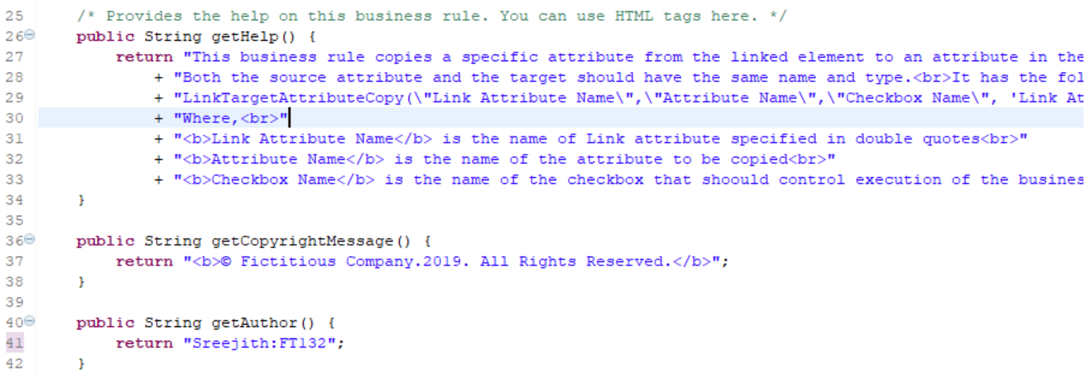
In this case, the business rule copies the value of the source attribute into the target attribute and updates the value in the database, as shown in the following image:



```
        /* All checks done. Finally copy the value and update the attribute. */
        try {
            targetAttr.copyValue(sourceAttr);
            global.updateAttributes(Collections
                .<Attribute> singletonList(targetAttr));
        } catch (AttributeNotModifiableException
                    | InvalidAttributeOperationException e1)
        {
            throw new BusinessRuleUserException(e1);
        }

        return "Attribute successfully copied";

    }
```

Help, Copyright Statement and Developer Information

The methods below can be used to provide information to configure the business rule and also identify the author of the business rule.



```
    /* Provides the help on this business rule. You can use HTML tags here. */
    public String getHelp() {
        return "This business rule copies a specific attribute from the linked element to an attribute in the
            + "Both the source attribute and the target should have the same name and type.<br>It has the fol
            + "LinkTargetAttributeCopy(\"Link Attribute Name\",\"Attribute Name\",\"Checkbox Name\", 'Link At
            + "Where,<br>"
            + "<b>Link Attribute Name</b> is the name of Link attribute specified in double quotes<br>"
            + "<b>Attribute Name</b> is the name of the attribute to be copied<br>"
            + "<b>Checkbox Name</b> is the name of the checkbox that shoould control execution of the busines
    }

    public String getCopyrightMessage() {
        return "<b>© Fictitious Company.2019. All Rights Reserved.</b>";
    }

    public String getAuthor() {
        return "Sreejith:FT132";
    }
```

Example 2- Copy Attribute from Parent Folder

Note Custom business rules that you create based on this example are not supported. Product support is limited to the business rule API and the business rules that are included in the standard installation.

Business Case

Another common requirement is to update the attribute of an element based on the attribute value of its parent folder element. For example, you might want to automatically close the children when the group they belong to is closed.

This scenario cannot be modelled in the same way as the `LinkTargetAttributeCopy` business rule that is described in the previous example. In that case, the value was copied from the linked element (whenever the attribute on the linked element changed) by specifying a `listen_to` parameter. A `listen_to` parameter only works with link or linklist attributes and not with a parent folder attribute. However, there are other ways to achieve the required functionality.

Split the requirement into two parts

One method for accomplishing this is to split the requirement into two parts.

1 Develop a business rule to automatically link every element to its parent folder element. This business rule should specify the parent folder attribute as a listener attribute (enclosing the attribute name in single quotes) so that whenever the elements are moved across the folders, the link attribute is properly updated.

2 Use the `LinkTargetAttributeCopy` business rule on this link attribute to pull the value of the attribute from the parent folder to the child element.

Push and pull attribute values

Another way of accomplishing this is through a combination of push and pull of attribute values.

1 Design a new business rule that has both the attribute to be copied and parent folder attribute as listener attributes. This can be achieved by specifying both these attribute names enclosed in single quotes.

2 When the business rule is triggered on elements (when their parent folder changes), the rule pulls the value of the attribute from the new parent folder element.

3 When the business rule gets triggered on folders (when their specific attribute changes), the rule pushes the new value of the attribute to all the children of the folder.

Business Rule Best Practices

- Make sure the name of the business rule is unique, short and concise.
- Provide a detailed explanation of the functionality and the working of the business rule as the help text.

  Also include details on the arguments passed to the business rule. The details that you provide should be sufficient for a new person to set up and maintain the business rule.
- Document all the exit scenarios and boundary cases for the business rule. Test cases should cover these scenarios.
- Fully test newly-created business rules on a test environment before moving them to production.
- Design business rules with performance considerations in mind.

Business rules are executed as part of the expression queue evaluation where normal expressions are evaluated. If the business rules are time-consuming or they are triggered very often, then it is likely to choke the expression queue evaluation. Due to this reason, normal and simple expressions might take longer to evaluate. Ideally, you should benchmark performance in your test environment.

- Set time-consuming business rules to run at non-peak hours.

  This might sacrifice the real-time nature of the rule, but the effect (on normal business operations) of evaluating large expressions can be minimized.

- Consider providing a provision to turn off the execution of the business rule.

  This can be implemented using a checkbox attribute as a flag. A business rule should read the value of the checkbox attribute and exit the execution if the checkbox attribute is off. Send the name of the attribute to the business rule as a positional argument.