

Digitize business processes successfully

Accelerate digital transformation with the X4 BPMS low code platform

X4 ADK

The information in this document is subject to change without notice. SoftProject GmbH assumes no responsibility for any errors that may appear in this document.

This document may not be copied, photocopied, reproduced, translated or converted to any electronic or machine-readable form in whole or in part without prior written approval of SoftProject GmbH.

Mentioned products are trademarks or registered trademarks of their respective owners.

Contact

SoftProject GmbH

Am Erlengraben 3

D-76275 Ettlingen – Germany

Website: www.softproject.de

Sales

Phone: +49 7243 56175-0

vertrieb@softproject.de

SoftProject Support

Phone: +49 7243 56175-333

support@softproject.de

© SoftProject GmbH. All rights reserved.


Table of Contents

1	Introduction	5
1.1	What is the X4 Adapter Development Kit (X4 ADK)?.....	5
1.2	How can I use the X4 Adapter Development Kit?	5
1.3	What prior knowledge is useful?	5
2	Developing Adapters with the X4 BPMS	6
2.1	The Structure of Adapter Projects.....	6
2.2	Creating Adapter Projects	6
2.3	Defining the Adapter Interface	7
2.4	Providing the Adapter Project as Module	10
2.5	Providing Adapters on the X4 Server	10
2.6	Debugging processes containing native adapters.....	11
3	Developing Own Adapters in Java	13
3.1	Adapter Basics	13
3.2	Configuring the ADK Environment	13
3.3	Implementing the Adapter Interface.....	13
3.3.1	Interface and Constructor	13
3.3.2	Defining Adapter Metadata.....	14
3.3.3	Using BaseAdapter.....	14
3.3.4	Using the Adapter Interface.....	14
3.3.5	Example	16
3.4	Creating the Configuration Bean	17
3.4.1	Purpose of a Configuration Bean	17
3.4.2	Creating a Configuration Bean.....	17
3.4.3	Defining Metadata for Adapter Parameters.....	17
3.4.4	Example	18
3.5	Specifying Operations	18
3.5.1	Operations for Adapters	19
3.5.2	Assigning Methods to Operations	19
3.5.3	Defining Metadata for Adapter Operations.....	19

- 3.6 Providing Adapters on the X4 Server 19
- 3.7 Defining the Adapter Icon20
 - 3.7.1 Default Icons20
 - 3.7.2 Providing the Icon20
- 4 Adapter Framework Reference..... 21
 - 4.1 Adapter Interface 21
 - 4.1.1 Adapter Interface Methods 21
 - 4.2 X4Document Interfaces..... 21
 - 4.2.1 X4Document Methods22
 - 4.2.2 X4DocumentFactory Methods23
 - 4.2.3 Example 23
 - 4.3 ProjectExplorer API for the Project Access.....24
 - 4.4 Developing Adapters24
 - 4.4.1 Status Methods25
 - 4.5 Exceptions and Error Handling25
 - 4.5.1 AdapterException Methods26
 - 4.5.2 MissingParameterException Methods26
 - 4.5.3 AdapterConfigurationException Methods26

1 Introduction

1.1 What is the X4 Adapter Development Kit (X4 ADK)?

 To use the adapter development kit of the X4 BPMS, an ADK license is required.

With the X4 ADK, you can develop your own adapters within adapter projects in order to integrate third-party systems into processes. Implementing your own adapters is an easy way of integrating various applications into the X4 BPMS and extending its range of functions.

1.2 How can I use the X4 Adapter Development Kit?

For this purpose, X4 BPMS provides a framework that simplifies adapter development in many ways. This documentation provides an overview of the adapter development framework (ADK) and explains the procedure for developing your own adapters.

Features of the X4 ADK are:


- Arbitrary extensibility of the X4 ESB
- Simple creation and integration of own adapters
- Use of already existing functionalities in technical processes

1.3 What prior knowledge is useful?

This documentation is intended for Java developers who want to develop project or system-specific adapters for the X4 BPMS. Java knowledge in the JEE application development, basic knowledge about XML, XSLT and XPath and especially detailed technical knowledge of the data and the workflows are important requirements for this purpose.


2 Developing Adapters with the X4 BPMS

Adapter projects allow to develop new adapters for the X4 BPMS using the various tools provided within the X4 BPMS.


 To create and use custom adapters, an ADK license is required.

2.1 The Structure of Adapter Projects

Adapter projects have a predefined and unchangeable folder structure, which is created automatically when a new Adapter project is created.

 The automatically created folders and subfolders cannot be deleted, moved or renamed.

Adapters	Adapters, which are used within technical processes, are stored here.
Icons	The different adapter icons are stored here.
Operations	Any number of technical processes (.wrf) can be created here. Each process provides an adapter operation thereby.
Resources	Folders, resources such as images, text and XML documents and other files can be stored here.
TemporaryFiles	Temporarily used files are stored here.
Transformations	Transformations (.xsl) and reports (.rep) are stored here.
<Project>.nad	Adapter definition, which is created and named after the project automatically when creating a new adapter project, e.g. <i>AdapterProject_Documentation.nad</i> .

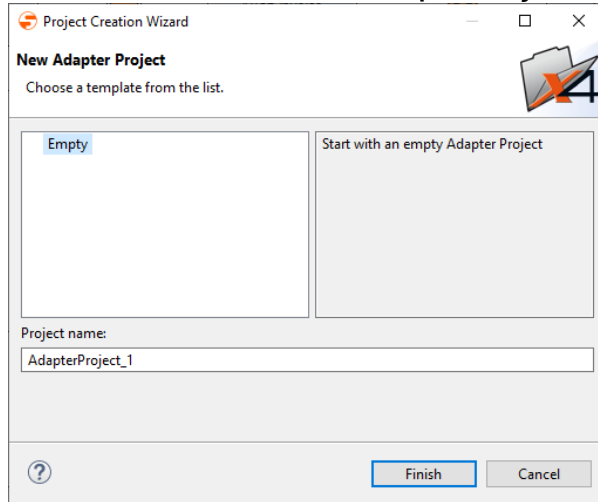
 A project can contain only one adapter definition.

2.2 Creating Adapter Projects

There are different options to create adapter projects:

- Via **New Adapter** on the X4 Designer Welcome page
- Via the context menu **New > Adapter Project** within the Repository Navigator
- Via the menu **File > New > Adapter Project**

1. Select the menu **File > New > Adapter Project** within the X4 Designer.



2. Select an empty project template or an already existing template.
3. Enter the project's name in **Project name**.
4. Click **Finish** to create the project.

An empty adapter project with the predefined structure is created within the Repository Navigator.

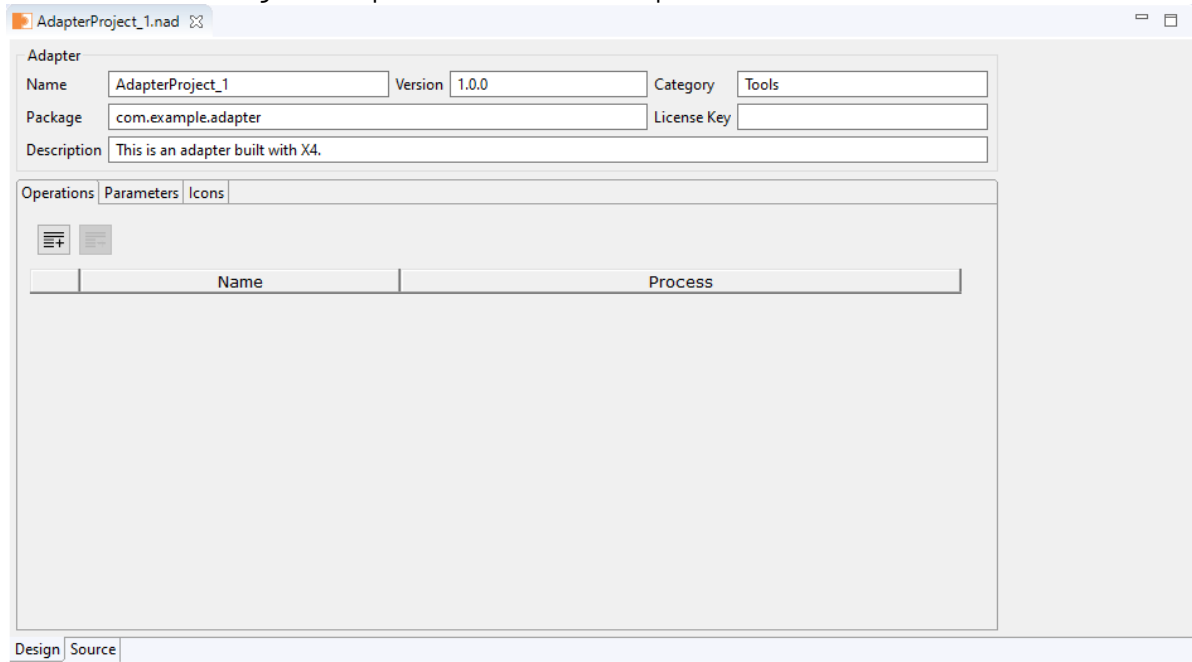
i The various elements of an ESB project can now be created via the context menu. The available options change depending on where in the tree structure the context menu is opened.

2.3 Defining the Adapter Interface


The adapter interface can be defined within the adapter definition file (.nad). The available adapter operations are specified using technical processes.


1. Create an adapter project, see [Creating Adapter Projects](#).



- Open the adapter definition file <Projectname> .nad by double-clicking it. An editor for defining the adapter interface will be opened.



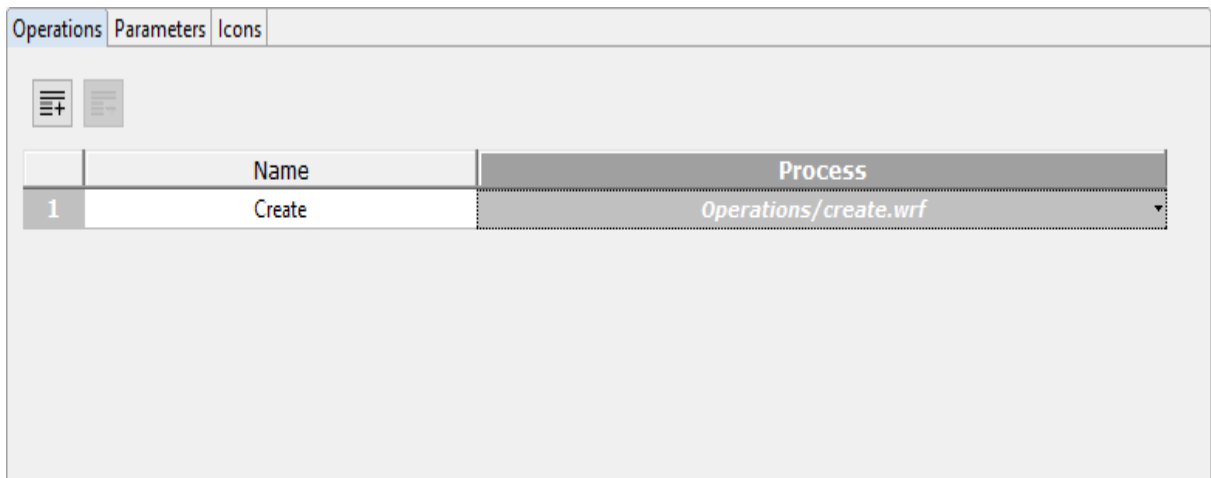
- Define the general adapter settings within the **Adapter** area. The following settings can be made:
 - Name:** Adapter name
 - Version:** Adapter version
 - Category:** Adapter category, e.g. Tools
 - Package:** Name of the WildFly module to be provided.


 This setting is relevant when using the option **Package as Adapter**.

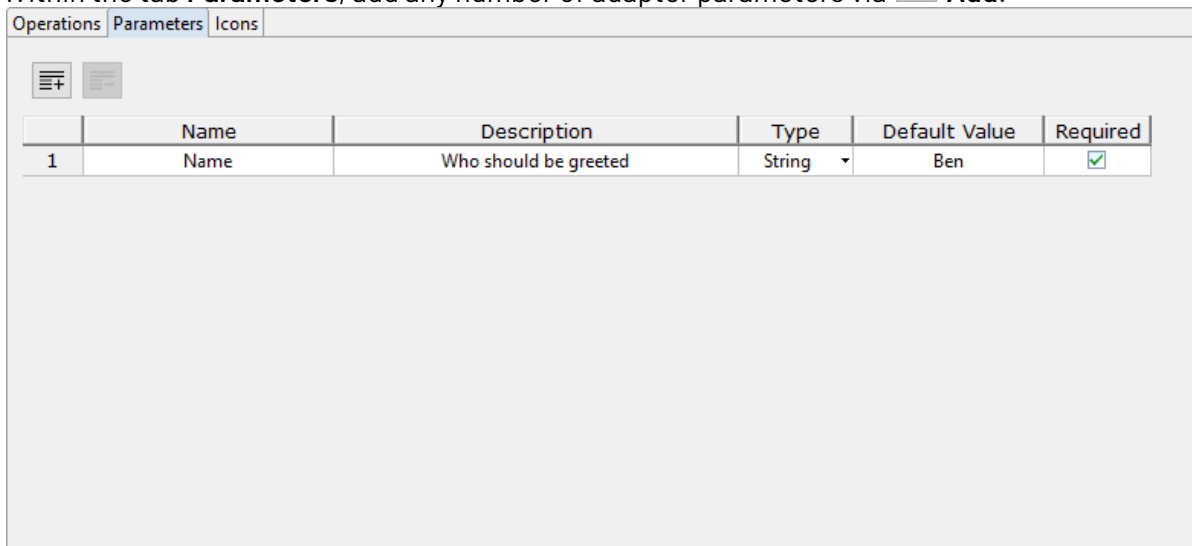
- License Key:** Adapter license key
 - Description:** Adapter description
- Within the tab **Operations**, add any number of adapter operations via  **Add**. Each operation has to be linked to a technical process.

-  To delete an operation select the row and click  **Remove**.
- Newly created processes are updated automatically.

- Select for each operation the corresponding technical process from the folder **Operations**




6. Within the tab **Parameters**, add any number of adapter parameters via  **Add**.




7. Define the adapter parameters. The following settings can be made:


- **Name:** Parameter name
- **Description:** Parameter description
- **Type:** Parameter type
 - **String:** For strings
 - **Number:** For numbers
 - **Boolean:** For Boolean values
 - **Combo:** For drop-down lists

 Names and values for possible options of a combo box can be created in the **Combo Options**. The **Combo Options** are displayed automatically, when type **Combo** is selected for a parameter.

- **Password:** For passwords

 Values with type **password** are not in plain text and cannot be copied.

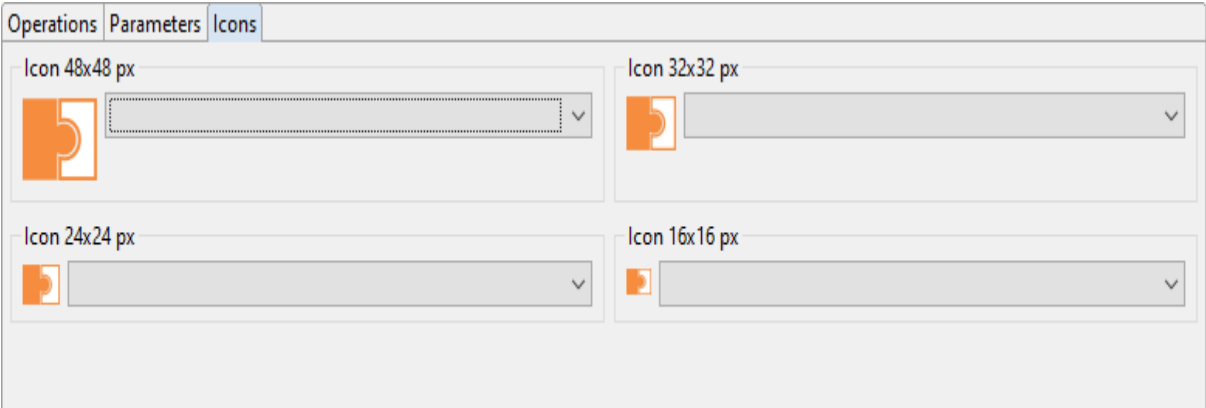
- **Default Value:** Default value of the parameter
- **Required:** Information, whether a parameter is mandatory or optional


 The parameters will be available within technical processes as placeholder PARAM_INP UT_<Name>.

8. Within the tab **Icons**, define images from the folder

Icons

as adapter icons.



 Newly added icons are updated automatically.

9. Save the adapter definition.

2.4 Providing the Adapter Project as Module

Adapters created with the X4 BPMS can be packaged like Java adapters and provided as module in WildFly. The function `Package as a Wildfly Module` is available via the context menu of the adapter project.


1. Make the adapter definition as required, see [Defining the Adapter Interface](#).
2. Open the context menu on the top project level (project name) and select **Package as Adapter**. A dialog for exporting the project is now opened.
3. Select the desired storage location and enter the adapter name.
4. Click **Save** to export the adapter project as a ZIP folder.
5. Provide the adapter on the Server, see [Providing Adapters on the X4 Server](#).

2.5 Providing Adapters on the X4 Server

To include the finished adapter package into the X4 Server configuration, the compiled adapter classes and required external libraries are provided as Java archive (.jar file) on the application server.

The sample project within the ADK package already contains the corresponding Maven build, which generates the .jar file and descriptors already with the expected folder structure.

1. Unpack the generated .zip file under C:
 \X4\Server\<wildfly>\modules\system\layers\base.
2. Add a reference to the adapter's own module within the X4-Extensions module
 (de.softproject.x4.extensions).
3. Restart the X4 Server.
 The adapter will be available within the X4 Designer after the next connection with the X4 Server and can be then used in processes.

 Note that the adapter project and the provided adapter module cannot be used simultaneously on the same X4 Server.


2.6 Debugging processes containing native adapters

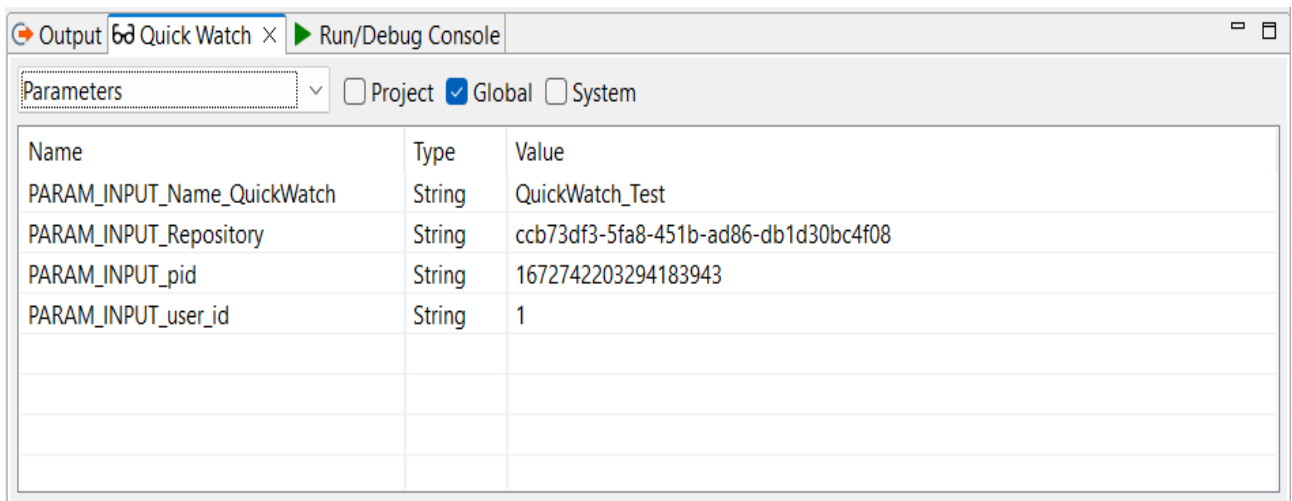
When debugging processes that contain one or more native adapters, you proceed in the same way as when debugging processes with preconfigured adapters. For more information, see the section Debugging and Running Processes.

When you debug a process that contains a native adapter, the **Quick Watch** view also displays the input parameters of the native adapter. You can recognize the input parameters by the PARAM_INPUT prefix before the parameter name.

How to display the parameters of native adapters in the Quick Watch view:

- Select **Parameters** from the drop-down list to display all parameters.

 If you want to limit the displayed parameters further, select the corresponding checkbox. For example, select **Global** to display only global parameters. The input parameters of native adapters belong to the category of global parameters.



The screenshot shows the 'Quick Watch' window in the X4 ADK software. The window title bar includes 'Output', 'Quick Watch', and 'Run/Debug Console'. Below the title bar, there is a dropdown menu set to 'Parameters' and three radio buttons: 'Project' (unchecked), 'Global' (checked), and 'System' (unchecked). The main area of the window contains a table with the following data:

Name	Type	Value
PARAM_INPUT_Name_QuickWatch	String	QuickWatch_Test
PARAM_INPUT_Repository	String	ccb73df3-5fa8-451b-ad86-db1d30bc4f08
PARAM_INPUT_pid	String	1672742203294183943
PARAM_INPUT_user_id	String	1

✓ **Additional information:**

- Section Views in the chapter with the description of the X4 Designer Interface
- Section Creating parameters in the chapter Creating and modelling processes

3 Developing Own Adapters in Java

How to develop your own adapter using the adapter framework of X4 ADK.

3.1 Adapter Basics


X4 adapters are process components providing basic functions such as data conversions or document processing or allowing the access to complex enterprise systems. Since the X4 ESB mainly works with XML data, adapters within technical processes usually create or modify XML documents. An adapter usually requires an input and outputs an XML document, which is further processed in the next process step.

Implementing custom adapters is the fastest and easiest way to integrate various applications into the X4 BPMS and to extend its functionality. The X4 BPMS provides a framework that simplifies the adapter development in many ways.

3.2 Configuring the ADK Environment

Learn about the development environment is required and how to configure it.

In order to develop an adapter for the X4 BPMS you first need to provide the necessary classes and configure the Java development environment (e.g. *Eclipse*) with a Java Runtime Environment (JRE) in version 1.8 or later.

 You can request the ADK package with the needed X4 Java classes and examples from SoftProject support via support@softproject.de.

Unpack the `adk-example.zip` and import it as already existing Maven project into your Java IDE.

The Java project now contains the folder `src/main/java` (code examples).

3.3 Implementing the Adapter Interface

How to create the basic framework for the adapter implementation using the standardized adapter interface

3.3.1 Interface and Constructor

All adapters use the interface `de.softproject.integration.adapter.core.Adapter` and the default constructor (without parameters) to create the main class. The abstract class `de.softproject.integration.adapter.core.BaseAdapter` contains a base implementation to get the adapter status and to access the project directory.

3.3.2 Defining Adapter Metadata

To define adapter metadata the annotation `@AdapterDescription` is used. The following fields must be set:

Field	Description	Mandatory
<code>category</code>	Technical adapter category	yes
<code>description</code>	Description of the adapter's purpose	yes
<code>displayName</code>	Displayed adapter name	yes
<code>version</code>	Adapter version with the format <code>major.minor.patch</code> (e.g. <code>1.0.0</code>)	yes
<code>configurationBeanClass</code>	Type of the <code>ConfigurationBean</code> , if the adapter requires parameters. If the adapter class contains the configuration, no values have to be set here.	no

3.3.3 Using BaseAdapter

This base class contains protected methods for accessing the adapter status and the `ProjectExplorer`. The basic implementation assumes that the configuration is made within the adapter class, which means that the adapter parameters are defined in the class itself. If you want to use your own class for the adapter configuration, the method `getConfigurationBean` must be overwritten as described in the section *Defining Adapter Metadata*.

3.3.4 Using the Adapter Interface

1. Create a new adapter class that implements the interface `de.softproject.integration.adapter.core.Adapter`.
2. Use the default constructor. This X4 class will be instantiated via default constructor at runtime of the X4 ESB.

The methods `init()`, `cleanup()` and `getConfigurationBean()` are created and the required packages are imported. With this constructor, the following elements are created:

- `public void cleanup()`: This method *always* cleans up after the execution of the `operation` method and closes resources that have been allocated by the adapter instance.
- `public void init(Status arg0, long arg1)`: After creating the adapter instance, a status instance will be created and will be handed over to the adapter with the `init()` method call. Parameter `arg0` specifies an object's adapter instance to set a status; parameter `arg1` specifies the adapter instance (within the code).

When executed, the adapter can manipulate its status instance within its `do...` methods, and this state will be evaluated by the X4 ESB.

- `public Object getConfigurationBean()`: Returns an instance of the configuration bean containing setter methods for setting configuration parameters. Adapter instances can access configuration parameters via the configuration bean object.
3. Create a status variable and assign an instance status.
 4. Create a configuration bean object (see [Creating the Configuration Bean](#)), and use `getConfigurationBean()` as return value.

The basic framework for the adapter main class is now ready.

3.3.5 Example

```

import java.nio.charset.StandardCharsets;
import de.softproject.integration.adapter.annotations.AdapterDescription;
import de.softproject.integration.adapter.annotations.AdapterOperation;
import de.softproject.integration.adapter.annotations.AdapterParameter;
import de.softproject.integration.adapter.core.AdapterException;
import de.softproject.integration.adapter.core.AdapterParameterDataType;
import de.softproject.integration.adapter.core.BaseAdapter;
import de.softproject.integration.util.x4documents.MimeTypeUtils;
import de.softproject.integration.util.x4documents.X4Document;
import de.softproject.integration.util.x4documents.X4DocumentFactory;

/**
 * This is a simple Adapter showing how Adapters are declared in the
 * X4-BPMS.<br>
 * An adapter needs some Annotation to be recognized by the X4-System:<br>
 * <ul>
 * <li>{@link AdapterDescription}: Mandatory for making an Adapter visible. It
 * describes the metadata of the adapter</li>
 * <li>{@link AdapterParameter}: Mandatory for Parameters. They will be mapped
 * to the corresponding setter Method. It must match the parameter name.
 * <li>{@link AdapterOperation}: Mandatory for Operations. It describes the
 * metadata for an adapter operation
 * </ul>
 */
@AdapterDescription(category = "Demo", description = "Say Hello...", displayName =
"HelloWorld", version = "1.0.0")
public class HelloWorldAdapter extends BaseAdapter {

    @AdapterParameter(dataType = AdapterParameterDataType.STRING, description = "say
hello to", optional = true)
    private String name = "default";

    @AdapterOperation(name = "HelloWorld")
    public X4Document sayHelloWorld(X4Document input) {
        return X4DocumentFactory.createX4Document("Hello World",
StandardCharsets.UTF_8.name(),
        MimeTypeUtils.TEXTPLAIN);
    }

    @AdapterOperation(name = "HelloTo")
    public X4Document sayHelloTo(X4Document input) throws AdapterException {
        if (name == null || "".equals(name)) {
            throw new AdapterException(123, "Parameter \"name\" ist nicht
gesetzt...");
        }
        getStatus().setOk();
        return X4DocumentFactory.createX4Document("Hello " + name,
StandardCharsets.UTF_8.name(),
        MimeTypeUtils.TEXTPLAIN);
    }
}

```



```

public void setName(String name) {
    this.name = name;
}
}

```

3.4 Creating the Configuration Bean

How to create the basic framework for a configuration bean

3.4.1 Purpose of a Configuration Bean

A configuration bean is used to bundle configuration parameters of an adapter instance and contains methods for setting the configuration parameters. Adapter instances can access the configuration bean object using the method `getConfigurationBean()`.

3.4.2 Creating a Configuration Bean

1. Create a new configuration bean class within the package.
2. Define variables which serve as configuration parameters (only of the type `String`).
3. The annotation `@AdapterParameter` specifies a field as adapter configuration. There can either be a Configuration Bean class in which all parameters are defined, or the definition can be made in the adapter class itself (a mixture is not possible!).
4. Define set/get methods for each configuration parameter. They have to be named following the pattern `set<Parameter name>(<value>)` or `get<Parameter name>()`.
The basic frame of the configuration bean is now ready.

3.4.3 Defining Metadata for Adapter Parameters

To define metadata for adapter parameters the annotation `@AdapterParameter` is used:

Field	Description	Mandatory
<code>description</code>	Textual description of the parameter's purpose	yes
<code>dataType</code>	Parameter type <i>Possible values:</i> PASSWORD, BOOLEAN, STRING, ARRAY, DOUBLE, INTEGER	yes
<code>displayName</code>	The parameter's display name. The field's name is default.	no
<code>optional</code>	Specifies whether the parameter is mandatory or not <i>Possible values:</i> <ul style="list-style-type: none"> • <code>true</code>: the parameter is not mandatory • <code>false</code>: the parameter is mandatory (default) 	no

Field	Description	Mandatory
defaultValue	Sets an initial parameter value	no

i The annotation `@AllowedPropertyValue` has to be used additionally, if `dataType` is set to `ARRAY` in order to set the possible values.

Field	Description
displayName	Specifies the value's display name
value	Technical identifier that is written to the field during selection

3.4.4 Example

```
import de.softproject.integration.adapter.annotations.AdapterParameter;
import de.softproject.integration.adapter.core.AdapterParameterDataType;

public class SimpleChartConfiguration {

    @AdapterParameter(dataType = AdapterParameterDataType.STRING, description = "set
the title of the chart")
    private String chartTitle;
    @AdapterParameter(dataType = AdapterParameterDataType.INTEGER, description = "set
the width of the chart", defaultValue = "300")
    private int chartWidth = 300;

    public String getChartTitle() {
        return chartTitle;
    }

    public void setChartTitle(String chartTitle) {
        this.chartTitle = chartTitle;
    }

    public int getChartWidth() {
        return chartWidth;
    }


    public void setChartWidth(int chartWidth) {
        this.chartWidth = chartWidth;
    }
}
```

3.5 Specifying Operations

How to define one or more operations for an adapter


3.5.1 Operations for Adapters

Adapters offer certain functions that can be selected within the *X4 Designer* using the property `Operation` in the Properties view. Each list entry of this selection menu corresponds to an operation method within the adapter.

 An adapter needs at least one operation. If the adapter offers only a single operation, it will be automatically selected in the Process Designer.

3.5.2 Assigning Methods to Operations

1. Specify for each operation the annotation `@AdapterOperation` for declaring to a public method in the adapter class.
2. Define the parameters.

-  Each method can define a parameter, which corresponds to the adapter input.
- The method's return type can be `X4Document` or a [JAXB-conform](#) Java bean. If the return type is `void`, the input is considered as output.
 - To use Java beans, the input has to be XML and conform to the Java bean definition. Otherwise, the adapter execution will be aborted with the status `-1`.

The basic frame of an Operation method is now ready.

3.5.3 Defining Metadata for Adapter Operations

To define metadata for adapter operations the annotation `@AdapterOperation` is used:

Field	Description
name	Specifies the operation's name and can be freely selected. However, it has to be unique for each adapter.

3.6 Providing Adapters on the X4 Server

To include the finished adapter package into the X4 Server configuration the compiled adapter classes and required external libraries are provided as Java archive (`.jar` file) on the application server.

The sample project within the ADK package already contains the corresponding Maven build, which generates the JAR file and descriptors already with the expected folder structure.

1. Execute *Maven clean install*.
2. Unpack the generated ZIP file `target/adk-examples-1.0.0-SNAPSHOT-wildfly-module.zip` under `C:\X4\Server\<wildfly>\modules\system\layers\base`.




3. Add a reference to the adapter's own module within the *X4-Extensions* module.
4. Restart the X4 Server.
The adapter will be available within the X4 Designer after the next connection with the X4 Server and can be then used in processes.

3.7 Defining the Adapter Icon

You can define custom icons for already existing and self-developed adapters. They are loaded from the X4 Repository and displayed within the process diagram instead of the standard adapter icon. You can either define icons for single adapters or for the different adapter types.

3.7.1 Default Icons

For default icons, annotations are available that can be annotated to the adapter class.

Adapter type	Symbol	Annotation
Converter		@ConverterIcon
Transfer		@TransferAdapterIcon
Connector		@ConnectorIcon



If no annotation is specified, the function adapter icon  is used.

3.7.2 Providing the Icon

The adapter symbol is provided using the annotation @AdapterIcon, which can be written to the adapter class.

Icon sizes of 16x16, 24x24, 32x32 and 48x48 can be defined. At least one size can be specified, and all other sizes are then calculated from the largest image size. The specified value must be a classpath to the image resource in `src/main/resources`.

Example: The directory `src/main/resources/HelloWorld/icons` contains the image `icon_16.png` which is to be used as adapter icon. The corresponding path is `/HelloWorld/icons/icon_16.png`.

4 Adapter Framework Reference

Which components of the *X4 ADK* are available for adapter developers

4.1 Adapter Interface

How the adapter interface, the constructor and the methods are used

All adapters use the interface `de.softproject.integration.adapter.core.Adapter` and the default constructor (without parameters) to create the main class. Each adapter instance has a `status` object in order to set a status after the adapter component's execution, and a unique identifier.

4.1.1 Adapter Interface Methods

de.softproject.integration.adapter.core.Adapter	
<code>public void cleanup()</code>	Always cleans up after the execution of the <code>operation</code> method and closes resources having been occupied by the adapter instance.
<code>public void init(Status arg0, long arg1)</code>	Will be called after creating the adapter instance. The parameter <code>arg0</code> names the object for whose adapter instance a status is to be set, the parameter <code>arg1</code> specifies the adapter instance (within the code).
<code>public Object getConfigurationBean()</code>	Returns an instance of the configuration bean containing setter methods for setting configuration parameters. Adapter instances can access configuration parameters via the configuration bean object.

4.2 X4Document Interfaces

Which methods are available to access input documents, process documents and create documents within adapters

Process components, i. e. instances of an adapter for example, can read and output many document types: XML documents, as well as text or binary data. With the `X4Document` interface, adapters can access documents or generate with `X4DocumentFactory` an output document as result of the adapter operation. This can be an `X4Document` instance of a `dom4jtree`, a `java.io.InputStream` or a byte array.

i Character encoding problems can be avoided, if XSL transformations get a DOM-based input within X4 processes. For byte-based documents the byte array is passed to the XML parser without additional information.

4.2.1 X4Document Methods

de.softproject.integration.util.x4documents.X4Document	
<i>public byte[] getAsByteArray()</i>	Returns the content as byte array
<i>public byte[] getAsByteArray(String encoding)</i>	Returns the content as byte array with the encoding specified within encoding
<i>public java.io.InputStream getAsInputStream()</i>	Returns the InputStream of the access to the document content
<i>public org.dom4j.Document getAsDocument() throws org.dom4j.DocumentException</i>	Returns the content as <i>dom4j</i> DOM tree
<i>public org.dom4j.Document getAsDocument(String encoding) throws org.dom4j.DocumentException, java.io.UnsupportedEncodingException</i>	Returns the content as <i>dom4j</i> DOM tree with the encoding specified within encoding
<i>public void parseWithHandler(ContentHandler chandler) throws SAXException, IOException</i>	Passes the content to the provided <i>org.sax.ContentHandler</i> instance (flush)
<i>public String getMimeType()</i>	Returns the MIME type of the document
<i>public String getEncoding()</i>	Returns the encoding of the document
<i>public void setMimeType(String mimeType)</i>	Sets the MIME type of the document
<i>public void setEncoding(String encoding)</i>	Sets the encoding of the document

4.2.2 X4DocumentFactory Methods

de.softproject.integration.util.x4documents.X4DocumentFactory	
<i>public static X4Document createX4Document(byte[] bytes, String encoding, String mimeType)</i>	Outputs the X4Document instance with the byte array content bytes, which has the encoding encoding and the MIME type mimeType
<i>public static X4Document createDocument(Document doc, String encoding, String mimeType)</i>	Outputs the X4Document instance with the dom4J tree content doc, which has the encoding encoding and the MIME type mimeType
<i>public static X4Document createX4Document(InputStream is, String encoding, String mimeType) throws X4DocumentException</i>	Outputs the X4Document instance with the java.io.InputStream content is, which has the encoding encoding and the MIME type mimeType

4.2.3 Example

```

public class X4DocumentTest {
    public static void main(String[] args) {
        InputStream is = null;
        try {
            is = new FileInputStream("Sample.xml");
            // create a new X4Document
            X4Document x4doc = X4DocumentFactory.createX4Document(is,
                "cp-1252", "text/xml");
            // get content as a dom4j document
            System.out.println(x4doc.getAsDocument().asXML() + "\n");
            // get content as a byte array
            System.out.println(new String(x4doc.getAsByteArray())
                + "\n");
            // parse the x4document's content
            SAXContentHandler handler = new SAXContentHandler();
            x4doc.parseWithHandler(handler);
            System.out.println(handler.getDocument().asXML() + "\n");
        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            if (is != null)
                try {
                    is.close();
                } catch (IOException ioex) {
                }
        }
    }
}

```

4.3 ProjectExplorer API for the Project Access

The ProjectExplorer API allows to access project files using the adapter.

Method	Description
String getName()	Returns the project name
LocalDateTime getLastModifiedTime(ProjectPath path)	Returns the file's last modification date
InputStream read(ProjectPath path)	Returns the files content as input stream
List<ProjectPath> getProjectContent()	Returns all files as project paths
void write(ProjectPath path, InputStream inputStream)	Writes the data stream to the specified position and creates a new file, if the project type's rules allow it
boolean exists(ProjectPath path)	Checks whether the file or folder exists
void create(ProjectPath path)	Creates a directory
void delete(ProjectPath path)	Deletes a directory or a file
void move(ProjectPath sourcePath, ProjectPath targetPath)	Moves a file or directory to the specified location according to the project type's rules
void copy(ProjectPath sourcePath, ProjectPath targetPath)	Copies a file or directory to the specified location according to the project type's rules
List<ProjectPath> getFolderContent(ProjectPath folderPath)	Returns all paths within the folder and all subfolders

4.4 Developing Adapters

For all adapters status values can be set via the class `de.softproject.integration.adapter.core.Status`. These status values can be interpreted within X4 processes using `Condition` process components. Each executed operation method of an adapters can set a corresponding status.

Thereby, the three default status values `OK (1)`, `ERROR (-1)` and `TIMEOUT_EXCEEDED (0)`, or a custom integer status value can be set. If no status is set in the adapter, occurring exceptions while executing the operation method will be treated as `ERROR (-1)`, otherwise the status `OK (1)` will be returned.

4.4.1 Status Methods

de.softproject.integration.adapter.core.Status	
<i>public void setOk()</i>	Sets the status of the operation method to <i>executed successfully</i>
<i>public void setError()</i>	Sets the status of the operation method to <i>executed incorrectly</i>
<i>public void setTimeoutExceed()</i>	Sets the status of the operation method to <i>not executed within the defined time</i>
<i>public void setStatus(int Status)</i>	Sets the status code Not allowed values are: <i>-999, -2 and 999!</i>
<i>public boolean isOk()</i>	Checks if the status is set to <i>executed successfully (1)</i>
<i>public boolean isError()</i>	Checks if the status is set to <i>executed incorrectly (-1)</i>
<i>public boolean isTimeoutExceed()</i>	Checks if the status is set to <i>not executed within the defined time</i>
<i>public int getStatus()</i>	Reads the status value as integer number
<i>public static final int OK=1</i>	Associates the status OK with the value <i>1</i>
<i>public static final int ERROR=-1</i>	Associates the status ERROR with the value <i>-1</i>
<i>public static final int TIMEOUT_EXCEED=0</i>	Associates the status TIMEOUT_EXCEED with the value <i>0</i>

4.5 Exceptions and Error Handling

An easy possibility to handle errors within adapters, is throwing exceptions. The class `de.softproject.integration.adapter.core.AdapterException` allows an error code and a message to be assigned to the root exception.

If the adapter misses parameter values, which can happen during the process modeling, they can throw a `de.softproject.integration.adapter.core.MissingParameterException`.

Configuration problems should be signalized by adapters by throwing an `de.softproject.integration.adapter.core.AdapterConfigurationException`.

4.5.1 AdapterException Methods

de.softproject.integration.adapter.core.AdapterException	
<i>public AdapterException(int reasonCode, String msg)</i>	Constructor for an adapter exception with reasonCode (numeric code to identify fault cases) and msg (to describe the fault case)
<i>public AdapterException(String msg, Exception ex)</i>	Constructor for an adapter exception with msg (to describe the fault case) and ex (root exception)
<i>public AdapterException(int reasonCode, String msg, Exception ex)</i>	Constructor for an adapter exception with reasonCode (numeric code to identify fault cases), msg (to describe the fault case), and ex (root exception)

4.5.2 MissingParameterException Methods

de.softproject.integration.adapter.core.MissingParameterException	
<i>public MissingParameterException(int reasonCode, String msg)</i>	Constructor for a MissingParameter exception with reasonCode (numeric code to identify fault cases) and msg (to describe the fault case)

4.5.3 AdapterConfigurationException Methods

de.softproject.integration.adapter.core.AdapterConfigurationException	
<i>public AdapterConfigurationException(int reasonCode, String msg)</i>	Constructor for an AdapterConfigurationException with reasonCode (numeric code to identify fault cases) and msg (to describe the fault case)