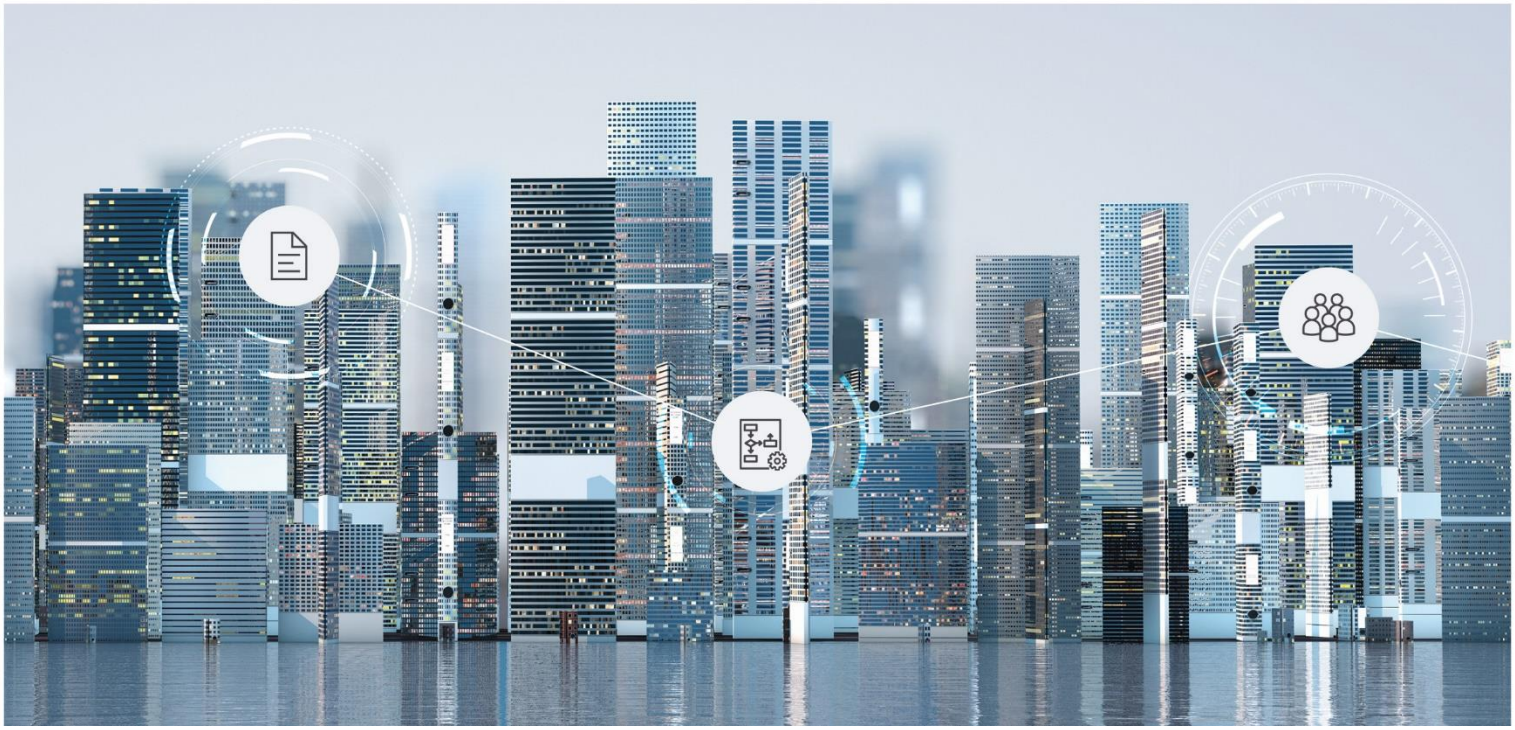
Fabasoft



White Paper

Fabasoft app.ducx Extensions for Java Developers

2022 June Release

Contents

1 Introduction

Fabasoft app.ducx is Fabasoft's powerful use case-oriented application development system for developing composite content applications (CCAs).

The Fabasoft app.ducx reference documentation series consists of the following three consecutive documents:

- Fabasoft app.ducx for Cloud Developers
  This document explains all concepts and use cases that are relevant for Cloud and Folio developers. This document is aimed at software developers interested in exploring the tremendous potential of Fabasoft app.ducx.
- Fabasoft app.ducx Extensions for Folio Developers
  This document describes extensions that are relevant for Folio developers. It is highly recommended to read "Fabasoft app.ducx for Cloud Developers" beforehand, because all the presented concepts are also directed at Folio developers.
- **Fabasoft app.ducx Extensions for Java Developers**
  This document provides an overview of using Java in your app.ducx project. Java can only be used along with Folio. It is highly recommended to read "Fabasoft app.ducx Extensions for Folio Developers" beforehand.
  Note: You are currently reading this document.

2 Working With Fabasoft app.ducx Projects Using Eclipse

If you want to use Java for implementing use cases, you have to consider following additional aspect when working with app.ducx projects.

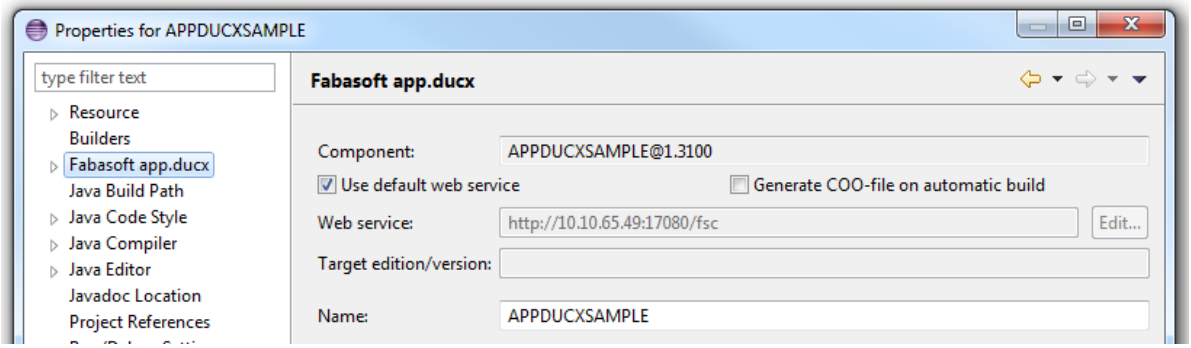
2.1 Adding Additional Contents (JARs etc.)

Additional contents may be specified within the Fabasoft app.ducx project settings. It is solely for additional contents, e.g. help contents (Compiled Help Files) or contents for a specific platform, e.g. Windows x64 only. These contents will be part of your software component.

Contents are simply added by dropping a bunch of files from the file system. Additional properties may be set after drop. The basename is initially constructed from the basename of the file. The type is `CCT_NORMAL` by default. The basename may be changed.

Fabasoft app.ducx tries to identify available variables. The longest match is used. Variables include environment variables and variables available within in Eclipse.

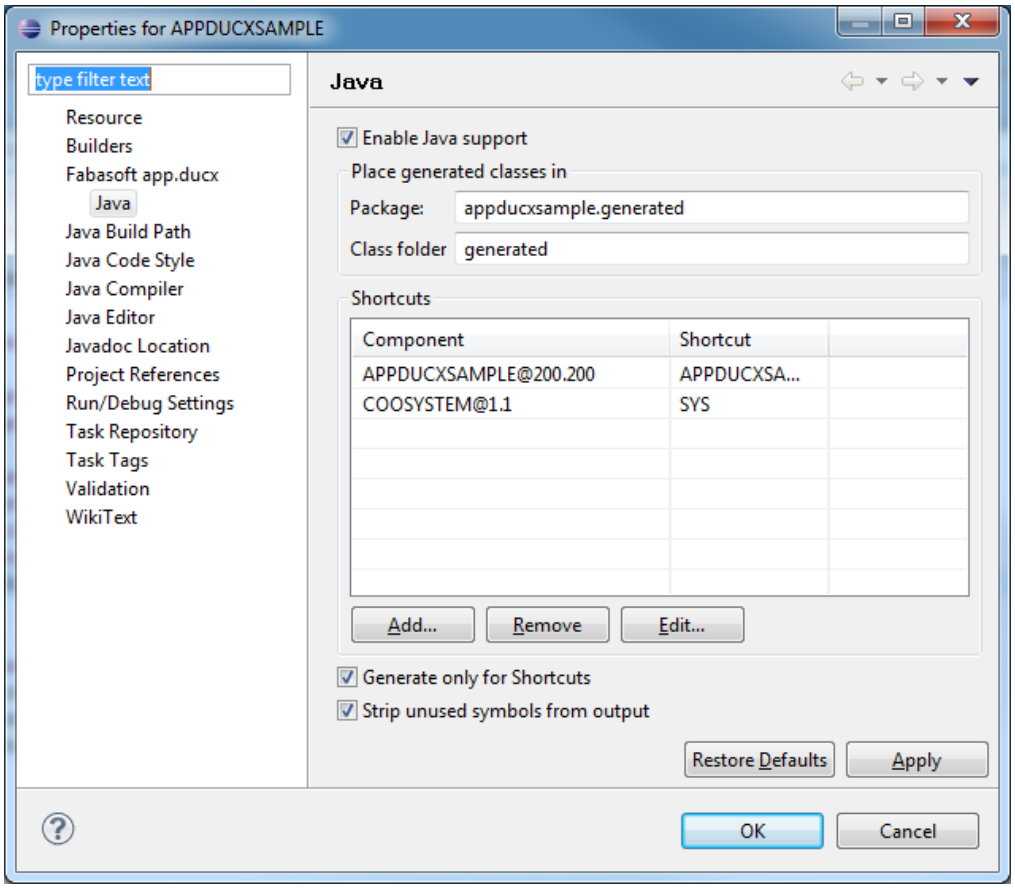
Note: When using a build process out of eclipse, Eclipse variables might not be available.



Fabasoft app.ducx projects using Java implementations also provide a Java archive which contains the implementations of the dedicated actions and use cases. This archive is created by a separate builder which runs only when the project is uploaded to the specified Fabasoft Folio domain.

3 Implementing a Use Case in Java

Before you can start to implement use cases in Java, you have to enable Java support for your Fabasoft app.ducx project. You can enable Java support by selecting *Enable Java support* when creating a new Fabasoft app.ducx project. Java support may also be enabled later in the "Properties" dialog box of your Fabasoft app.ducx project (see next figure).



When enabling Java support, you have to provide a *Package* name and a *Class folder* for the classes generated automatically by Fabasoft app.ducx to allow you to access the Fabasoft Folio object model in Java. These classes are encapsulated in a Java package and put in a class folder, which is then added to your Fabasoft app.ducx project. The files generated by Fabasoft app.ducx are updated automatically whenever you modify your object model or your use case definitions.

It is possible to generate Java classes only for software components that are defined as shortcuts. Shortcuts for `COOSYSTEM@1.1` and the own software component are mandatory and created automatically if not already defined. This way the compiling performance can be improved.

Strip unused symbols from output should be used for a release build to reduce the size of the JAR file. No stripping of unused symbols is an advantage in the development process because the project compiles faster.

Note:

- JAR files of other software components in the Fabasoft Folio Domain are fetched when executing "Update All References". These JAR files can be referenced within the Fabasoft app.ducx project (“Properties” > “Java Build Path” > “Libraries” > “Add JARs”). The JAR files are located in the `.references\lib` folder. In a newly created project it may be necessary to refresh the project to be able to select the JAR files.
  Because the JAR files of the Fabasoft Folio Domain are used, no upload of these files is needed.
- In the following examples, the package name `APPDUCXSAMPLE.generated` is used for the class files generated by Fabasoft app.ducx.

Fabasoft app.ducx projects with java are Java projects, too. For compiling with Java, the appropriate JRE for the target Folio platform needs to be installed and selected in Eclipse.

3.1 Defining a Use Case to Be Implemented in Java

The app.ducx use case language must be used to define the use case.

Syntax

```
impl = java:package.class.method;
```

After the `java:` prefix, specify the package name for your Java implementation package, followed by the reference of the object class, and the name of the method. The individual parts must be separated by periods.

Example

```
usecases APPDUCXSAMPLE@200.200
{
  import COOSYSTEM@1.1;
  usecase CreateInvoice(out Invoice invoice) {
    variant Order {
      java = APPDUCXSAMPLE.Order.CreateInvoice;
    }
  }
}
```

3.2 Implementing the Java Method

When defining a use case implemented as a Java method, Fabasoft app.ducx will offer to generate a Java class and a method stub for your use case in the Java class on behalf of you.

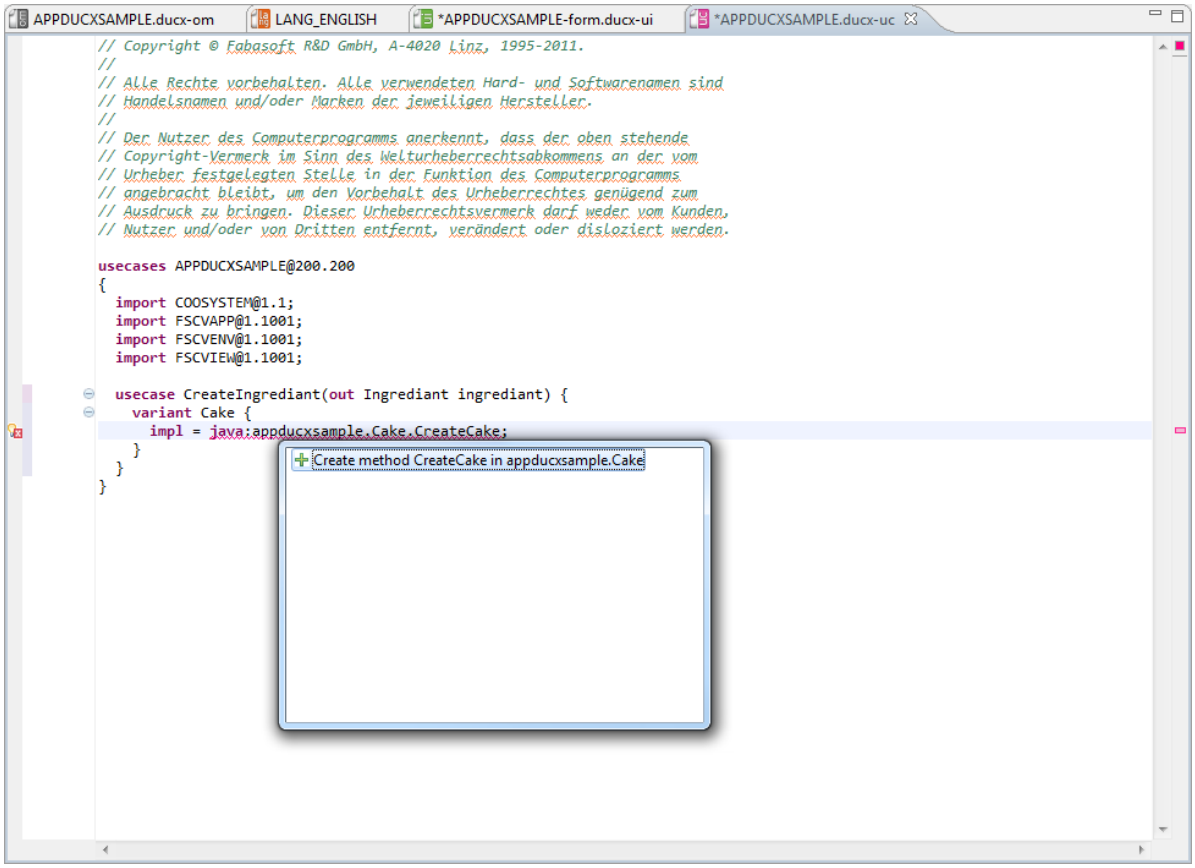
After providing the name of the java method in the use case file, an error symbol is shown on the left-hand side next to the corresponding line in the source code indicating that the Java method does not exist yet. A left-click on the error symbol opens a quick fix (see next figure).

In the Fabasoft app.ducx wizard dialog box, double-click “Create method” to let the Fabasoft app.ducx wizard create a Java method stub for your use case.

Java methods may also be defined manually without the help of the app.ducx wizard. However, please note that all Java methods invoked from a use case must be attributed in the following format:

```
@DUCXImplementation("<Fully Qualified Reference of the Use Case>")
```

Furthermore, you may add your own member variables and methods that are not invoked by a use case to the Java classes generated by the Fabasoft app.ducx wizard as well.



The following example illustrates the Java source code generated by the Fabasoft app.ducx wizard for a typical Java method stub.

Example

```
package APPDUCXSAMPLE;
import CooLib.CooObject;
import CooLib.CooException;
import static APPDUCXSAMPLE.generated.DUCX.coort;
import static APPDUCXSAMPLE.generated.DUCX.cootx;
import APPDUCXSAMPLE.generated.DUCXImplementation;
import APPDUCXSAMPLE.generated.APPDUCXSAMPLE_200_300.actions.CreateInvoiceResult;
public class Order extends APPDUCXSAMPLE.generated.APPDUCXSAMPLE_200_300.
  classes.Order {
  public Order(CooObject obj) {
    super(obj);
  }
  @DUCXImplementation("APPDUCXSAMPLE@200.200:CreateInvoice")
  public CreateInvoiceResult CreateInvoice() throws CooException {
    CreateInvoiceResult result = new CreateInvoiceResult();
    // TODO: Auto-generated implementation stub
    return result;
  }
}
```

For a comprehensive example demonstrating how to implement a use case in Java, please refer to chapter 4 “Comprehensive Java Example”.

Note: The examples in the remainder of this chapter are not self-contained in order to improve readability. Required import statements are omitted to keep the examples more concise.

3.3 Importing Packages Generated by Fabasoft app.ducx

The classes generated by Fabasoft app.ducx allow you to access the Fabasoft Folio object model so you can retrieve and set property values, invoke other use cases, and access component objects such as object classes. These classes reside in a separate package. The name of this package must be specified along with the class folder when enabling Java support for your app.ducx project.

Fabasoft app.ducx generates a set of Java class files for all software components referenced in your Fabasoft app.ducx project. The generated Java class files are updated as you add or remove software component references.

For each software component referenced in your Fabasoft app.ducx project, a package is created and the reference of the software component is used as package name. In this package, a class with the same name as the package is provided which contains get methods for accessing all component objects that belong to the software component.

This package also contains four sub packages that allow you to access object classes, to invoke use cases, and to use enumeration types and compound types:

- The classes package contains classes for accessing object classes.
- The actions package contains classes for transporting the output parameters when invoking use cases.
- The enums package contains classes for working with enumeration types.
- The structs package contains classes for working with compound types.

In order to keep your source code concise, it is recommended to add import statements to the header of your source files to import the packages you need.

Example

```
// Import for accessing component objects belonging to APPDUCXSAMPLE@200.200,
// e.g. the object representing the software component
import APPDUCXSAMPLE.generated.APPDUCXSAMPLE_200_300.APPDUCXSAMPLE_200_300;
// Import for making available object class APPDUCXSAMPLE@200.200:Invoice
import APPDUCXSAMPLE.generated.APPDUCXSAMPLE_200_300.classes.Invoice;
// Import for making available class CreateInvoiceResult for retrieving
// the output parameters after invoking the use case APPDUCXSAMPLE@200.200:
// CreateInvoice
import APPDUCXSAMPLE.generated.APPDUCXSAMPLE_200_300.actions.CreateInvoiceResult;
// Import for making available the enumeration type APPDUCXSAMPLE@200.200:
// InvoiceState
import APPDUCXSAMPLE.generated.APPDUCXSAMPLE_200_300_enums.InvoiceState;
// Import for making available the compound type APPDUCXSAMPLE@200.200:
// OrderPosition
import APPDUCXSAMPLE.generated.APPDUCXSAMPLE_200_300.structs.OrderPosition;
```

In addition to the imports of packages generated by Fabasoft app.ducx, you also need to import parts of the `CooLib` package which contains some required classes. The library containing the `CooLib` package is automatically added to your project.

3.4 Data Types in Java

The following table contains the list of data types used for accessing the Fabasoft Folio object model in Java.

Fabasoft Folio Data Type	app.ducx Keyword	Java Class

COOSYSTEM@1.1:STRING	string	java.lang.String
COOSYSTEM@1.1:BOOLEAN	boolean	java.lang.Boolean
COOSYSTEM@1.1:INTEGER	integer, time, timespan	java.lang.Integer
COOSYSTEM@1.1:FLOAT	float	java.lang.Double
COOSYSTEM@1.1:DATETIME	date, datetime	java.util.Date
COOSYSTEM@1.1:Currency	currency	COOSYSTEM_1_1.structs.Currency
COOSYSTEM@1.1:CONTENT	content	CooLib.CooContent
COOSYSTEM@1.1:DICTIONARY	dictionary	CooLib.CooDictionary
COOSYSTEM@1.1:OBJECT	object	CooLib.CooObject

Typed lists are used for accessing lists of values, i.e. properties allowing multiple values. For instance, the `List<String>` interface can be used for accessing a string list property of data type `COOSYSTEM@1.1:STRINGLIST`.

The type definition component objects for simple data types are accessible over the `app.ducx` class. For example, `DUCX.getTypeBoolean()` returns the type definition component object `COOSYSTEM@1.1:BOOLEAN`.

3.5 Accessing Properties and Invoking Use Cases

The Fabasoft app.ducx wizard automatically generates methods for accessing properties and use cases belonging to the software components referenced in your Fabasoft app.ducx project. The generated methods allow you to retrieve and set property values and to invoke use cases.

Note: The object the use case is invoked on can be accessed using the `this` variable.

Please refer to chapter 3.6 "Working With Use Case Parameters" for detailed information on how to handle input and output parameters when implementing or invoking use cases.

Example

```
@DUCXImplementation("APPDUCXSAMPLE@200.200:InitializeOrder")
public void InitializeOrder(final Person customer) throws CooException {
  // Lock the order object
  this.COOSYSTEM_1_1_ObjectLock(true, true, null, null);
  // Initialize order date with current date and set order status to
  // OS PENDING
  this.APPDUCXSAMPLE_200_300_orderdate = new Date();
  this.APPDUCXSAMPLE_200_300_orderstate = OrderState.OS_PENDING;
  // Set express delivery for wholesalers
  if (customer.APPDUCXSAMPLE_200_300_customertype == CustomerType.
    CT_WHOLESALER) {
    this.APPDUCXSAMPLE 200 300 orderdelivery = DeliveryType.DT EXPRESS;
  }
  else {
    this.APPDUCXSAMPLE_200_300_orderdelivery = DeliveryType.DT_STANDARD;
  }
  // Add the order to the customer's list of orders
  this.COODESK 1 1 ShareObject(null, null, APPDUCXSAMPLE 200 300.
    getProperty_customerorders(), customer);
}
```

3.6 Working With Use Case Parameters

This chapter describes how to handle input and output parameters when implementing or invoking use cases.

3.6.1 Retrieving Input Parameters

When you add a method implementation to your Java project, a signature is generated automatically for the new Java method. This signature contains a parameter list that corresponds to the parameter list of your use case. The input parameters defined for the use case are passed in to your Java method as Java objects.

Example

app.ducx Use Case Language

```
usecase AddPositionToOrder(Product product, integer quantity) {
  variant Order {
    impl = java:APPDUCXSAMPLE.Order.AddPositionToOrder;
  }
}
```

Java Implementation

```
@DUCXImplementation("APPDUCXSAMPLE@200.200:AddPositionToOrder")
public void AddPositionToOrder(final Product product, final Long
  quantity) throws CooException {
  // Lock the order
  this.COOSYSTEM_1_1_ObjectLock(true, true, null, null);
  // Create a new order position
  OrderPosition position = OrderPosition.create();
  position.APPDUCXSAMPLE_200_300_product = product;
  position.APPDUCXSAMPLE_200_300_quantity = quantity;
  // Add the new order position to the order
  List<OrderPosition> positions = this.APPDUCXSAMPLE_200_300_orderpositions;
  positions.add(position);
  this.APPDUCXSAMPLE_200_300_orderpositions = positions;
}
```

3.6.2 Returning Output Parameters

Java methods do not support output parameters. Instead, a so-called result object must be used to return output parameters. The corresponding Java class for creating a result object is automatically created by a wizard when adding a new use case implementation in Java if the use case has one or more output parameters. The name of the Java class for retrieving the output parameters corresponds to the reference of the use case to be invoked, suffixed by the string “Result”.

When invoking another use case from within your Java method, you can follow the same approach for retrieving its output parameters: the invoked use case returns a result object containing a member variable for each output parameter.

Example

app.ducx Use Case Language

```
usecase GetCustomerOrders(out Order[] orders) {
  variant Person {
    impl = java:APPDUCXSAMPLE.Person.GetCustomerOrders;
  }
}
```

Java Implementation

```
@DUCXImplementation("APPDUCXSAMPLE@200.200:GetCustomerOrders")
public GetCustomerOrdersResult GetCustomerOrders() throws CooException {
  // Output parameters must be passed back using the result object
  GetCustomerOrdersResult result = new GetCustomerOrdersResult();
  // The sort order is determined by a list of property definitions
  ArrayList<Object> sortorder = new ArrayList<Object>();
  sortorder.add(APPDUCXSAMPLE_200_300.getProperty_orderdate());
  // Sort the customer's list of orders (the sorted list is returned in
  // the valuelist member variable of the SortResult object)
  SortResult sortresult = this.COOATTREDIT_1_1_Sort(
    this.APPDUCXSAMPLE_200_300_customerorders, true, sortorder, null);
  // Return the sorted list of orders
  result.orders = sortresult.valuelist;
  return result;
}
```

The result object provides the two methods `SetError(CooObject)` and `SetError(CooObject, String)`. When calling `SetError` the output parameters are saved and afterwards the error raises an exception.

3.7 Working With Objects

This chapter describes how to work with Folio objects in Java.

3.7.1 Creating New Objects in Fabasoft Folio

For creating a new instance of an object class, you can invoke the `create` method of the Java class representing the object class.

In the following example, a new instance of object class `APPDUCXSAMPLE@200.200:Invoice` is created and referenced in the `APPDUCXSAMPLE@200.200:orderinvoice` property of an order.

Example

```
@DUCXImplementation("APPDUCXSAMPLE@200.200:MarkOrderAsShipped")
public void MarkOrderAsShipped() throws CooException {
  // Lock the order object
  this.COOSYSTEM_1_1_ObjectLock(true, true, null, null);
  // Create a new instance of object class APPDUCXSAMPLE@200.200:Invoice
  Invoice invoice = Invoice.create();
  // Reference the new invoice object in the order object and set
  // the order's state to OS_SHIPPED
  this.APPDUCXSAMPLE_200_300_orderinvoice = invoice;
  this.APPDUCXSAMPLE_200_300_orderstate = OrderState.OS_SHIPPED;
}
```

3.7.2 Comparing Objects

You can use the `equals` method of an object to determine if it is equivalent to a given object instance in Fabasoft Folio.

Example

```
@DUCXImplementation("APPDUCXSAMPLE@200.200:ProcessPendingOrders")
public void AddOrderCategory(final Case case) throws CooException {
  ComponentDocumentCategory category = this.COOTC_1_1001_objcategory;
  List<Object> casecategories = case.COOTC_1_1001_allowedcategories;
  // Check whether APPDUCXSAMPLE@200.200:DocumentCategoryOrder was selected
  // for the order object
  if (category.equals(APPDUCXSAMPLE_200_300.getClass_DocumentCategoryOrder())) {
    if (!casecategories.contains(APPDUCXSAMPLE_200_300.
      getClass_DocumentCategoryOrder())) {
```

```
      ComponentDocumentCategory.from(APPDUCXSAMPLE_200_300.
        get_DocumentCategoryOrder()).COODESK_1_1_ShareObject(null, null,
        COOTC_1_1001.getProperty_allowedcategories(), case);
    }
    else {
      coort.SetError(APPDUCXSAMPLE_200_300.
      getErrorMessage_CategoryAlreadyAdded());
    }
  }
  else {
    coort.SetError(APPDUCXSAMPLE_200_300.
    getErrorMessage_UnexpectedOrderCategory());
  }
}
```

3.7.3 Important Methods of an Object

The following table summarizes the most important methods that can be invoked on instances of Fabasoft Folio objects.

Method	Description
`isValid()`	`isValid` determines whether the object is valid.
`IsClass(class)`	`IsClass` determines whether the object class it is invoked on is derived from or identical to `class`.
`HasClass(class)`	`HasClass` determines whether the object is an instance of or derived from `class`.
`GetClass()`	`GetClass` returns the object class of the object.
`GetName()`	`GetName` returns the *Name* (`COOSYSTEM@1.1:objname`) of the object. If the name of the object cannot be read (e.g. because the user does not have sufficient rights for reading the object name), a string explaining why the name of the object cannot be read is returned.
`GetAddress()`	`GetAddress` returns the *Address* (`COOSYSTEM@1.1:objaddress`) of the object.
`GetReference()`	`GetReference` returns the *Reference* (`COOSYSTEM@1.1:reference`) of a component object.
`GetIdentification()`	`GetIdentification` returns the full identification of the object, which is a combination of the *Address* (`COOSYSTEM@1.1:objaddress`) and a version timestamp.
`GetVersNr()`	`GetVersNr` returns the version number of the object. If no version exists, „0“ is returned.
`GetVersDate()`	`GetVersDate` returns the date and time of the version of the object. This method can only be used for versions of objects, otherwise an exception is thrown. Use `GetVersNr` to find out whether the object is a version object.

`LoadAllAttributes(tx)`	`LoadAllAttributes` loads the values of all properties into the Fabasoft Folio Kernel Cache.
`LoadSpecificAttributes(tx, properties)`	`LoadSpecificAttributes` loads the values of the properties specified in `properties` into the Fabasoft Folio Kernel Cache.
`HasAttribute(tx, property)`	`HasAttribute` checks whether the specified `property` is assigned to the object.
`HasAttributeValue(tx, property)`	`HasAttributeValue` checks whether the specified `property` is assigned to the object and whether it is assigned a value.
`CheckAccess(tx, accesstype)`	`CheckAccess` checks whether accessing the object with the specified access type is allowed.
`CheckGetAccess(tx, property)`	`CheckGetAccess` checks whether the specified `property` of the object may be read.
`CheckSetAccess(tx, property)`	`CheckGetAccess` checks whether the specified `property` of the object may be changed.
`IsGhost()`	`IsGhost` checks whether object is still alive.

3.8 Working With Enumeration Types, Compound Types, Contents, and Dictionaries

This chapter describes how to work with following types.

3.8.1 Working With Enumeration Types

The app.ducx wizard automatically generates Java classes for each enumeration type. These classes allow you to access all enumeration items of a particular enumeration type.

Example

```
@DUCXImplementation("APPDUCXSAMPLE@200.200:ArchiveOrder")
public void ArchiveOrder() throws CooException {
  if (this.APPDUCXSAMPLE_200_300_orderstate == OrderState.OS_COMPLETED) {
    // Lock the order object and set the order state to OS_ARCHIVED
    this.COOSYSTEM_1_1_ObjectLock(true, true, null, null);
    this.APPDUCXSAMPLE_200_300_orderstate = OrderState.OS_ARCHIVED;
    // Invoke an XML web service to archive the order
    archiveOrderXML();
  }
  else {
    // Raise an error if the order is not completed yet
    coort.SetError(APPDUCXSAMPLE_200_300.getErrorMessage_InvalidOrderState());
  }
}
```

3.8.2 Working With Compound Types

Data structures representing compound types are referred to as compound values. Compound values must be created before they can be used. The `create` method of the Java class representing the compound type must be invoked for creating a new compound value.

The following example demonstrates how to add a new item to a compound property.

Example

```
@DUCXImplementation("APPDUCXSAMPLE@200.200:AddPositionToOrder")
public void AddPositionToOrder(final Product product,
  final Long quantity) throws CooException {
  // Lock the order object
  this.COOSYSTEM_1_1_ObjectLock(true, true, null, null);
  // Retrieve the existing order positions
  List<OrderPosition> positions = this.APPDUCXSAMPLE_200_300_orderpositions;
  // Create a new entry and add it to the list of existing order positions
  OrderPosition position = OrderPosition.create();
  position.APPDUCXSAMPLE_200_300_product = product;
  position.APPDUCXSAMPLE_200_300_quantity = quantity;
  positions.add(position);
  // Write the modified list of order positions back to the order
  this.APPDUCXSAMPLE_200_300_orderpositions = positions;
}
```

3.8.3 Working With Contents

In Java, contents are represented by the `CooContent` class. The most important methods supported by the `CooContent` class can be found here:

https://help.developer.fabasoft.com/index.php?topic=doc/Fabasoft-appducx-for-Cloud-Developers/appducx-expression-language.htm#working-with-contents

In the following example, a string property of type `COOSYSTEM@1.1:STRINGLIST` is retrieved as a scalar string and stored in a `CooContent`. Please note that the `CooContent` has to be initialized first by invoking the `CreateContent` method of the Fabasoft Folio Runtime. Afterwards, the `CooContent` is stored in a compound value of type `COOSYSTEM@1.1:Content`.

Example

```
@DUCXImplementation("APPDUCXSAMPLE@200.200:SendProductDescription")
public void SendProductDescription() throws CooException {
  // Get the product description as a scalar string value
  // Note: this.APPDUCXSAMPLE_200_300_productdescription returns a List<String>
  String description = this.GetAttributeString(cootx, APPDUCXSAMPLE_200_300.
    getProperty_productdescription(), null, CooFlags.COODISP_ROWLIST);
  // Store the description string in a content
  CooContent descriptioncontent = coort.CreateContent();
  descriptioncontent.SetContent(cootx, CooFlags.COOGC_MULTIBYTEFILE,
    CooFlags.COOGC_UTF8, description);
  // Create a structure of compound type COOSYSTEM@1.1:Content and
  // store the content in this structure
  Content content = Content.create();
  content.COOSYSTEM_1_1_contcontent = descriptioncontent;
  content.COOSYSTEM_1_1_contextension = "txt";
  // Invoke an XML web service to transmit the content
  sendProductDescriptionXML(content);
}
```

3.8.4 Working With Dictionaries

In Java, dictionaries are represented by the `CooDictionary` class, which allows you to store a list of key-value pairs. The key must be a `String` value. The most important methods supported by the `CooDictionary` class can be found here:

https://help.developer.fabasoft.com/index.php?topic=doc/Fabasoft-appducx-for-Cloud-Developers/appducx-expression-language.htm#working-with-dictionaries

In many cases, dictionaries are used as local or global scope when evaluating app.ducx expressions.

Note: Before you can use a `CooDictionary`, it has to be initialized by invoking the `CreateDictionary` method of the Fabasoft Folio Runtime.

In the following example, an expression is evaluated by the implementation of use case `APPDUCXSAMPLE@200.200:ValidateInvoice`. The invoice object is provided in the local scope, and a dictionary is provided in the global scope.

Example

```
@DUCXImplementation("APPDUCXSAMPLE@200.200:ValidateInvoice")
public void ValidateInvoice() throws CooException {
  Invoice invoice = this;
  Date paymentdate = invoice.APPDUCXSAMPLE_200_300_invoicepaymentdate;
  InvoiceState invoicestate = invoice.APPDUCXSAMPLE_200_300_invoicestate;
  String expressiontext = getValidationExpression(invoice);
  // Create and populate global scope dictionary
  CooDictionary globalscope = coort.CreateDictionary();
  globalscope.SetEntryValue("paydate", paymentdate);
  globalscope.SetEntryValue("state", invoicestate);
  // Evaluate validation expression and get result
  CooValue[] result = CooExpression.Evaluate(cootx, expressiontext,
    CooValue.asArray(globalscope), CooValue.asArray(invoice));
  if (result[0].getBool().getValue()) {
    // Validation expression returned "true"
    processInvoice(invoice);
  }
  else {
    // Validation expression returned "false"
    coort.SetError(APPDUCXSAMPLE_200_300.getErrorMessage_ValidationError());
  }
}
```

3.9 Accessing the Fabasoft Folio Runtime

When implementing a method in Java, the static variable `coort` of the `DUCX` package can be used to access the Fabasoft Folio Runtime. The next table contains a list of the most important methods supported by the Fabasoft Folio Runtime.

Method	Description
`CreateContent()`	`CreateContent` initializes a new `CooContent` object. Please note that this is a memory structure, and not a persistent object in Fabasoft Folio.
`CreateDictionary()`	`CreateDictionary` initializes a new `CooDictionary` object. Please note that this is a memory structure, and not a persistent object in Fabasoft Folio.

`GetCurrentUser()`	`GetCurrentUser` returns the user object of the user currently logged in to Fabasoft Folio.
`GetCurrentUserEnvironment()`	`GetCurrentUserEnvironment` returns the active user environment object of the user currently logged in to Fabasoft Folio.
`GetCurrentUserRoot()`	`GetCurrentUserRoot` returns the desk object of the user currently logged in to Fabasoft Folio.
`GetCurrentUserLanguage()`	`GetCurrentUserLanguage` returns the language of the user currently logged in to Fabasoft Folio.
`GetCurrentUserRoleGroup()`	`GetCurrentUserRoleGroup` returns the group object of the current role of the user currently logged in to Fabasoft Folio.
`GetCurrentUserRolePosition()`	`GetCurrentUserRolePosition` returns the position object of the current role of the user currently logged in to Fabasoft Folio.
`GetObject(address)`	`GetObject` returns the object with the specified `address`.
`SearchObjects(tx, query)`	`SearchObjects` executes a Fabasoft Folio `query` and returns the matching objects.
`SetError(errormessage)`	`SetError` returns a `CooException` using the `errormessage` object.
`LoadAllAttributes(tx, objects)`	`LoadAllAttributes` loads the values of all properties of the specified `objects` into the Fabasoft Folio Kernel Cache.
`Log(string)`	`Log` writes the specified `string` to Fabasoft app.telemetry.
`Log(level, string)`	`Log` writes the specified `string` to Fabasoft app.telemetry using the specified `level`. `level` can be one of the following values: • `COOLL_LOG` • `COOLL_IPC` • `COOLL_NORMAL` • `COOLL_DETAIL` • `COOLL_DEBUG` • `COOLL_DEFAULT` (same as `COOLL_DETAIL`)
`Trace(string)`	`Trace` writes the specified `string` to the Fabasoft app.ducx Tracer (also if the software component is not in trace mode).

`Trace(string, value)`	`Trace` writes the specified `string` and a value of arbitrary data type to the Fabasoft app.ducx Tracer (also if the software component is not in trace mode).
`ReportEvent(source, string, type, category)`	`ReportEvent` creates a new entry in the event log.

Example

```
@DUCXImplementation("APPDUCXSAMPLE@200.200:ProcessPendingOrders")
public void ProcessPendingOrders() throws CooException {
  // Get the pending orders of the current user
  User user = (User) coort.GetCurrentUser();
  List<Order> pendinglist = user.APPDUCXSAMPLE_200_300_pendingorders;
  // Trace the number of pending orders
  coort.Trace("Found " + pendinglist.size() + " pending orders");
  // Load all properties of the pending order objects
  Order[] orders = pendingList.toArray(new Order[pendinglist.size()]);
  coort.LoadAllAttributes(cootx, orders);
  processPendingOrders(orders);
}
```

3.10 Accessing the Transaction Context

When implementing a method in Java, the static variable `cootx` of the `DUCX` package can be used to access the current transaction context. `cootx` returns an instance of Java class `CooTransaction`. The most important methods of the `CooTransaction` class are listed her:

https://help.developer.fabasoft.com/index.php?topic=doc/Fabasoft-appducx-for-Cloud-Developers/appducx-expression-language.htm#accessing-the-transaction-context

Example

```
@DUCXImplementation("COOSYSTEM@1.1:ObjectPrepareCommit")
public void InvoicePrepareCommit(final Boolean internalchange)
  throws CooException {
  // Check if the invoice has been created in the current transaction
  // and if property APPDUCXSAMPLE@200.200:invoicestate has been changed
  if (!cootx.IsCreated(this) && cootx.IsAttributeChanged(this,
    APPDUCXSAMPLE_200_300.getProperty_invoicestate())) {
    // Rebuild the object's name based on the name build configuration
    this.FSCCONFIG_1_1001_ObjectPrepareCommitNameBuild();
  }
}
```

3.10.1 Creating a New Transaction

In some scenarios it is necessary to carry out operations in a separate transaction. Any changes that have been made in a new transaction can be committed or rolled back separately from the main transaction.

Example

```
@DUCXImplementation("APPDUCXSAMPLE@200.200:CreateInvoice")
public void CreateInvoice() throws CooException {
  // Create a new transaction
  CooTransaction backuptx = DUCX.cootx;
  CooTransaction localtx = new CooTransaction();
```

```
  try {
    // Perform the following operations in context of the new transaction
    coort.SetThreadTransaction(localtx);
    Invoice invoice = Invoice.create();
    this.APPDUCXSAMPLE_200_300_InitializeInvoice(invoice);
    // Commit the changes
    localtx.Commit();
  }
  catch (Exception ex) {
    // In case of an error, roll back the changes
    localtx.Abort();
  }
  finally {
    // Restore original transaction context
    coort.SetThreadTransaction(backuptx);
    // Clear the variables holding the transactions
    backuptx = null;
    localtx = null;
  }
}
```

3.10.2 Working With Transaction Variables

A transaction variable is a temporary variable identified by a software component, a transaction variable reference, and a unique identification number. Transaction variables must be declared using the app.ducx use case language before they can be used.

For accessing a transaction variable, you need a combination of either software component reference and transaction variable reference or software component reference and transaction variable identification number.

Transaction variables are stored and transported in the so-called transaction context, i.e. they are cleared when a transaction is committed or aborted.

The purpose of transaction variables is to transfer status information between different use cases invoked within the same transaction.

In some cases, necessary context information cannot be provided in form of parameters when a use case is invoked. For example, the Fabasoft Folio workflow engine makes available a set of four transaction variables (see next table) when a work item of an activity is executed by a user.

Identifier	Description
`WFVAR_THIS`	Transaction variable `WFVAR_THIS` of `COOWF@1.1` holds the business object attached to the process.
`WFVAR_PROCESS`	Transaction variable `WFVAR_PROCESS` of `COOWF@1.1` holds the process instance.
`WFVAR_ACTIVITY`	Transaction variable `WFVAR_ACTIVITY` of `COOWF@1.1` holds the current activity instance.
`WFVAR_WORKITEM`	Transaction variable `WFVAR_WORKITEM` of `COOWF@1.1` holds the zero-based index of the work item executed by the user.

To access transaction variables, you need to use the access methods exposed by the `cootx` interface object of the `DUCX` package (see chapter 3.10 “Accessing the Transaction Context”).

Example

```
@DUCXImplementation("APPDUCXSAMPLE@200.200:ReleaseOrder")
public void ReleaseOrder() throws CooException {
  // Get the order object from transaction variable 1 of COOWF@1.1
  Order order = Order.from(cootx.GetVariableValue(COOWF_1_1.
    getSoftwareComponent(), 1).getObject());
  // Determine if batch mode is enabled by checking transaction
  // variable TV_BATCHMODE of COOSYSTEM@1.1
  Boolean batchmode = COOSYSTEM_1_1.getTransactionVariables_TV().
    COOSYSTEM_1_1_TV_BATCHMODE;
  if (order != null && !batchmode) {
    releaseOrder(order);
  }
}
```

For your software component, you may also declare your set of transaction variables using the app.ducx use case language and use them in your Java code as illustrated by the following example.

Example

```
@DUCXImplementation("APPDUCXSAMPLE@200.200:ProcessOrder")
public void ProcessOrder() throws CooException {
  if (processOrder(this)) {
    Invoice invoice = this.APPDUCXSAMPLE_200_300_orderinvoice;
    Boolean printinvoice = invoice.APPDUCXSAMPLE_200_300_invoicestate ==
      InvoiceState.IS_PENDING;
    // Set transaction variables TV_PRINTINVOICE and TV_INVOICE of
    // APPDUCXSAMPLE@200.200
    COOSYSTEM_1_1.getTransactionVariables_TV().
      APPDUCXSAMPLE_200_300_TV_PRINTINVOICE = printinvoice;
    COOSYSTEM_1_1.getTransactionVariables_TV().
      APPDUCXSAMPLE_200_300_TV_INVOICE = invoice;
  }
}
@DUCXImplementation("APPDUCXSAMPLE@200.200:ShipOrder")
public void ShipOrder() throws CooException {
  // Retrieve and evaluate transaction variable TV_PRINTINVOICE of
  // APPDUCXSAMPLE@200.200
  Boolean printinvoice = COOSYSTEM_1_1.getTransactionVariables_TV().
    APPDUCXSAMPLE_200_300_TV_PRINTINVOICE
  if (printinvoice) {
    Invoice invoice = COOSYSTEM_1_1.getTransactionVariables_TV().
      APPDUCXSAMPLE_200_300_TV_INVOICE;
    invoice.APPDUCXSAMPLE_200_300_SendInvoice();
  }
  shipOrder(this);
}
@DUCXImplementation("APPDUCXSAMPLE@200.200:SendInvoice")
public void SendInvoice() throws CooException {
  Invoice invoice = this;
  Order order = invoice.APPDUCXSAMPLE_200_300_invoiceorder;
  ContactPerson customer = order.APPDUCXSAMPLE_200_300_ordercustomer;
  CooContent mailbodycontent = coort.CreateContent();
  // Create and populate global scope dictionary
  CooDictionary globalscope = coort.CreateDictionary();
  globalscope.SetEntryValue("invoice", invoice);
  globalscope.SetEntryValue("order", order);
  globalscope.SetEntryValue("customer", customer);
  // Make available the values stored in the global scope dictionary
  // when the invoice report is evaluated by initializing transaction
  // variable 1 of COOAR@1.1
  cootx.SetVariableValue(COOAR 1 1.getSoftwareComponent(), 1,
    DUCX.getTypeDictionary(), globalscope);
  // Evaluate the invoice report template to get the HTML mail body
  CreateReportResult reportresult = invoice.COOAR_1_1_CreateReport(
    APPDUCXSAMPLE_200_300.getClass_InvoiceTemplateReport(), false, null, null,
    mailbodycontent);
  mailbodycontent = reportresult.resultcontent;
  if (mailbodycontent != null) {
    Addressee mainaddress = customer.FSCFOLIO_1_1001_mainaddress;
```

```
    String recipient = mainaddress.FSCFOLIO_1_1001_emailaddress;
    String sender = "orderprocessing@APPDUCXSAMPLE.fabasoft.com";
    String subject = invoice.GetName();
    ArrayList<String> recipients = new ArrayList<String>();
    recipients.add(recipient);
    ArrayList<CooContent> mailbody = new ArrayList<CooContent>();
    mailbody.add(mailbodycontent);
    %%TRACE("Mail Recipient:", recipient);
    %%TRACE("Mail Subject:", subject);
    %%TRACE("Mail Body:", mailbodycontent.GetContent(cootx,
      CooFlags.COOGC_MULTIBYTEFILE, CooFlags.COOGC_UTF8));
    // Send the invoice mail to the customer
    invoice.FSCSMTP_1_1001_SendHTML(sender, recipients, subject, mailbody,
      null, null, null, null, null, null, null);
  }
}
```

3.11 Tracing in Java

In addition to using the tracing functionality built into Fabasoft app.ducx, you may also write custom trace messages to Fabasoft app.ducx Tracer.

You can use the `Trace` method of the `coort` interface object for invoking the trace functionality of the Fabasoft Folio Runtime. However, this method only supports tracing simple strings.

The `trace` method of the `DUCX` class is more powerful, and allows you to trace all kinds of Fabasoft Folio data types.

Moreover, the `DUCX` class also exposes the `traceEnter` and `traceLeave` methods that can be used for logging in the trace output when your Java method has been entered and left.

Example

```
@DUCXImplementation("APPDUCXSAMPLE@200.200:ProcessPendingOrders")
public void ProcessPendingOrders() throws CooException {
  DUCX.traceEnter();
  // Get the pending orders of the current user
  User user = (User) coort.GetCurrentUser();
  List<Order> pendinglist = user.APPDUCXSAMPLE_200_300_pendingorders;
  // Trace the list of pending orders
  DUCX.trace("List of Orders:", pendinglist);
  // Load all properties of the pending order objects
  Order[] orders = pendingList.toArray(new Order[pendinglist.size()]);
  coort.LoadAllAttributes(cootx, orders);
  processPendingOrders(orders);
  DUCX.traceLeave();
}
```

3.12 Debugging with Java

Before you can debug an app.ducx project in Eclipse, make sure that the `COOJAVA_JVMOPTIONS` environment variable is set to:

`-agentlib:jdwp=server=y,transport=dt_socket,address=8000,suspend=n.`

You may also use a port other than "8000" by setting the `address` parameter to the desired value. Keep in mind, that only one JVM can be bound to the same socket, otherwise the JVM will not be initialized.

Note: You can also use the `COOJAVA_JVMOPTIONS` environment variable to set other parameters for the Java virtual machine, e.g. the `-Xms` and `-Xmx` parameters for defining the size of the heap space.

Adjusting the size of the heap space might be necessary if you run into the following Java runtime error:

```
java.lang.OutOfMemoryError: Java heap space
```

The `COOJAVA_JVMOPTIONS` environment variable can also be individually defined for each web service instance in order to avoid conflicts if multiple web services are hosted on a single machine.

To define the `COOJAVA_JVMOPTIONS` environment variable for a particular web service instance, add the following entries to the registry:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Fabasoft\FscWeb\Modules\FSC]
@="C:/Program Files/Fabasoft/Components/Web/1_FSC/ASP/content/bin/fscvext.dll"

[HKEY_LOCAL_MACHINE\SOFTWARE\Fabasoft\FscWeb\Modules\FSC\COOJAVA_JVMOPTIONS]
@="-agentlib:jdwp=server=y,transport=dt_socket,address=8000,suspend=n"
```

Note: If you are manually adding the described entries to the registry using the Registry Editor, you have to create the keys that do not exist yet. The "@" character in the example refers to the "(Default)" registry value.

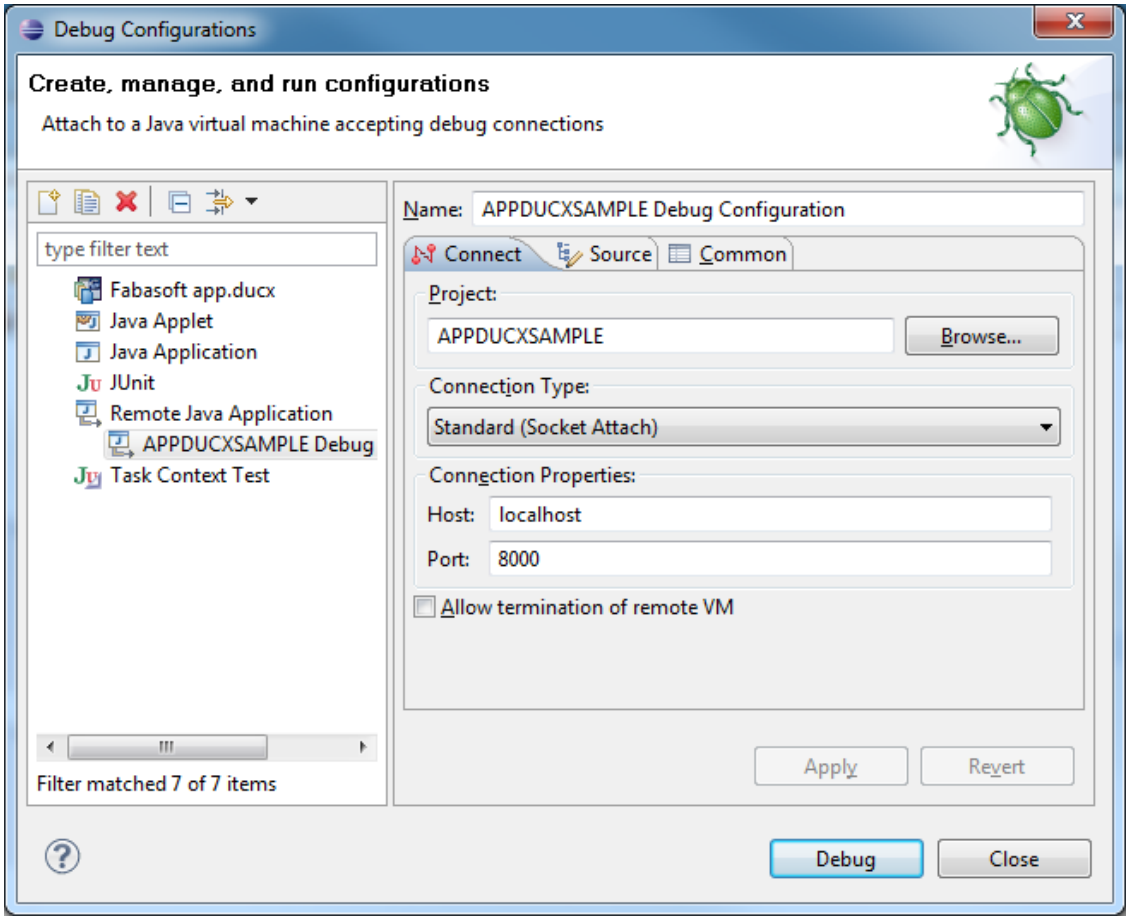
Adapt the path to the virtual directory associated with your web service as appropriate. The correct path can be obtained by carrying out the following steps:

1. Start the Fabasoft app.ducx Tracer.
2. Restart the Fabasoft Folio Web Service.
3. Point your web browser to the Fabasoft Folio Web Service.
4. Search the trace output for the modulepath variable. Copy the value into the default registry value of the registry key `[HKEY_LOCAL_MACHINE\SOFTWARE\Fabasoft\FscWeb\Modules\FSC]`.

After setting the environment variable, the Fabasoft Folio Web Service must be restarted. If you are using Microsoft Internet Information Services to host the Fabasoft Folio Web Service, do not issue an `iisreset` command but restart the "World Wide Web Publishing Service" in the "Services" snap-in instead.

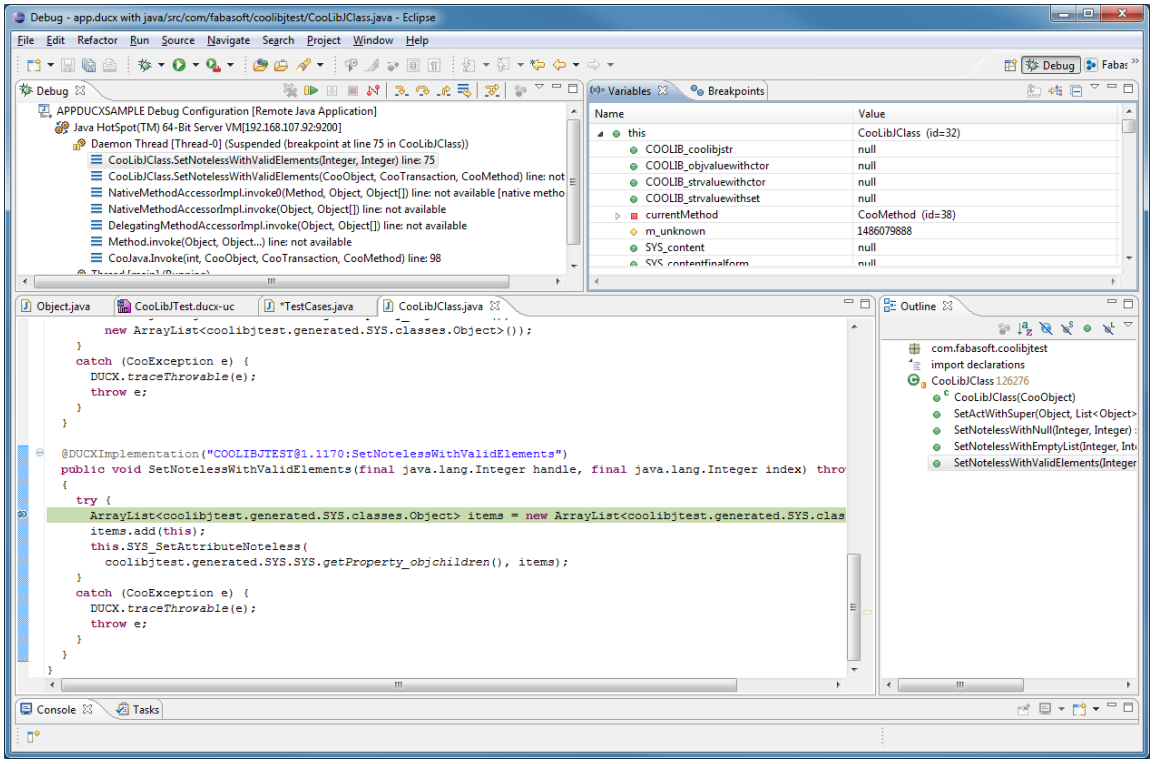
In order to verify that the remote debugging listening port has been installed correctly issue `netstat -a | more` and search the output for the specified listening port (e.g. 8000).

Furthermore, you have to create a new debug launch configuration in Eclipse. To do this, select the *Debug* from the *Run* menu. This will bring up the dialog box depicted in the next figure. Select "Remote Java Application" and click *New Launch Configuration*. Enter a *Name* for the debug launch configuration, select the *Project*, enter the name of the web server hosting the Fabasoft Folio Web Service in the *Host* field, click *Apply* to save your settings, and *Debug* to attach the remote debugger to the remote Java virtual machine.



Once a debug launch configuration has been created, you can start debugging your Java use case implementations.

You can set breakpoints in your Java code by clicking *Toggle Breakpoint* in the context menu in the corresponding line of code. When you click *Debug* from the *Run* menu, select your debug launch configuration and click *Debug* to run your Fabasoft app.ducx project. Execution will stop when a breakpoint is reached to take you to the Eclipse Debugger (see next figure).



3.13 Support of Old-Style Java Implementations

If a Java archive (`.jar`) file with an old-style java implementation is used, the Java support must not be activated in the app.ducx project.

For all defined Java implementation neither the package string is verified nor the prefix is generated For further information on how to attach a trigger action to a trigger event in a property definition, please refer to chapter 3.1 “Defining a Use Case to Be Implemented in Java”.

If the project is loaded into a Fabasoft domain, the Java archive (`.jar`) file has to be copied manually to the web server.

3.14 Working With Type Definition of a Customization Point

The type of a customization point is accessible with the class `TypeCustomizationPointDef`.

Example

```
public List<ActionParameterList> GetNameBuildInParameters() throws CooException {
  //Get the type of the customization point
  TypeCustomizationPointDef cptdef = FSCCONFIG_1_1001.getCPT_NameBuild();
  //Return customization point parameters
  List<ActionParameterList> parameters = tcpt.COOSYSTEM_1_1_typecpparameters;
  ...
  return inparameters;
}
```

The customizations in a customization point can only be accessed through a use case implementation with Fabasoft app.ducx Expression.

4 Comprehensive Java Example

The following example demonstrates the implementation of a use case for importing customer orders from an XML document. The XML document is stored in a structure of compound type `COOSYSTEM@1.1:Content` that is passed to the use case as an input parameter. An order object is created in Fabasoft Folio for each of the order records found in the XML document. Afterwards, the orders' properties are populated with data from the XML records. Finally, the new order objects are added to the customer's list of orders.

Example

XML File

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<orders>
  <order>
    <orderid>1000251</orderid>
    <orderdate>22.01.2010</orderdate>
    <orderpositions>
      <orderposition>
        <productid>J82-K32-629</productid>
        <quantity>10</quantity>
      </orderposition>
      <orderposition>
        <productid>A72-G53-882</productid>
        <quantity>2</quantity>
      </orderposition>
    </orderpositions>
  </order>
  <order>
    <orderid>1000252</orderid>
    <orderdate>23.01.2010</orderdate>
    <orderpositions>
      <orderposition>
        <productid>B03-X53-341</productid>
        <quantity>1</quantity>
      </orderposition>
    </orderpositions>
  </order>
</orders>
```

app.ducx Use Case Language

```
usecase ImportOrders(Content xmlcontent) {
  variant Person {
    impl = java:APPDUCXSAMPLE.Person.ImportOrders;
  }
}
```

Java Source Code

```
package APPDUCXSAMPLE;

import java.io.File;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

import static APPDUCXSAMPLE.generated.DUCX.coort;

import APPDUCXSAMPLE.generated.DUCX;
import APPDUCXSAMPLE.generated.DUCXImplementation;

import APPDUCXSAMPLE.generated.COOSYSTEM_1_1.structs.Content;

import APPDUCXSAMPLE.generated.APPDUCXSAMPLE_200_300.APPDUCXSAMPLE_200_300;
import APPDUCXSAMPLE.generated.APPDUCXSAMPLE_200_300.classes.Order;
import APPDUCXSAMPLE.generated.APPDUCXSAMPLE_200_300.classes.Product;
import APPDUCXSAMPLE.generated.APPDUCXSAMPLE_200_300.structs.OrderPosition;
```

```
import CooLib.CooObject;
import CooLib.CooContent;

public class Person extends APPDUCXSAMPLE.generated.FSCFOLIO_1_1001.
  classes.Person {
    public Person(CooObject obj) {
      super(obj);
  }
  private String getXMLValue(Node parentnode, String tagname) {
    Element elmnt = (Element) parentnode;
    NodeList nodelst = elmnt.getElementsByTagName(tagname);
    Element childelmnt = (Element) nodelst.item(0);
    return childelmnt.getChildNodes().item(0).getNodeValue();
  }
  @DUCXImplementation("APPDUCXSAMPLE@200.200:ImportOrders")
  public void ImportOrders(final Content xmlcontent) throws Exception {
    try {
      // Build XML DOM from XML content
      CooContent content = xmlcontent.COOSYSTEM_1_1_contcontent;
      File file = new File(content.GetFile("", false));
      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
      DocumentBuilder db = dbf.newDocumentBuilder();
      Document doc = db.parse(file);
      doc.getDocumentElement().normalize();

      NodeList ordernodelst = doc.getElementsByTagName("order");

      for (int i = 0; i < ordernodelst.getLength(); i++) {
        // Initialize the list of order positions
        ArrayList<OrderPosition> poslst = new ArrayList<OrderPosition>();

        Node ordernode = ordernodelst.item(i);

        if (ordernode.getNodeType() == Node.ELEMENT_NODE) {
          // Retrieve order date from XML
          String orderDatestr = getXMLValue(ordernode, "orderdate");
          SimpleDateFormat dateformat = new SimpleDateFormat("dd.MM.yyyy");
          Date orderdate = dateformat.parse(orderDatestr);

          // Retrieve list of order positions from XML
          Element orderelmnt = (Element) ordernode;
          Element orderpositionselmnt = (Element) orderelmnt.
            getElementsByTagName("orderpositions").item(0);
          NodeList orderpositionnodelst = orderpositionselmnt.
            getElementsByTagName("orderposition");

          // Iterate through order positions (an order may consist of one
          // or more order positions)
          for (int j = 0; j < orderpositionnodelst.getLength(); j++) {
            Node orderposnode = orderpositionnodelst.item(j);

            if (orderposnode.getNodeType() == Node.ELEMENT_NODE) {
              // Retrieve product ID for current order position
              String productid = getXMLValue(orderposnode, "productid");

              // Retrieve quantity
              Long orderquantity = new Long(getXMLValue(orderposnode,
                "quantity"));

              // Find existing product object matching the product ID
              // in Fabasoft Folio
              CooObject[] orderproducts = coort.SearchObjects(DUCX.
                cootx, "LIMIT 1 SELECT objname FROM APPDUCXSAMPLE@200.200:
                Product WHERE .APPDUCXSAMPLE@200.200:productid = \"" +
                productid + "\"");

              if (orderproducts.length > 0) {
                Product orderproduct = Product.from(orderproducts[0]);

                if (orderproduct.isValid() && orderquantity != null) {
                  // Create new order position item and add it to the
                  // list of order positions
                  OrderPosition item = OrderPosition.create();
                  item.APPDUCXSAMPLE_200_300_product = orderproduct;
                  item.APPDUCXSAMPLE_200_300_quantity = orderquantity;
                  poslst.add(item);
                }
                else {
                  coort.SetError(APPDUCXSAMPLE_200_300.
                  getErrorMessage_InvalidOrder());
                }
              }
              else {
```

```
                coort.SetError(APPDUCXSAMPLE_200_300.
                getErrorMessage_ProductNotFound());
              }
            }
          }
          // Create new order object in Fabasoft Folio
          Order order = Order.create();
          // Set order date and positions to order object
          order.APPDUCXSAMPLE_200_300_orderdate = orderdate;
          order.APPDUCXSAMPLE_200_300_orderpositions = poslst;
          // Add the new order to the customer's list of orders
          order.COODESK_1_1_ShareObject(null, null, APPDUCXSAMPLE_200_300.
            getProperty_customerorders(), this);
        }
      }
    } catch (Exception e) {
      coort.Trace("Exception occurred: " + e.getMessage());
      throw e;
    }
  }
}
```