Developing Custom Actions in SAP MII 14.0

Document Version 1.2 - November 11, 2012

This document was created using stylesheet 2006-06-15 (V4.3) and XSLT processor SAXON 6.5.2 from Michael Kay (http://saxon.sf.net/), XSLT version 1.

Disclaimer

Some components of this product are based on Java™. Any code change in these components may cause unpredictable and severe malfunctions and is therefore expressively prohibited, as is any decompilation of these components.
Any Java™ Source Code delivered with this product is only to be used by SAP's Support Services and may not be modified or altered in any way.

Documentation in the SAP Service Marketplace

You can find this documentation at the following Internet address:
`service.sap.com/instguides`

Table of Contents

1 Overview

Many users of SAP Manufacturing Integration and Intelligence (SAP MII) need special operations that are not provided by SAP MII. These operations may include special calculation mechanisms or specific communication functions. The purpose of this document is to summarize how a Java developer can create custom actions for such operations.

2 Background

Custom actions have been available to SAP MII users since its inception through custom interfaces and helper classes. Due to the major architectural changes to the core SAP MII Business Logic Services (BLS) Engine, these older interfaces have been deprecated and could cease to function in a future release. No end date has been determined at the release of this document. Custom actions developed prior to SAP MII 12.1 will work in 12.1; however, they will run slower than 12.1 custom actions because a legacy action layer translates the calls to the 12.1 methodology.

There have been many changes to the 12.1 custom action APIs, which make actions easier to develop and maintain. These features require the Java 1.5 JVM, especially the generic and annotation constructs.

3 Annotations

3.1 Action

The Action annotation identifies a method as an action.

Parameter	Type	Description
name†	`String`	Name of the action
icon	`String`	Location of the icon in the class path
retention	`ActionRetention`	Specifies if the action runs in a particular execution state or not. Defaults to ALWAYS.
forceOutputLinks	`boolean`	If set to True, it forces the action to expose and run output links. Default setting is False.
referenceDocumentGenerator	`Class implementing IReferenceDocumentGenerator`	Declares the class needed to generate a reference document. Used in design time only. Defaults to a generator that produces no reference documents.

†Required parameter

3.2 Outputs

The Outputs annotation is used to describe the known output of an action at compile time. You can also create dynamic outputs (see the IActionConfiguration interface). If there are defined outputs, this annotation is required..

Parameter	Type	Description
names†	`String[]`	Holds the array of output names
types†	`VariantDataTypes[]`	Holds the array of types corresponding to the names

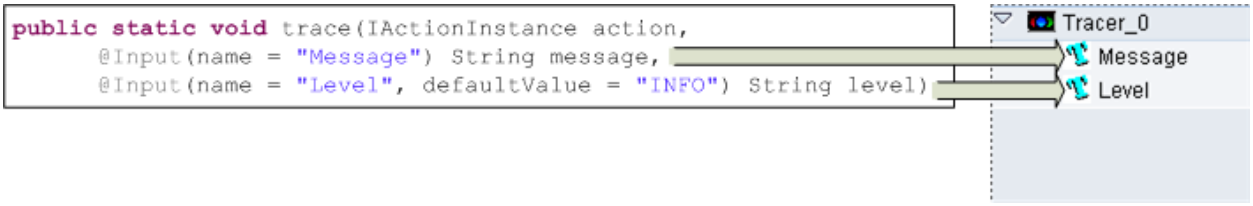
†Required parameter

NOTE: The types and names arrays must be the same size, and the order of the names corresponds to the order of the types.

3.3 Input

The Input annotations are put on the parameters for the action method and describe the inputs to the action. All parameters, except the `IActionInstance`, must have this annotation attached or the loading of the action fails.

The allowable types for action parameters include those described in the `VariantDataTypes` enumeration, the `VaraintData`, and one `IActionInstance` parameter. If you use the `VariantData` as a parameter, you should specify the annotations type parameter.



Parameter	Type	Description
name†	`String`	Holds the name of the parameter as the user sees it. This name does not need to be identical to the methods parameter name.
defaultValue	`String`	Holds the parameter default value as a string. If the parameter type is not a string, the BLS engine automatically cases this value. If this annotation parameter is not specified, and if the value is not overridden, the system uses the default value for the

		type.
type	`VariantDataTypes`	Holds the array of types corresponding to the names

†Required parameter

3.4 ConfigurationDialog

You use the ConfigurationDialog annotation to create a configuration dialog so the user can set action parameters. If you annotate the action method with this annotation and do not define a dialogClass, you will create a default configuration dialog in which all inputs are configurable.

Parameter	Type	Description
dialogClass	`Class implementing IConfigurationDialog`	References the class to use as the dialog class

3.5 DefinesAccompanimentData

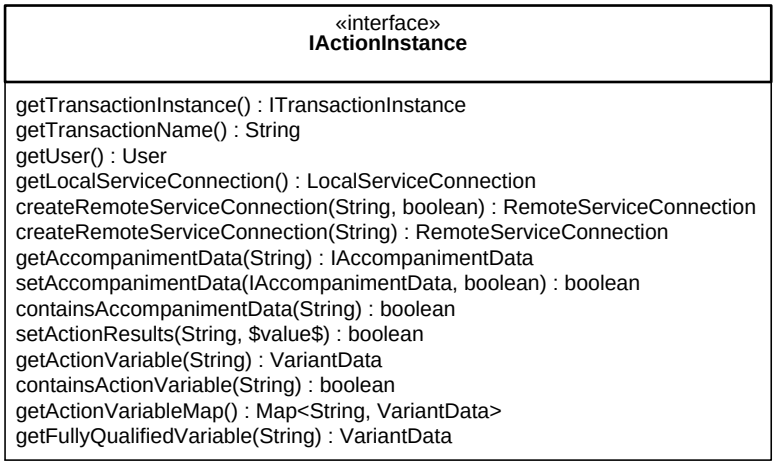
This annotation tells the engine that an `IAccompanimentData` object will be created by this action. The engine makes it easier for downstream actions to reference the action. The action can create any name for the `IAccompanimentData` interface; however, it is standard to call the object by the object instance name, which is acquired through `IActionInstance.getName()`.

Parameter	Type	Description
Guids	`String[]`	The list of GUIDs used to identify the `IAccomanimentData` class. For constraint information, see `IAccompanimentData`.

4 Interfaces

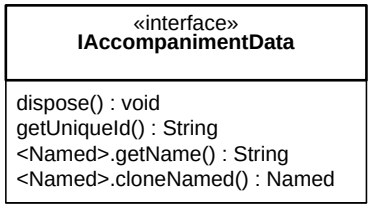
4.1 IActionInstance

This is the main runtime state model for the action instance. It contains methods for retrieving and updating state information. The action developer does not implement this interface, the engine implements it.



4.2 IAccompanimentData

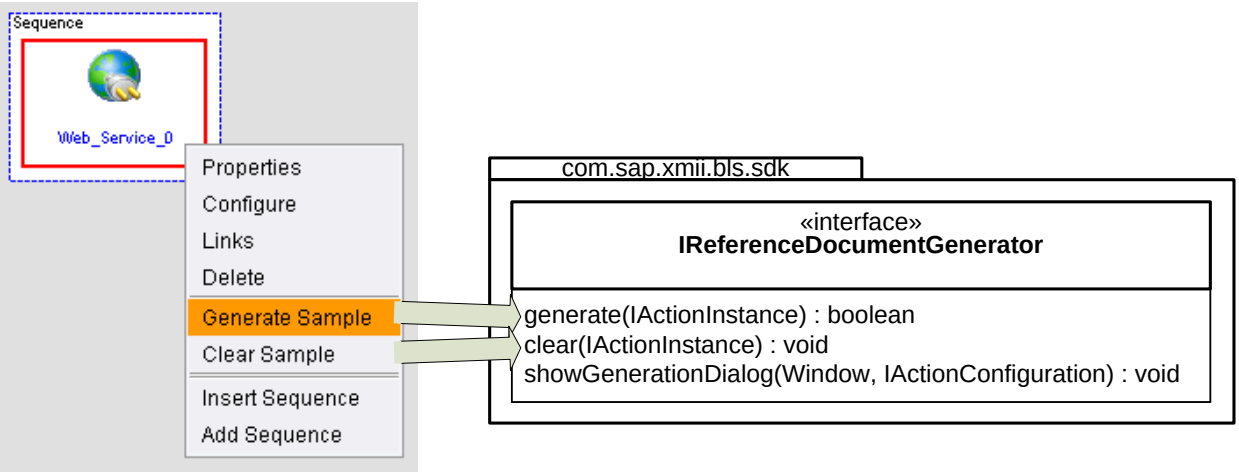
If a complex stateful object must be shared by multiple actions, or multiple action calls, the action developer should implement this interface.



4.3 IReferenceDocumentGenerator

This interface generates the reference document that helps the user develop links. The returned document is typically an XML document that requires external lookup.

You can access this functionality by clicking the object with your secondary mouse button and choosing *Generate Sample* or *Clear Sample*.

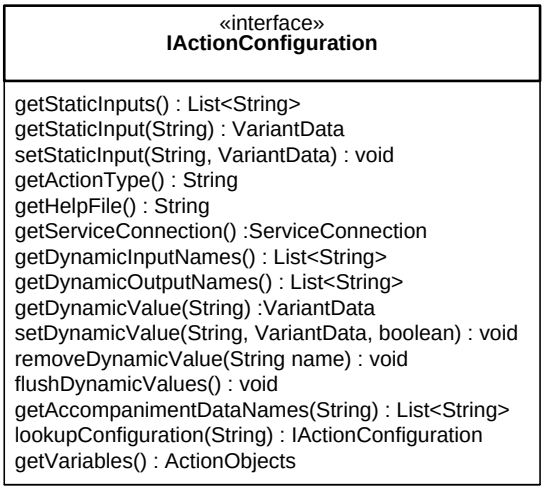


4.3.1 Definition

Method	Description
generate	Called when the user calls the *Generate Sample* operation. The return from this method indicates if the operation was successful (`true`) or not (`false`).
clear	Called when the user calls the *Clear Sample* operation
showGenerationDialog	Typically called after the configuration dialog is closed. This allows for better interaction than requiring the user to close the dialog and select *Generate Sample*.

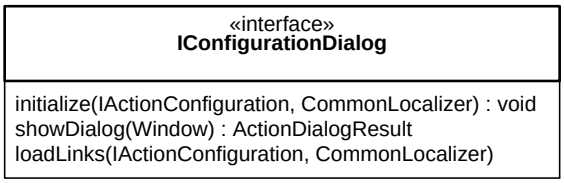
4.4 IActionConfiguration

This interface is used as a configuration mechanism for the actions. It is only available at design time and assists the `IConfigurationDialog`.



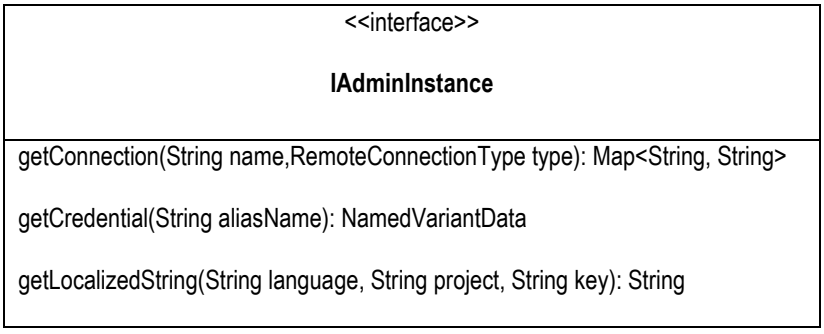
4.5 IConfigurationDialog

If you use the `BaseConfigurationDialog` as a base class, you do not need to implement this interface.



4.6 IAdminInstance

You get the localized text for a key by providing inputs for the language, project and key to the method getLocalizedString() of the IAdminInstance interface.



5 Class

5.1 BaseConfigurationDialog

This class uses a basic configuration dialog to implement the `IConfigurationDialog` interface. You can use this method as a base for any dialog you need to create.

6 Enumerations

6.1 ActionRetention

This enumeration determines if an action should be put into the execution tree.

Value	Description
DEBUG_ONLY	Run the action in debug environments only
ALWAYS	Run the action in both debug and production environments

7 Deployment

To deploy custom actions you need to zip all class files into a JAR and add a catalog.xml file to the root folder of the JAR. Once the JAR is complete, you can add the JAR to SAP MII at *System Management* → *Custom Actions*.

The following is an example of a catalog.xml file:

```
<?xml version = "1.0" encoding="utf-8" ?>
<ComponentCatalog>
      <Category Name="BlackBerry" Description="BlackBerry">
            <Component Type="Action"
            Name="AddTwoNumbers"
            Description=""
            Label="CustomActionSample"
            ClassName="customactions.SimpleActions"
            AssemblyName="CustomAction.jar"
            HelpFileName="" />
      </Category>
</ComponentCatalog>
```

Note: You can add multiple actions to the catalog.xml by adding more component nodes.

Examples

7.1 Simple Action

```
package customactions;

import com.sap.lhcommon.common.*; // Needed for the VariantDataTypes and
                                          //VariantData types.
import com.sap.xmii.bls.sdk.*; // This is the main SDK for custom actions.

/**
* This is a simple action construct used to show how to use some of the basic
* MII 12.1 Custom Action constructs.
*
*/
public class SimpleActions {

        /**
        * To ensure the parameters are always named the same, it is good practice to
        * create a static final string with the parameters name. This also make it
        * easy to use across actions.
        */
        private static final String PARAM_OUTPUT = "Output";
        /**
        * Describes a simple action that adds two numbers together.
        * @param instance This is the action instance. This is the main interface
        * from the action code back into the transaction engine. Most
        * users will only need to set their variables through this
        * interface, though much more powerful operations are also
        * available.
        * @param in1 This is an input into the action. The type is detected by the
        * transaction engine and automatically cast to the correct value.
        * @param in2 This is a second input.
        * @throws InvalidVariableException This exception is
        */
        @Action(name = "AddTwoNumbers" ,
                        referenceDocumentGenerator = ReferenceDocumentGenSample.class) // This annotation
tells the engine that this
        // is an action available to execute.
        @Outputs(names = { PARAM_OUTPUT },
        types = { VariantDataTypes.INTEGER }) // This annotation tells
        // the engine that one
        // integer output called
        // 'Output' is going to be
        // returned.
        @ConfigurationDialog(dialogClass = SimpleConfigurationDialog.class, localizationType =
ConfigurationDialogLabelLocalizationType.BUILT_IN)
        // This annotation tells the engine the class that implements IConfigurationDialog.
        public static void addTwoNumbers(
        IActionInstance instance, // Besides the basic types, the
        // IActionInstance interface is the only
        // other type allowed to be defined in
        // parameter list of a custom action.
        @Input(name = "Input1") int in1, // @Input annotations are used to
        // indicate the user modifiable
        // inputs to this action.
        @Input(name = "Input2") int in2)
        throws InvalidVariableException {// Exceptions can be thrown directly from
        // the actions without causing critical
        // execution failures. These exceptions
        // will be caught and logged by the engine
```

```
        // and will cause the Success flag to be
        // set to false.
        // The following code describes how to set an actions output.
                try {
                        instance.setActionResult(PARAM_OUTPUT, in1 + in2);
                } catch (Exception e ) {
                        // Do what you like.
                }
         }
}
```

7.2 Reference Document Generator

```
package customactions;


import java.awt.*;
import javax.swing.*;
import org.w3c.dom.*;
import com.sap.lhcommon.common.*;
import com.sap.lhcommon.exceptions.*;
import com.sap.lhcommon.xml.*;
import com.sap.xmii.bls.sdk.*;
/**
* This is a sample for generating reference documents. It simply adds the
* following XML:
*
* <? xml version="1.0" encoding="UTF-8 ?>
* <Sample>This is some data to store.</Sample>
*
*/
public class ReferenceDocumentGenSample implements IReferenceDocumentGenerator {
/**
* Clear the configuration of this sample document.
*/
public void clear(IActionConfiguration configuration) {
try {
configuration.setStaticInput("Output",
new VariantData(new XMLDataType()));
} catch(DataConversionException e) {
configuration.setStaticInput("Output", new VariantData());
}
}
/**
* Generate the sample documents.
*/
public boolean generate(IActionConfiguration configuration) {
boolean succceeded = false;
try {
        Document doc = XMLHandler.createDocument();
        Element elem = doc.createElement("Sample");
        doc.appendChild(elem);
        elem.appendChild(doc.createTextNode(
                        "This is some data to store."));
        configuration.setStaticInput("Output",
                                new VariantData(new XMLDataType(doc)));
        succceeded = true;
```

```
} catch(DataConversionException e) {
        configuration.setStaticInput("Output", new VariantData());
}
        return succceeded;
}
/**
* Show a dialog asking if the reference document should be generated.
*/
public void showGenerationDialog(Window parentWindow,
IActionConfiguration configuration) {
if (JOptionPane.showConfirmDialog(parentWindow,
"Do you want to generate a sample document?") ==
JOptionPane.YES_OPTION) {
generate(configuration);
}
}

/**
 * Generate the reference documents upon initialization from the configuration.
 */
public boolean initialize(IActionConfiguration configuration)
{
        return true;

}

}
```

7.3 Accompaniment Data

```
package customactions;

import com.sap.lhcommon.common.*;
import com.sap.xmii.bls.sdk.*;

/**
 * A simple accompaniment data sample.
 */
public class SimpleAccompanimentData implements IAccompanimentData {

	/**
	 * Defines the serialization version.
	 */
	private static final long serialVersionUID = 1L;

	/**
	 * Defines the StatisticsMemory id.
	 */
	public static final String UID = "054BC6DC-07AE-4cc0-A818-D5881A494268";

	/**
	 * The object to store in this accompaniment data object.
	 */
	private Object storedObject;

	/**
	 * The name of this object.
	 */
	private final String name;

	/**
	 * Creates this object with a member to store.
	 * @param storedObject The object to store
	 */
	public SimpleAccompanimentData(Object storedObject, String name) {
		this.storedObject = storedObject;
		this.name = name;
	}

	/**
	 * dispose of the object.
	 */
	public void dispose() {
		this.storedObject = null;
	}

	/**
	 * Get the unique id for this object type.
	 */
	public String getUniqueId() {
		return UID;
	}

	/**
	 * clone this object.
	 */
	public Named cloneNamed() {
		return new SimpleAccompanimentData(this.storedObject, this.name);
	}

	/**
	 * Get the objects name.
	 */
	public String getName() {
		return this.name;
	}
}
```

7.4 Configuration Dialog

```
package customactions;


import java.awt.*;
import javax.swing.*;

import com.sap.lhcommon.common.*;
import com.sap.lhcommon.exceptions.*;
import com.sap.xmii.bls.sdk.*;

/**
* A simple dialog.
*/
public class SimpleConfigurationDialog extends BaseConfigurationDialog {

private static final long serialVersionUID = 1L;

private JTextField input1;
private JTextField input2;
/**
* Creates the Tracer Action dialog.
*/
public SimpleConfigurationDialog() {
        super(475, 250);
        input1 = new JTextField(10);
        input2 = new JTextField(10);
}
/**
* @see com.sap.xmii.bls.sdk.BaseConfigurationDialog#load()
*/
@Override
protected void load() {
        JPanel mainPanel = new JPanel(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();

    c.gridy = 0;
    c.weightx = 0.1;
    c.fill = GridBagConstraints.NONE;
    mainPanel.add(new JLabel("Input1"), c);
    c.weightx = 0.9;
    c.fill = GridBagConstraints.HORIZONTAL;
    mainPanel.add(input1, c);

    c.gridy = c.gridy + 1;
    c.weightx = 0.1;
    c.fill = GridBagConstraints.NONE;
    mainPanel.add(new JLabel("Input2"), c);
    c.weightx = 0.9;
    c.fill = GridBagConstraints.HORIZONTAL;
    mainPanel.add(input2, c);

        getContentPane().setLayout(new GridBagLayout());
        GridBagConstraints e = new GridBagConstraints();
    e.gridy = 0;
    e.weightx = 1.0;
    e.weighty = 1.0;
    e.fill = GridBagConstraints.BOTH;
        getContentPane().add(mainPanel, e);

    e.gridy = e.gridy + 1;
    e.weightx = 1.0;
    e.fill = GridBagConstraints.NONE;
    e.anchor = GridBagConstraints.LINE_END;
    createOKCancelPanel(this, e);

    String option1 = "256";
```

```
    String option2 = "256" ;
try {
        option1 = this.getConfiguration().getStaticInput("Input1").stringValue();
        option2 = this.getConfiguration().getStaticInput("Input2").stringValue();
} catch(DataConversionException x) {}
        input1.setText(option1);
        input2.setText(option2);
}
/**
* @see com.sap.xmii.bls.sdk.BaseConfigurationDialog#saveData()
*/
@Override
protected void saveData() {
        try {
        this.getConfiguration().setStaticInput("Input1", new
                        VariantData(Integer.parseInt(input1.getText())));
        this.getConfiguration().setStaticInput("Input2", new
                        VariantData(Integer.parseInt(input2.getText())));
        } catch (Exception e) {};
}

        }
```