Geschäftsprozesse erfolgreich digitalisieren

Die digitale Transformation durch die Low-Code-Plattform X4 BPMS beschleunigen

X4 ADK

SoftProject

Die in dieser Dokumentation enthaltenen Informationen und die zugehörigen Programme können ohne besondere Ankündigung geändert werden. Für etwaige Fehler übernimmt SoftProject keine Haftung.

Diese Dokumentation und die zugehörigen Programme dürfen ohne schriftliche Zustimmung der SoftProject GmbH weder ganz noch teilweise kopiert, reproduziert, verändert oder in irgendeine elektronische oder maschinenlesbare Form umgewandelt werden.

Alle genannten Warenzeichen sind Warenzeichen der jeweiligen Eigentümer.

Kontakt

SoftProject GmbH

Am Erlengraben 3

D-76275 Ettlingen

Website: www.softproject.de

Vertrieb

Telefon: +49 7243 56175-0

vertrieb@softproject.de

SoftProject-Support

Telefon: +49 7243 56175-333

support@softproject.de

Inhaltsverzeichnis

1 Einführung

1.1 Was ist das X4 Adapter Development Kit (X4 ADK)?

> ⓘ Um das X4 Adapter Development Kit der X4 BPMS zu nutzen, wird eine ADK-Lizenz vorausgesetzt.

Mit dem X4 ADK können innerhalb von Adapter-Projekten eigene Adapter entwickelt werden, um Drittsysteme in Prozesse zu integrieren. Eigene Adapter zu implementieren ist eine schnelle und einfache Möglichkeit, um verschiedene Anwendungen in die X4 BPMS zu integrieren und den Funktionsumfang der X4 BPMS zu erweitern.

1.2 Wie kann das X4 Adapter Development Kit genutzt werden?

Mit dem X4 ADK stellt die X4 BPMS stellt ein Framework bereit, das die Adapter-Entwicklung in vielerlei Hinsicht vereinfacht. Diese Dokumentation gibt einen Überblick über das Framework zur Adapter-Entwicklung (ADK) und erläutert die Vorgehensweise zur Entwicklung eigener Adapter.

Merkmale des X4 ADK sind:

- Beliebige Erweiterbarkeit des X4 ESB
- Einfache Erstellung und Einbindung eigener Adapter
- Nutzung bereits bestehender Funktionalitäten in technischen Prozessen

1.3 Welches Wissen ist erforderlich?

Diese Dokumentation richtet sich an Java-Entwickler, die für die X4 BPMS projekt- oder systemspezifische Adapter entwickeln möchten. Dazu sind neben guten Java-Kenntnissen in der Java EE-Anwendungsentwicklung und grundsätzlichem Wissen zu XML, XSLT und XPath vor allem auch detailliertes fachliches Wissen der Daten und Arbeitsabläufe erforderlich.

2 Eigene Adapter mit der X4 BPMS entwickeln

Innerhalb von Adapter-Projekten lassen sich neue Adapter für die X4 BPMS mit Hilfe der verschiedenen Werkzeuge der X4 BPMS erstellen.

> ⓘ Um eigene Adapter zu erstellen und diese zu nutzen, wird eine ADK-Lizenz vorausgesetzt.

2.1 Die Struktur von Adapter-Projekten

Adapter-Projekte haben eine vordefinierte und nicht veränderbare Ordnerstruktur, die beim Anlegen eines neuen Projektes automatisch angelegt wird.

> ⓘ Die automatisch angelegten Ordner und Unterordner können nicht gelöscht, verschoben oder umbenannt werden.

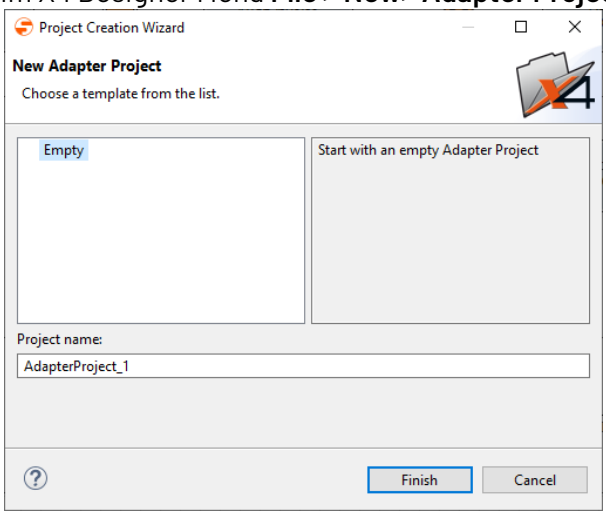
`Adapters`	Adapter, die in den technischen Prozessen verwendet werden, werden hier angelegt.
`Icons`	Hier werden die verschiedenen Adapter-Icons abgelegt.
`Operations`	Hier können beliebig viele technische Prozesse (`.wrf`) für die bereitzustellenden Operationen angelegt werden.
`Resources`	Hier können neben Ordnern auch Ressourcen wie Bilder, Text- und XML-Dokumente sowie weitere Dateien abgelegt werden.
`TemporaryFiles`	Hier werden temporär verwendete Dateien abgelegt.
`Transformations`	Hier können Transformationen (`.xsl`) sowie Reports (`.rep`) abgelegt werden
`<Projekt>.nad`	Adapter-Definition, die automatisch beim Anlegen des Projektes angelegt und nach dem Projekt benannt wird, z. B. *`AdapterProject_Documentation.nad`*. ⓘ Ein Projekt kann lediglich eine Adapter-Definition enthalten.

2.2 Adapter-Projekt anlegen

Adapter-Projekte lassen sich auf verschiedene Arten anlegen:

- Über **New Adapter** auf der Welcome-Seite des X4 Designers
- Über das Kontextmenü **New > Adapter Project** im Repository Navigator
- Über das Menü **File > New > Adapter Project**

1. Im X4 Designer Menü **File > New> Adapter Project** wählen.



2. Leere Projektvorlage oder eine vorhandene Projektvorlage wählen.
3. In **Project name** den Namen des Projekts eingeben.
4. **Finish** klicken, um das Projekt anzulegen.
   Ein leeres Adapter-Projekt mit der vordefinierten Struktur wird im Repository Navigator angelegt.

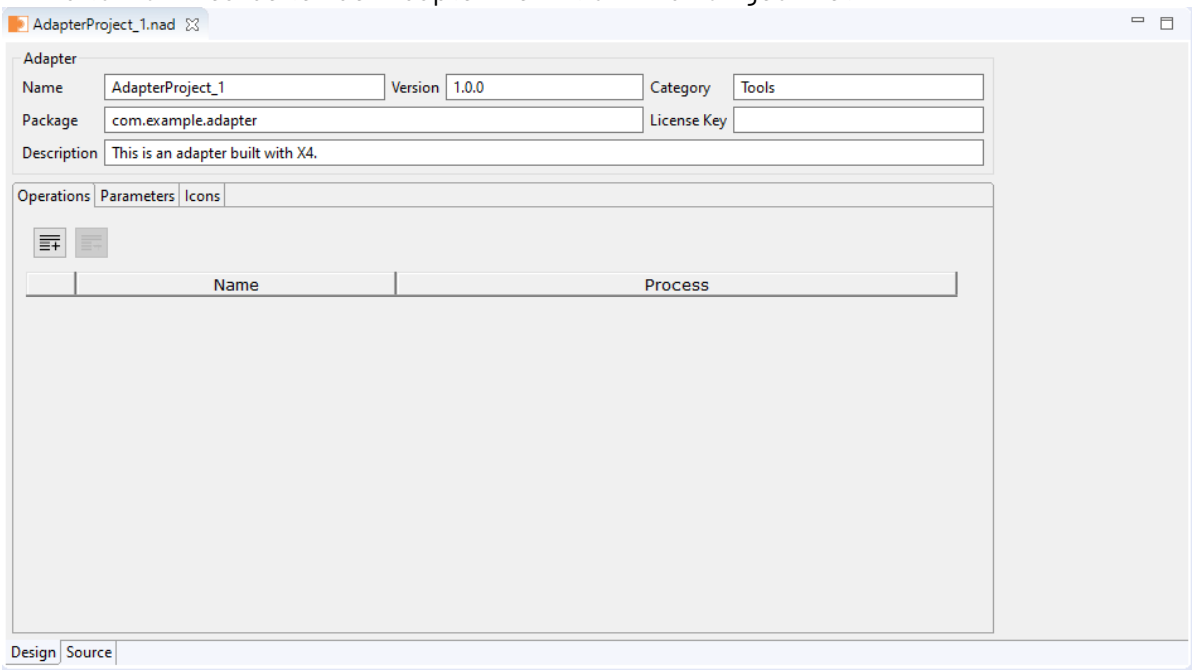
   > ⓘ Über das Kontextmenü können nun die verschiedenen Elemente eines Adapter-Projektes angelegt werden. Die verfügbaren Optionen ändern sich je nachdem an welcher Stelle in der Baumstruktur das Kontextmenü aufgerufen wird.

2.3 Adapterdefinition vornehmen

Die Schnittstelle des Adapters lässt sich über die Adapter-Definitionsdatei (`.nad`) definieren. Die später verfügbaren Adapter-Operationen werden dabei über technische Prozesse definiert.

1. Adapter-Projekt anlegen, siehe Adapter-Projekt anlegen.

2. Definitionsdatei `<Projektname>.nad` per Doppelklick öffnen.
   Ein Editor zum Bearbeiten der Adapter-Definition wird nun geöffnet.



3. In **Adapter**-Bereich die Grundeinstellungen des Adapters definieren. Folgende Einstellungen lassen sich vornehmen:
   - **Name:** Adapter-Name
   - **Version:** Adapter-Version
   - **Category:** Adapter-Kategorie, z.B. Tools
   - **Package:** Name des später bereitzustellenden WildFly-Moduls.

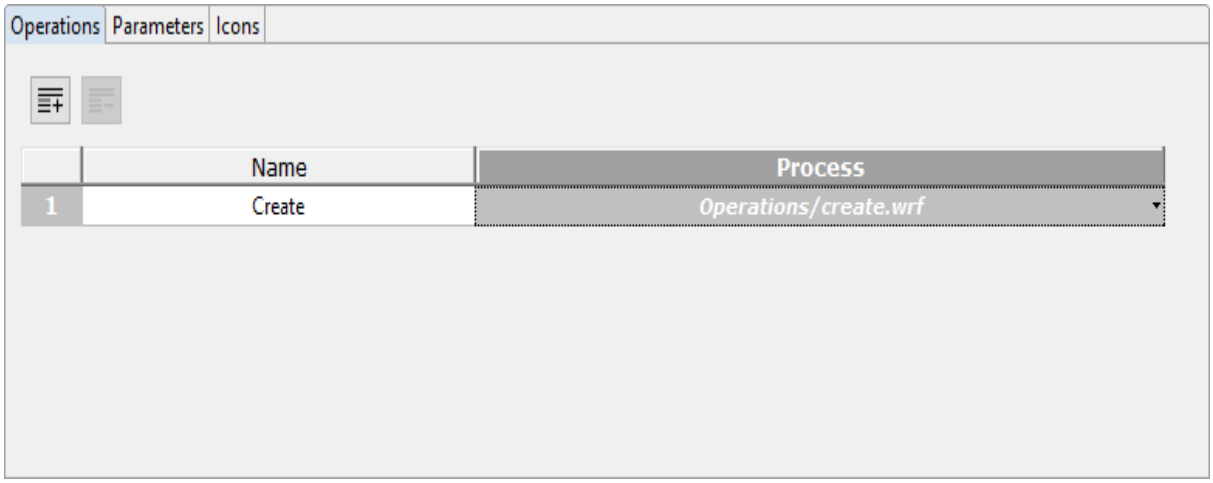
     Diese Einstellung ist für die Option **Package as Adapter...** relevant.

   - **License Key**: Lizenzschlüssel für den Adapter
   - **Description:** Adapter-Beschreibung
4. Unter **Operations** beliebig viele Adapter-Operationen mit **Add** hinzufügen.
   Die einzelnen Operationen müssen jeweils mit einem technischen Prozess verknüpft werden.

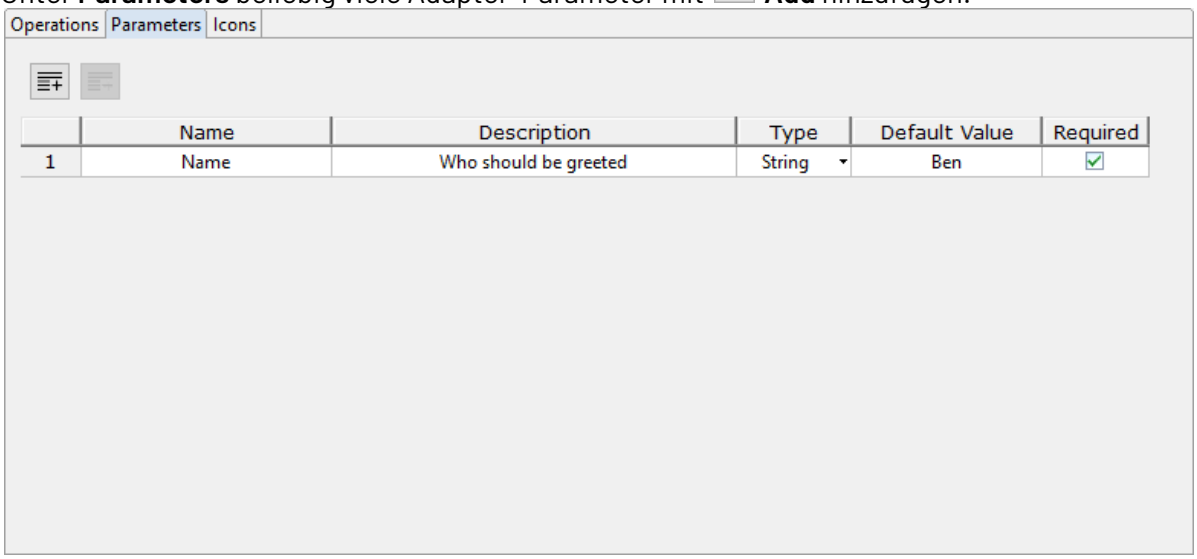
   - Um eine Operation zu löschen, die entsprechende Zeile markieren und **Remove** klicken.
   - Neu angelegte Prozesse werden automatisch geladen.

5. Zu jeder Operation den entsprechenden technischen Prozessen aus dem Ordner
   `Operations`

wählen.



6. Unter **Parameters** beliebig viele Adapter-Parameter mit **Add** hinzufügen.



7. Adapter-Parameter definieren. Folgende Einstellungen lassen sich vornehmen:
   - **Name**: Parameter-Name
   - **Description:** Parameter-Berschreibung
   - **Type:** Parameter-Typ
     - **String:** Für Zeichenketten
     - **Number:** Für Zahlen
     - **Boolean:** Für Boolsche-Werte
     - **Combo:** Für Auswahllisten

       > Namen und Werte für die möglichen Optionen einer Combo-Box können in den **Combo Options** angelegt werden. Die **Combo Options** erscheinen automatisch, sobald für einen Parameter der Typ **Combo** ausgewählt wird.

     - **Password:** Für Passwörter

> Werte vom Typ **Password** sind nicht in Plaintext und können nicht kopiert werden.

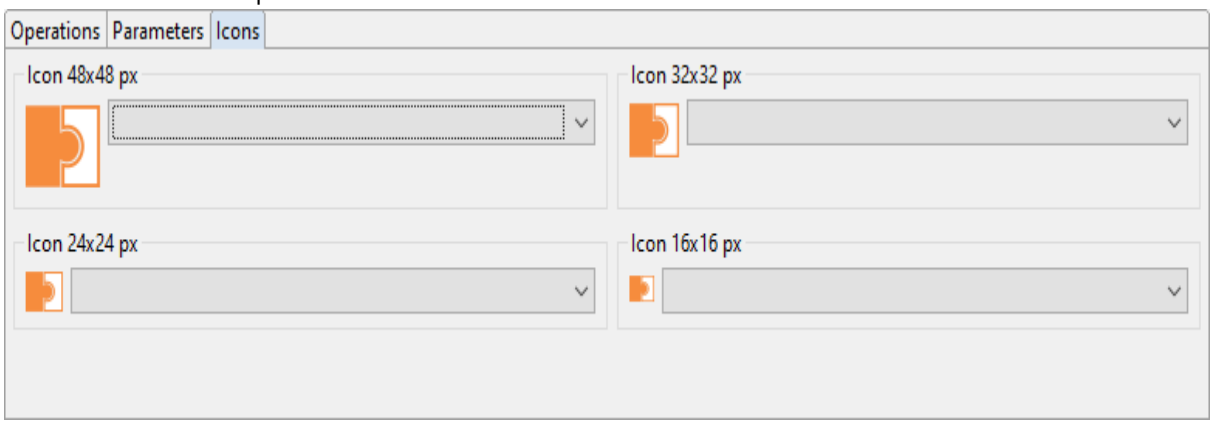
- **Default Value**: Standardwert des Parameters
- **Required:** Angaben, ob es ein Pflicht- oder Optionaler-Parameter ist

> Die Parameter stehen in den Prozessen als Platzhalter `PARAM_INPUT_<Name>` zur Verfügung.

8. Unter **Icons** die hinterlegten Bilder aus dem Ordner

   `Icons`

   als Adapter-Icons definieren.



> Neu hinzugefügte Icons werden automatisch geladen.

9. Adapter-Definition speichern.

2.4 Adapter-Projekt als Modul bereitstellen

Mit der X4 BPMS erstellte Adapter können wie Java-Adapter paketiert und in WildFly als Modul bereitgestellt werden. Über das Kontext-Menü des Adapter-Projekts steht die Funktion `Package as a Wildfly Module` zur Verfügung.

1. Adapter-Definition wie gewünscht vornehmen, siehe Adapterdefinition vornehmen
2. Das Kontextmenü auf der obersten Projektebene (Projektname) öffnen und **Package as Adapter** wählen.
   Ein Dialog zum Exportieren des Projektes wird nun geöffnet.
3. Gewünschten Speicherort wählen und in **Dateiname** den Adapternamen eingeben.
4. **Speichern** klicken, um das Adapter-Projekt als ZIP-Ordner zu exportieren.
5. Adapter anschließend auf dem Server bereitstellen, siehe Adapter auf dem X4 Server bereitstellen.

2.5 Adapter auf dem X4 Server bereitstellen

Um das fertige Adapter-Paket in die X4 Server-Konfiguration einzubinden, werden die kompilierten Adapter-Klassen sowie ggf. benötigte externe Bibliotheken im Applikations-Server als Java-Archiv (`.jar`-Datei) bereitgestellt.
Das Beispielprojekt aus dem ADK-Paket enthält bereits den entsprechenden Maven Build, der `.jar`-Datei und Deskriptoren in der erwarteten Verzeichnisstruktur erzeugt.

1. Die erzeugte `.zip`-Datei im Ordner `X4\Server\<wildfly>\modules\system\layers\base` entpacken.
2. Im `X4-Extensions`-Modul (`de.softproject.x4.extensions`) eine Referenz auf das eigene Modul des Adapters hinzufügen.
3. X4 Server neu starten.
   Der Adapter kann nun bei der nächsten Verbindung im X4 Designer aus der Adapter-Liste ausgewählt und in X4-Prozessen verwendet werden.

> ⚠ Das Adapter-Projekt und das bereitgestellte Adapter-Modul können nicht gleichzeitig auf dem gleichen X4 Server verwendet werden.

2.6 Prozesse mit eigenen Adaptern debuggen

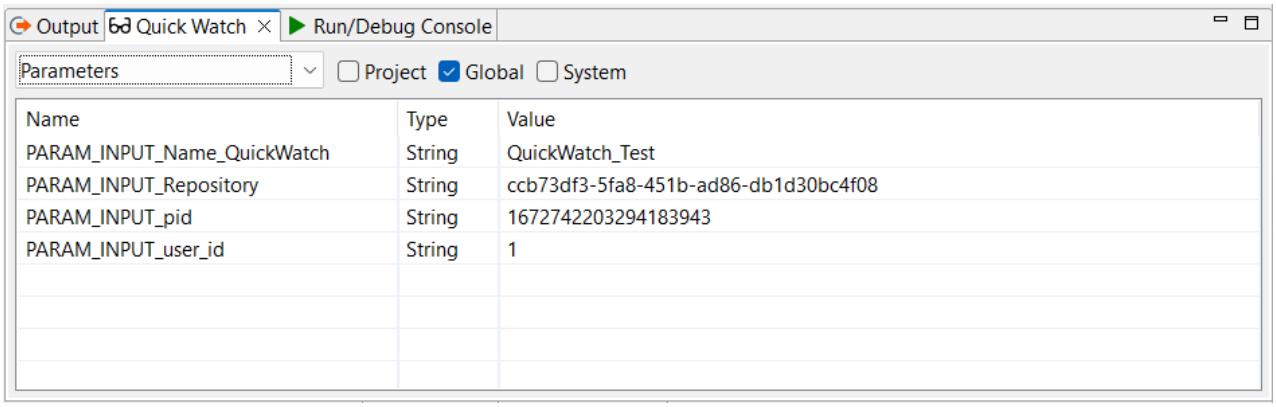
Beim Debuggen von Prozessen, die einen oder mehrere native Adapter enthalten, gehen Sie genauso vor wie beim Debuggen von Prozessen mit vorkonfigurierten Adaptern. Weitere Informationen hierzu finden Sie im Abschnitt Prozesse debuggen und ausführen.

Wenn Sie einen Prozess debuggen, der einen nativen Adapter enthält, werden in der Sicht **Quick Watch** auch die Input-Parameter des nativen Adapters angezeigt. Sie erkennen die Input-Parameter am Präfix `PARAM_INPUT` vor dem Parameternamen.

So zeigen Sie die Parameter nativer Adapter in der Sicht Quick Watch an:

- Wählen Sie in der Dropdown-Liste den Eintrag **Parameters** aus, um alle Parameter anzuzeigen.

> ⓘ Wenn Sie die angezeigten Parameter weiter eingrenzen möchten, aktivieren Sie das entsprechende Kontrollkästchen. Aktivieren Sie z. B. **Global**, um nur globale Parameter anzuzeigen. Die Input-Parameter nativer Adapter gehören zur Kategorie der globalen Parameter.



Weitere Informationen:

- Abschnitt Sichten im Kapitel mit der Beschreibung der Oberfläche des X4 Designers
- Abschnitt Parameter erstellen im Kapitel Prozesse anlegen und modellieren

3 Eigene Adapter in Java entwickeln

Wie Sie einen Adapter in Java mithilfe des Adapter-Frameworks von X4 ADK entwickeln.

3.1 Grundlegendes zu Adaptern

Adapter sind Prozessbausteine, die grundlegende Funktionen bereitstellen, etwa Datenkonvertierungen oder Datenaustausch mit Drittsystemen. Da X4 ESB hauptsächlich mit XML-Daten arbeitet, werden in technischen Prozessen mit Adaptern meist XML-Dokumente erzeugt oder verändert. Ein Adapter benötigt in der Regel einen Input und gibt meist ein XML-Dokument aus, das im nächsten Prozess-Schritt weiterverarbeitet wird.

Eigene Adapter zu implementieren ist die schnellste und einfachste Möglichkeit, verschiedene Anwendungen in die X4 BPMS zu integrieren und den Funktionsumfang der X4 BPMS zu erweitern. Die X4 BPMS stellt hierzu ein Framework bereit, das die Adapter-Entwicklung in vielerlei Hinsicht vereinfacht.

3.2 ADK-Umgebung einrichten

Welche Entwicklungsumgebung vorausgesetzt wird und wie sich diese einrichten lässt, wird im Folgenden beschrieben.
Um einen Adapter für die X4 BPMS zu entwickeln, ist es zunächst erforderlich, die notwendigen Klassen bereitzustellen und ggf. die Java-Entwicklungsumgebung (beispielsweise *Eclipse*) mit einer Java-Laufzeitumgebung (JRE) in Version 1.8 oder höher einrichten.

> ⓘ Das ADK-Paket mit den notwendigen X4-Java-Klassen und Beispielen erhalten Sie über den SoftProject-Support via support@softproject.de.

`adk-example.zip` entpacken und als existierendes Maven-Projekt in Ihrer Java-IDE importieren.

Das Java-Projekt enthält nun den Ordner `src/main/java` (Code-Beispiele).

3.3 Adapter-Interface implementieren

Wie Sie über das einheitliche Adapter-Interface das Grundgerüst für eine Adapter-Implementierung erstellen

3.3.1 Interface und Konstruktor

Alle Adapter implementieren das Interface `de.softproject.integration.adapter.core.Adapter` und den Standard-Konstruktor (ohne Parameter) zum Erstellen der Hauptklasse. Die abstrakte Klasse

`de.softproject.integration.adapter.core.BaseAdapter` enthält eine Basisimplementierung, um den Adapter-Status und den Zugriff im Projektverzeichnis zu erhalten.

3.3.2 Adapter-Metadaten definieren

Um Adapter-Metadaten zu definieren wird die Annotation `@AdapterDescription` verwendet. Folgende Felder müssen gesetzt sein:

Feld	Beschreibung	Pflicht
`category`	Fachliche Kategorie des Adapters	ja
`description`	Beschreibung über den Zweck des Adapters	ja
`displayName`	Sprechender Name des Adapters	ja
`version`	Version des Adapters im Format `major.minor.patch` (z.B. *`1.0.0`*)	ja
`configurationBeanClass`	Typ der `ConfigurationBean`, falls der Adapter Parameter benötigt. Ist die Konfiguration in der Adapterklasse selbst, muss hier nichts angegeben werden.	nein

3.3.3 BaseAdapter verwenden

Diese Basisklasse enthält geschützte Methoden für den Zugriff auf den Adapter-Status und den ProjectExplorer. Die Basisimplementierung geht davon aus, dass die Konfiguration in der Adapterklasse vorgenommen wird. Das heißt die Adapter-Parameter sind in der Klasse selbst definiert. Wenn eine eigene Klasse zur Adapter-Konfiguration verwendet werden soll, muss die Methode `getConfigurationBean` wie unter *Adapter-Metadaten definieren* beschrieben überschrieben werden.

3.3.4 Adapter-Interface verwenden

1. Eine neue Adapter-Klasse erstellen, die das Interface `de.softproject.integration.adapter.core.Adapter` implementiert.
2. Standard-Konstruktor verwenden. Die Klasse wird vom X4 ESB zur Laufzeit über den Standard-Konstruktor instantiiert.
   Die Methoden `init()` und `cleanup()` sowie `getConfigurationBean()` werden angelegt sowie die notwendigen Pakete importiert. Mit dem Konstruktor werden folgende Elemente angelegt:
   - `public void cleanup()`: Räumt *immer* nach dem Ausführen der `operation`-Methode auf und schließt Ressourcen, die von der Adapterinstanz belegt wurden.
   - `public void init(Status arg0, long arg1)`: Nach dem Anlegen der Adapter-Instanz wird eine Status-Instanz angelegt und an den Adapter mit dem Aufruf der `init()`-Methode übergeben. Parameter `arg0` gibt das Objekt an, für dessen Adapter-Instanz ein Status gesetzt werden soll, Parameter `arg1` bezeichnet die Adapter-Instanz (im Code).

Der Adapter kann bei dessen Ausführung in seinen do…-Methoden die Status-Instanz manipulieren, anschließend wird dieser Status vom X4 ESB ausgewertet.

- `public Object getConfigurationBean()`: Liefert eine Instanz der Configuration Bean zurück, die Setter-Methoden zum Setzen von Konfigurationsparameter enthält. Adapter-Instanzen können durch das Configuration-Bean-Objekt auf die Konfigurationsparameter zugreifen.

3. Statusvariable anlegen und Instanz-Status zuweisen.
4. Configuration-Bean-Objekt (optional) anlegen (siehe Configuration Bean erstellen), und in `getConfigurationBean()` als Rückgabewert verwenden bzw. bei Verzicht auf ein Configuration-Bean-Objekt `null` zurückgeben.
   Das Grundgerüst für die Hauptklasse des Adapters ist nun fertig.

3.3.5 Beispiel

```
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 Configuration Bean erstellen

Wie Sie ein Grundgerüst für eine Configuration Bean erstellen

3.4.1 Zweck einer Configuration Bean

Eine Configuration Bean dient falls erforderlich zum Bündeln von Konfigurationsparametern einer Adapterinstanz und enthält Methoden zum Setzen der Konfigurationsparameter. Adapterinstanzen können über die Methode `getConfigurationBean()` auf das Configuration-Bean-Objekt zugreifen.

3.4.2 Configuration Bean anlegen

1. Neue ConfigurationBean-Klasse im Paket anlegen.
2. Variablen definieren, die als Konfigurationsparameter dienen (nur vom Typ `String` möglich).
3. Die Annotation `@AdapterParameter` deklariert ein Feld als Adapterkonfiguration. Es kann entweder eine eigene Configuration-Bean-Klasse geben, in der alle Parameter definiert sind, oder die Definition kann in der Adapterklasse selbst vorgenommen werden (eine Mischung ist nicht möglich!).
4. `Set/Get`-Methoden für jeden Konfigurationsparameter definieren, die nach dem Muster `set<Parametername>(<Wert>)` bzw. `get<Parametername>()` benannt sein müssen.
   Das Grundgerüst der Configuration Bean ist nun fertig.

3.4.3 Metadaten für Adapter-Parameter definieren

Um Metadaten für Adapter-Parameter zu definieren, wird die Annotation `@AdapterParameter` verwendet:

Feld	Beschreibung	Pflicht
`description`	Textuelle Beschreibung, welchen Zweck der Parameter erfüllt	ja
`dataType`	Typ des Parameters *Mögliche Werte*: `PASSWORD`, `BOOLEAN`, `STRING`, `ARRAY`, `DOUBLE`, `INTEGER`	ja
`displayName`	Anzeigename der Parameters. Standard ist der Name des Felds	nein

Feld	Beschreibung	Pflicht
`optional`	Angabe, ob es sich um ein Pflichtparameter handelt *Mögliche Werte*: • *`true`*: der Parameter ist kein Pflichtparameter • *`false`*: der Parameter ist ein Pflichtparameter (Standard)	nein
`defaultValue`	Setzt einen initialen Wert für den Parameter	nein

ⓘ Die Annotation `@AllowedPropertyValue` muss zusätzlich verwendet werden, wenn `dataType` auf *`ARRAY`* gesetzt ist, um die möglichen Werte zu setzen.

Feld	Beschreibung
`displayName`	Gibt den Anzeigename des Wertes an
`value`	Technischer Bezeichner, der in das Feld bei Selektion geschrieben wird

3.4.4 Beispiel

```
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 Operationen festlegen

Wie Sie für einen Adapter eine oder mehrere Operationen definieren

3.5.1 Operationen für Adapter

Adapter bieten bestimmte Funktionen, die sich im *X4 Designer* in der Properties-Sicht über die Eigenschaft `Operation` auswählen lassen. Jeder Listeneintrag dieses Auswahlmenüs entspricht einer Operation-Methode im Adapter.

> ⓘ Ein Adapter benötigt mindestens eine Operation. Verfügt der Adapter nur über eine einzige Operation, so wird diese im Prozess-Designer automatisch ausgewählt.

3.5.2 Methoden zu Operationen zuordnen

1. Für jede Operation die Annotation `@AdapterOperation` für die Deklaration an eine `public`-Methode in der Adapter-Klasse angeben.
2. Parameter definieren.

   - Jede Methode kann optional einen Parameter definieren. Dieser entspricht dem Adapter-Input.
   - Der Rückgabetyp der Methode kann `X4Document` oder eine JAXB-konforme Java-Bean sein. Ist der Rückgabetyp `void`, wird der Input als Output betrachtet.
   - Um Java-Beans benutzen zu können, muss der Input XML und konform zur Java-Bean-Definition sein. Andernfalls wird die Adapter-Ausführung mit dem Status *`-1`* abgebrochen.

   Das Grundgerüst einer Operation-Methode ist nun fertig.

3.5.3 Metadaten für Adapter-Operation definieren

Um Metadaten für Adapter-Operationen zu definieren, wird die Annotation `@AdapterOperation` verwendet:

Feld	Beschreibung
`name`	Gibt den Namen der Operation an und kann frei gewählt werden. Er muss jedoch eindeutig pro Adapter sein.

3.6 Adapter auf dem X4 Server bereitstellen

Um das fertige Adapter-Paket in die X4 Server-Konfiguration einzubinden, werden die kompilierten Adapter-Klassen sowie ggf. benötigte externe Bibliotheken im Applikations-Server als Java-Archiv (`.jar`-Datei) bereitgestellt.
Das Beispielprojekt aus dem ADK-Paket enthält bereits den entsprechenden Maven Build, der JAR-Datei und Deskriptoren in der erwarteten Verzeichnisstruktur erzeugt.

1. *Maven clean install* ausführen.
2. Die generierte ZIP-Datei `target/adk-examples-1.0.0-SNAPSHOT-wildfly-module.zip` in dem Ordner `X4\Server\<wildfly>\modules\system\layers\base` entpacken.
3. Im `X4-Extensions`-Modul (`de.softproject.x4.extensions`) eine Referenz auf das eigene Modul des Adapters hinzufügen.
4. X4 Server neu starten.
   Der Adapter kann nun bei der nächsten Verbindung im X4 Designer aus der Adapter-Liste ausgewählt und in X4-Prozessen verwendet werden.

3.7 Symbol für Adapter hinterlegen

Für bestehende und selbst entwickelte Adapter können Sie benutzerdefinierte Symbole hinterlegen. Diese werden aus dem X4 Repository geladen und im X4 Designer anstatt des Standard-Adapter-Symbols im Prozessdiagramm angezeigt.
Dabei können Sie entweder Symbole für einzelne Adapter oder für die verschiedenen Adapter-Typen definieren.

3.7.1 Standardsymbole

Für Standard-Icons stehen Annotationen zur Verfügung, die an die Adapterklasse annotiert werden können.

Adapter-Typ	Symbol	Annotation
Converter		@ConverterIcon
Transfer		@TransferAdapterIcon
Connector		@ConnectorIcon

Ist keine Annotation angegeben, wird das Function-Adapter-Icon  verwendet.

3.7.2 Symbol bereitstellen

Zur Bereitstellung des Adapter-Symbols ist die Annotation `@AdapterIcon` zu verwenden. Diese kann an die Adapterklasse geschrieben werden.

Es können Größen von 16x16, 24x24, 32x32 und 48x48 angegeben werden. Mindestens eine Größe muss angegeben werden, dann werden die anderen Größen aus der größten Bildgröße berechnet. Als Wert muss ein Klassenpfad zur Bildresource in `src/main/resources` angegeben werden.

Beispiel: In `src/main/resources/HelloWorld/icons` liegt das Bild `icon_16.png`, das als Adapter-Icon verwendet werden soll. Der entsprechende Pfad ist */HelloWorld/icons/icon_16.png*.

4 Adapter-Framework-Referenz

Welche Komponenten des *X4 ADK* dem Adapter-Entwickler zur Verfügung stehen

4.1 Adapter-Interface

Wie das Adapter-Interface, der Konstruktor und die Methoden verwendet werden

Alle Adapter verwenden das Interface `de.softproject.integration.adapter.core.Adapter` und den Standard-Konstruktor (ohne Parameter) zum Erstellen der Hauptklasse. Jede Adapterinstanz besitzt ein `status`-Objekt, um nach dem Ausführen des Adapter-Bausteins einen Status zu setzen, sowie einen eindeutigen Bezeichner.

4.1.1 Adapter-Interface-Methoden

de.softproject.integration.adapter.core.Adapter	
`public void cleanup()`	Räumt immer nach dem Ausführen der `operation`-Methode auf und schließt Ressourcen, die von der Adapterinstanz belegt wurden.
`public void init(Status arg0, long arg1)`	Wird nach dem Anlegen der Adapterinstanz aufgerufen. Parameter `arg0` nennt das Objekt, für dessen Adapterinstanz ein Status gesetzt werden soll, Parameter `arg1` nennt die Bezeichnung der Adapterinstanz (im Code).
`public Object getConfigurationBean()`	Gibt eine Instanz der Configuration Bean zurück, die Setter-Methoden zum Setzen von Konfigurationsparameter enthält. Adapterinstanzen können durch das Configuration-Bean-Objekt auf die Konfigurationsparameter zugreifen.

4.2 X4Document-Interfaces

Welche Methoden für den Zugriff auf Input-Dokumente, die Verarbeitung von Dokumenten und das Erzeugen von Dokumenten innerhalb von Adaptern zur Verfügung stehen

Prozessbausteine, also Instanzen z. B. eines Adapters können zahlreiche Dokument-Arten einlesen und ausgeben: XML-Dokumente, aber auch Text- oder Binärdaten. Über das `X4Document`-Interface können Adapter auf Dokumente zugreifen bzw. mit der `X4DocumentFactory` als Ergebnis der Adapter-Operation ein Ausgabe-Dokument erzeugen. Dies kann eine `X4Document`-Instanz von einem *dom4j*-Baum sein, ein `java.io.InputStream` oder ein Byte-Array.

Wenn in *X4*-Prozessen XSL-Transformationen einen DOM-basierten Input erhalten, dann lassen sich Zeichenkodierungs-Probleme vermeiden. Bei Byte-basierten Dokumenten wird das Byte-Array ohne Zusatzinformationen an den XML-Parser gegeben.

4.2.1 Methoden von X4Document

de.softproject.integration.util.x4documents.X4Document	
public byte[ ] getAsByteArray()	Gibt den Inhalt als Byte-Array zurück
public byte[ ] getAsByteArray(String encoding)	Gibt den Inhalt als Byte-Array mit in encoding angegebener Zeichenkodierung zurück
public java.io.InputStream getAsInputStream()	Gibt den InputStream vom Zugriff auf den Dokument-Inhalt zurück
public org.dom4j.Document getAsDocument() throws org.dom4j.DocumentException	Gibt den Inhalt als *dom4J*-DOM-Baum zurück
public org.dom4j.Document getAsDocument(String encoding) throws org.dom4j.DocumentException, java.io.UnsupportedEncodingException	Gibt den Inhalt als *dom4J*-DOM-Baum mit in encoding angegebener Zeichenkodierung zurück
public void parseWithHandler (ContentHandler chandler) throws SAXException, IOException	Gibt den Inhalt an bereitgestellte org.sax.ContentHandler-Instanz weiter (flush)
public String getMimeType()	Gibt den MIME-Typ des Dokuments zurück
public String getEncoding()	Gibt die Zeichenkodierung des Dokuments zurück
public void setMimeType(String mimeType)	Setzt den MIME-Typ des Dokuments
public void setEncoding(String encoding)	Setzt die Zeichenkodierung des Dokuments

4.2.2 Methoden der X4DocumentFactory

de.softproject.integration.util.x4documents.X4DocumentFactory	
public static X4Document createX4Document(byte[ ] bytes, String encoding, String mimeType)	Gibt die `X4Document`-Instanz mit Byte-Array-Inhalt `bytes` aus, das Zeichenkodierung `encoding` und MIME-Typ `mimeType` besitzt
public static X4Document createDocument(Document doc, String encoding, String mimeType)	Gibt die `X4Document`-Instanz mit *dom4J*-Baum-Inhalt `doc` aus, die Zeichenkodierung `encoding` und MIME-Typ `mimeType` besitzt
public static X4Document createX4Document(InputStream is, String encoding, String mimeType) throws X4DocumentException	Gibt die `X4Document`-Instanz mit `java.io.InputStream`-Inhalt `is` aus, die Zeichenkodierung `encoding` und MIME-Typ `mimeType` besitzt

4.2.3 Beispiel

```
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 chandler = new SAXContentHandler();
            x4doc.parseWithHandler(chandler);
            System.out.println(chandler.getDocument().asXML() + "\n");
        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            if (is != null)
                try {
                    is.close();
                } catch (IOException ioex) {
                }
        }
    }
}
```

4.3 ProjectExplorer-API zum Zugriff auf das Projekt

Über die ProjectExplorer-API lässt sich auf Dateien des Projektes zugreifen in dem der Adapter aufgerufen wird.

Methode	Beschreibung
`String getProjectName()`	Gibt den Namen es Projekts zurück
`LocalDateTime getLastModifiedTime(ProjectPath path)`	Gibt den letzten Änderungszeitpunkts der Datei zurück
`InputStream read(ProjectPath path)`	Gibt den Inhalt der Datei als Input-Stream zurück
`List<ProjectPath> getProjectContent()`	Gibt alle Dateien als Pfade des Projekts zurück
`void write(ProjectPath path, InputStream inputStream)`	Schreibt den Datenstrom in die angegebene Position und erstellt ggf. eine neue Datei, sofern die Regeln der Projektart dies erlauben
`boolean exists(ProjectPath path)`	Prüft, ob die Datei oder der Ordner existieren
`void create(ProjectPath path)`	Erzeugt ein Verzeichnis
`void delete(ProjectPath path)`	Löscht ein Verzeichnis oder eine Datei
`void move(ProjectPath sourcePath, ProjectPath targetPath)`	Verschiebt eine Datei oder ein Verzeichnis an die angegebene Stelle gemäß den Regeln der Projektart
`void copy(ProjectPath sourcePath, ProjectPath targetPath)`	Kopiert eine Datei oder ein Verzeichnis an die angegebene Stelle gemäß den Regeln der Projektart
`List<ProjectPath> getFolderContent(ProjectPath folderPath)`	Gibt alle Pfade in dem Ordner und aller Unterordner zurück

4.4 Adapter entwickeln

Für sämtliche Adapter lassen sich über die Klasse `de.softproject.integration.adapter.core.Status` Status-Werte setzen, die in *X4*-Prozessen von `Condition`-Prozessbausteinen ausgewertet werden können. Jede ausgeführte Operation-Methode eines Adapters kann einen entsprechenden Status setzen.
Dabei können die drei Standard-Status-Werte `OK (1)`, `ERROR (-1)` und `TIMEOUT_EXCEEDED (0)` oder ein benutzerdefinierter ganzzahliger Status-Wert gesetzt werden. Wird im Adapter kein Status gesetzt, dann werden auftretende Exceptions beim Ausführen der Operation-Methode als `ERROR (-1)` behandelt, ansonsten wird der Status `OK (1)` zurückgegeben.

4.4.1 Status-Methoden

de.softproject.integration.adapter.core.Status	
public void setOk()	Setzt den Status der Operation-Methode auf *erfolgreich ausgeführt*
public void setError()	Setzt den Status der Operation-Methode auf *fehlerhaft ausgeführt*
public void setTimeoutExceed()	Setzt den Status der Operation-Methode auf *nicht innerhalb der gegebenen Zeit ausgeführt*
public void setStatus(int Status)	Setzt den Status-Code Nicht erlaubte Werte sind: *-999*, *-2* und *999*!
public boolean isOk()	Prüft, ob der Status auf *erfolgreich ausgeführt* (*1*) gesetzt ist
public boolean isError()	Prüft, ob der Status auf *fehlerhaft ausgeführt* (*-1*) gesetzt ist.
public boolean isTimeoutExceed()	Prüft, ob der Status auf *nicht innerhalb der gegebenen Zeit ausgeführt* gesetzt ist
public int getStatus()	Liest den Status-Wert als ganze Zahl
public static final int OK=1	Assoziiert den Status `OK` mit dem Wert *1*
public static final int ERROR=-1	Assoziiert den Status `ERROR` mit dem Wert *-1*
public static final int TIMEOUT_EXCEED=0	Assoziiert den Status `TIMEOUT_EXCEED` mit dem Wert *0*

4.5 Exceptions und Fehlerbehandlungen

Eine wichtige Möglichkeit, bei Adaptern mit Fehlern umzugehen, ist Exceptions zu werfen. Die Klasse `de.softproject.integration.adapter.core.AdapterException` ermöglicht eine Zuordnung eines Fehlercodes und einer Nachricht zu der Root-Exception.
Wenn dem Adapter Parameterwerte fehlen, was beim Modellieren von Prozessen häufig vorkommt, können diese eine `de.softproject.integration.adapter.core.MissingParameterException` werfen.

Konfigurationsprobleme sollten Adapter durch das Werfen von `de.softproject.integration.adapter.core.AdapterConfigurationException` signalisieren.

4.5.1 Methoden von AdapterException

de.softproject.integration.adapter.core.AdapterException	
public AdapterException(int reasonCode, String msg)	Konstruktor für eine Adapter-Exception mit reasonCode (Zahlencode zur Identifikation des Fehlerfalls) und msg (zur Beschreibung des Fehlerfalls)
public AdapterExeception(String msg, Exception ex)	Konstruktor für eine Adapter-Exception mit msg (zur Beschreibung des Fehlerfalls) und ex (Root-Exception)
public AdapterException(int reasonCode, String msg, Exception ex)	Konstruktor für eine Adapter-Exception mit reasonCode (Zahlencode zur Identifikation des Fehlerfalls), msg (zur Beschreibung des Fehlerfalls) und ex (Root-Exception)

4.5.2 Methoden von MissingParameterException

de.softproject.integration.adapter.core.MissingParameterException	
public MissingParameterException(int reasonCode, String msg)	Konstruktor für eine MissingParameter-Exception mit `reasonCode` (Zahlencode zur Identifikation des Fehlerfalls) und `msg` (zur Beschreibung des Fehlerfalls)

4.5.3 Methoden von AdapterConfigurationException

de.softproject.integration.adapter.core.AdapterConfigurationException	
public AdapterConfigurationException(int reasonCode, String msg)	Konstruktor für eine AdapterConfiguration-Exception mit `reasonCode` (Zahlencode zur Identifikation des Fehlerfalls) und `msg` (zur Beschreibung des Fehlerfalls)