



Artix™ Data Services

Getting Started

Version 3.7, May 2008

IONA Technologies PLC and/or its subsidiaries may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this publication. Except as expressly provided in any written license agreement from IONA Technologies PLC, the furnishing of this publication does not give you any license to these patents, trademarks, copyrights, or other intellectual property. Any rights not expressly granted herein are reserved.

IONA, IONA Technologies, the IONA logos, Orbix, Artix, Making Software Work Together, Adaptive Runtime Technology, Orbacus, IONA University, and IONA XMLBus are trademarks or registered trademarks of IONA Technologies PLC and/or its subsidiaries.

Java and J2EE are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. CORBA is a trademark or registered trademark of the Object Management Group, Inc. in the United States and other countries. All other trademarks that appear herein are the property of their respective owners.

IONA Technologies PLC makes no warranty of any kind to this material including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. IONA Technologies PLC shall not be liable for errors contained herein, or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

COPYRIGHT NOTICE

No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, photocopying, recording or otherwise, without prior written consent of IONA Technologies PLC. No third party intellectual property right liability is assumed with respect to the use of the information contained herein. IONA Technologies PLC assumes no responsibility for errors or omissions contained in this book. This publication and features described herein are subject to change without notice.

Copyright © 2008 IONA Technologies PLC. All rights reserved.

All products or services mentioned in this manual are covered by the trademarks, service marks, or product names as designated by the companies who market those products.

Updated: May 21, 2008

Contents

Preface	5
Chapter 1 Creating Projects	7
Starting Artix Data Services Designer	8
Creating a Project with the Project Wizard	9
Creating a Project Manually	12
Chapter 2 Creating Data Models	15
Downloading Sample Getting Started Data	17
Creating a Data Model Manually	18
Creating the Accounts Data Model Manually	19
Creating the Customers Data Model Manually	31
Creating a Data Model from a Text File	40
Creating the Transactions Data Model from Transactions.txt	41
Creating the Customers Data Model from Customers.txt	45
Creating a Data Model from an XML Schema	50
Creating a Data Model from Other Sources	52
Creating a Data Model from a Set of XML Documents	53
Creating a Data Model from a Database	56
Adding Validation Rules	59
Adding Validation Rules for Accounts Data Model	60
Adding Validation Rules for Transactions Data Model	63
Chapter 3 Creating Transformations	67
Creating a Simple Transformation	68
Starting to Create a Transformation	69
Adding an Input Model	70
Adding an Output Model	71
Creating a Local Transformation	72
Testing the Local Transformation in Your Main Transformation	75
Creating a Filter	77
Testing the Filter in Your Main Transformation	79

Making Your Transformation More Complex	81
Before You Continue	82
Adding More Input Models to Your Main Transformation	84
Adding Local Transformations	86
Adding Functions	90
Adding Nested Local Transformations	95
Adding Hash Tables	101
Adding Filters	104
Adding Java Methods	111
Adding Introspect Functions	114
Chapter 4 Overview of ANT Tasks	117

Preface

What This Book Covers

This book is intended to help you get started quickly with Artix Data Services. It provides demonstration walkthroughs of various tasks that you can perform in Artix Data Services Designer.

Who Should Read This Book

This book is intended for Artix Data Services users who wish to quickly familiarise with using the product.

Prerequisites

See the Artix Data Services *Installation Guide* for a full list of supported platforms and other prerequisites relating to the use of Artix Data Services.

How This Book Is Structured

This book contains the following chapters:

- [Chapter 1, “Creating Projects”](#) describes how to create projects in Artix Data Services Designer.
- [Chapter 2, “Creating Data Models”](#) describes how to create data models in Artix Data Services Designer from various different sources. It also describes how to validate data models to ensure that they can successfully parse valid data.
- [Chapter 3, “Creating Transformations”](#) describes how to create transformations in Artix Data Services Designer that allow you to map various elements in one or more input data models to various elements in an output data model. It also describes how to run your transformations to ensure that they are valid.

The Artix Data Services Documentation Library

For information on the organization of the Artix Data Services documentation library, and the document conventions used, see the Artix Data Services *Documentation Library Overview* at http://www.iona.com/support/docs/artix/data_services/3.7/index.xml

Creating Projects

In Artix Data Services, projects are used to store the data models, transformations and other working files for the various tasks you might wish to perform. Creating a project is therefore a prerequisite before you can perform any other task in Artix Data Services. There are different ways of creating new projects, depending on whether you choose to use the Project Wizard or create a project manually. This chapter demonstrates both methods of creating a project in Artix Data Services.

In this chapter

This chapter discusses the following topics:

Starting Artix Data Services Designer	page 8
Creating a Project with the Project Wizard	page 9
Creating a Project Manually	page 12

Starting Artix Data Services Designer

Overview


Because you can install or deploy Artix Data Services in different ways and on different platforms, there are various ways you can subsequently start Artix Data Services Designer.

Note: If you have not yet installed, see the Artix Data Services *Installation Guide* for more details of how to install the product.

Installed via IONA Downloads page

If you have installed Artix Data Services via the IONA Downloads page, do any of the following to start the Artix Data Services Designer:

Windows:

- Select **Programs | IONA | Artix Data Services | Artix DS Designer** from the **Start** menu.
- Click the  icon on your Windows desktop.
- Use Windows Explorer to navigate to your Artix Data Services installation directory and double click `artix-ds-designer.exe`.

UNIX:

- Run the `artix-ds-designer` command from your Artix Data Services installation directory.
-

Installed via Java Web Start

If you have deployed Artix Data Services using Java Web Start, the Artix Data Services Designer is automatically opened when you first deploy the product. To open the Designer on subsequent occasions:

Windows:

Select **Start > Run** and enter `javaws -viewer`.

UNIX:

Run the `javaws -viewer` command from any shell.

Creating a Project with the Project Wizard

Overview

The project wizard provides a step-by-step guide to creating projects. This demonstration shows how to use the project wizard to create a project called `MyProject.iop`. This project file will then be used as the basis for working through the rest of the Getting Started material.

Note: This demonstration caters for all properties associated with a wizard. Some of these properties are probably not very useful at the beginning stages of using Artix Data Services Designer, but it will become apparent later why the properties were created.



Demonstration steps

The steps are:

1. Start Artix Data Services Designer if you have not already.
Artix Data Services Designer opens with the Welcome window displayed. If this is the first time you have opened Artix Data Services Designer, the **Tip Of The Day** dialog is also displayed.
2. If it is displayed, uncheck the **Show Tips on startup** check box and click **Close** to cancel the **Tip Of The Day** dialog.
3. Click the **Project Wizard** link in the Welcome window. This opens the **Setup** panel of the Project Wizard.
4. For the purposes of this demonstration, type "MyProject" in the **File name** field. (Notice how the filename in the **Location** field is automatically updated to "MyProject.iop" as you type.)
5. Click the browse button beside the **Location** field to open the file browser.
6. For the purposes of this demonstration, navigate to `My IONA Projects/Getting Started`, and click **Open**.
The selected path is then automatically displayed in the **Location** field.

7. Click **Next** to open the **Paths** panel of the Project Wizard. This panel lets you specify one or more directory location paths in the file system where your working files, such as your data models, will be stored. These are the directories that Artix Data Services Designer will "know" about when you work within the project.

The default path on Windows is `C:\Documents and Settings\username\My Documents\My IONA Projects`. The default path on UNIX is `/userhome/My IONA Projects`. The alias represents the name by which the full path will be represented within Artix Data Services Designer.

8. You may add other paths if you wish by clicking the  icon. For the purposes of this demonstration, click the  icon to open the **Select** dialog, navigate to `My IONA Projects/Getting Started`, and click **Select**. The selected path is automatically added to the **Path** column, and the corresponding value in the **Alias** column is displayed as `Getting Started`.
9. The Project Wizard includes an **Advanced** button that allows you to view or hide two optional panels within the wizard. For the purposes of this demonstration, click the **Advanced** button to view the two optional panels. This means that the **Next** button on the **Paths** panel should now be enabled.
10. Click **Next** to open the **Project Properties** panel of the Project Wizard. These properties allow you to determine how your project file is to be stored and accessed.

For the purposes of this demonstration, accept all the default values for now. Try clicking on each of the fields listed and notice how context-sensitive descriptions of each field are displayed at the bottom of the panel.

11. Click **Next** to open the **Profile Settings** panel of the Project Wizard. These settings allow you to determine characteristics and behavior of deployed Java code in terms of code style, versioning and the location into which generated code is deployed.
For the purposes of this demonstration, accept all the default values for now. Again, try clicking on the various fields listed and notice how

context-sensitive descriptions of each field are displayed at the bottom of the panel.

12. Click **Finish**. If you are prompted to open the project in a new frame, click **Yes**. (This prompt does not appear if this is the first project you have created.)

The new project is then automatically displayed in the Project window along with the various paths you added for the project.

Creating a Project Manually

Overview

You can create a project manually without using the Project Wizard. This demonstration shows how to manually create a project called `MyProject.iop`. This project file will then be used as the basis for working through the rest of the Getting Started material.

Note: If you have already created `MyProject.iop` using the project wizard in the previous section, but you wish to work through this section anyway, simply choose another name for the project you create here. You could call it `MyProject2.iop` for example.

Note: This demonstration only pays attention to obvious project properties such as directories.

Demonstration steps

The steps are:




1. Start Artix Data Services Designer if you have not already. Artix Data Services Designer opens with the Welcome window displayed. If this is the first time you have opened Artix Data Services Designer, the **Tip Of The Day** dialog is also displayed.
2. If it is displayed, uncheck the **Show Tips on startup** check box and click **Close** to cancel the **Tip Of The Day** dialog.
3. Select **File > New Project**. This opens the **Create** wizard.
4. For the purposes of this demonstration, navigate to `My IONA Projects/Getting Started/Samples and Videos`, type "MyProject" in the **File name** field and click **Create**.

Note: Remember, if you have already created `MyProject.iop` using the project wizard in the previous section, type a different name in the **File name** field here. Type "MyProject2" for example.

If you are prompted to open the project in a new frame, click **Yes**. (This prompt does not appear if this is the first project you have created.)

This opens the **Project Properties** dialog for your project with the **Paths** icon automatically selected. This panel lets you specify one or more directory location paths in the file system where your working files, such as your data models, will be stored. These are the directories that Artix Data Services will "know" about when you work within the project.

The default path on Windows is `C:\Documents and Settings\username\My Documents\My IONA Projects`. The default path on UNIX is `/userhome/My IONA Projects`. The alias represents the name by which the full path will be represented within Artix Data Services Designer.

5. You may add other paths if you wish by clicking the  icon. For the purposes of this demonstration, click the  icon to open the **Select** dialog, navigate to `My IONA Projects/Getting Started/Standards Libraries`, and click **Select**. The selected path is automatically added to the **Path** column, and the corresponding value in the **Alias** column is displayed as `Standards Libraries`.
6. Click on the  icon to add another path. This opens the **Select** dialog.
7. For the purposes of this demonstration, navigate to `My IONA Projects/Examples`, and click **Select**.
The selected path is automatically added to the **Path** column, and the corresponding value in the **Alias** column is displayed as `Examples`.
8. Click the **Properties** icon to view the various project properties. These properties allow you to determine how your project file is to be stored and accessed.

For the purposes of this demonstration, accept all the default values for now. Try clicking on each of the fields listed and notice how context-sensitive descriptions of each field are displayed at the bottom of the panel.

9. Click the **Profiles** icon and then click the **Open** button to view the various profile settings. These settings allow you to determine characteristics and behavior of deployed Java code in terms of code style, versioning and the location into which generated code is deployed.

For the purposes of this demonstration, accept all the default values for now. Again, try clicking on the various fields listed and notice how context-sensitive descriptions of each field are displayed at the bottom of the panel.

10. Click **OK**.

The new project is then automatically displayed in the Project window along with the various paths you added for the project.

Creating Data Models

In Artix Data Services, data models are organised within projects and can consist of various different types of data components, including simple and complex types. They are used to represent some real-world data in which you are interested. From data models you can generate Java code that can then be used to parse, validate and transform conformant data. Data models generally consist of about 10 or more different types of data components but, for the purposes of illustration, this chapter focuses specifically on four components—simple data types, complex types, elements and enumerations. This chapter describes how to create data models in various different ways and from various different data sources.

Note: Before you continue, ensure that you follow the instructions in [“Downloading Sample Getting Started Data”](#) on page 17.

In this chapter

This chapter discusses the following topics:

Downloading Sample Getting Started Data	page 17
Creating a Data Model Manually	page 18

Creating a Data Model from a Text File	page 40
Creating a Data Model from an XML Schema	page 50
Creating a Data Model from Other Sources	page 52
Adding Validation Rules	page 59

Downloading Sample Getting Started Data

Overview

Your Artix Data Services installation includes a series of sample data files and completed examples that are designed to assist you in working your way through these demonstrations. Before you continue, you must ensure that you download all the relevant Getting Started material.

Download steps

Follow these steps to download the sample Getting Started material:

1. In the main window of the Artix Data Services Designer workbench, click the **Getting Started** link. This opens the **Confirm Download** dialog.
 2. Click **Yes** to proceed with the download. When the download completes, a message dialog opens indicating that the plug-in has been successfully installed.
-

Location of sample data

By default, the sample Getting Started material is downloaded to the following location on your machine:

Windows:

```
C:\Documents and Settings\username\My Documents\My IONA  
Projects\Getting Started
```

UNIX:

```
/userhome/My IONA Projects/Getting Started
```

Layout of sample data

The `Getting Started/Guide` folder contains a PDF copy of this book that you are currently reading. The `Getting Started/Samples` and `Videos` folder contains a series of subfolders that correspond to the various chapters in this book. Each subfolder under `Samples` and `Videos` contains various data files that you will need to complete various demonstrations. Each subfolder also contains a completed example of the end result of the particular demonstration it covers. As you work through the instructions in this book, you will be prompted at various stages to work with a particular sample file. Provided you have downloaded the Getting Started data as instructed, you are now ready to continue working your way through the demonstrations.

Creating a Data Model Manually

Overview

This section describes how to manually create two different data models—one called Accounts, and another called Customer.

In this section

This section discusses the following topics:

Creating the Accounts Data Model Manually	page 19
Creating the Customers Data Model Manually	page 31

Creating the Accounts Data Model Manually

Overview

This subsection demonstrates how to manually create an Accounts data model. The data model is built up from simple types into complex types. Each simple type has its own properties, such as minimum and maximum lengths, that are specified accordingly. The model contains two complex types—one that represents an individual account record (called Account) and another that represents a series of account records (called Accounts File). It then shows how to deploy the Accounts model and test its accuracy by parsing a valid text file through it.


Note: This demonstration is illustrated by the video tutorial within the Getting Started/Samples and Videos/Creating Data Models/Manually folder of your Artix Data Services Getting Started material. This sample data model is based on the information in the Accounts tab in the IONA Universal Banking System.xls file that is supplied within the same folder.

Note: Some types, such as dates, also require validation. However, validation rules are outside the scope of this particular demonstration and will be covered later in this chapter.

Creating the empty data model

Follow these steps to start creating your data model:

1. In the Project window of the workbench, ensure that `MyProject.iop` is opened and then navigate to the `My IONA Projects/Getting Started/Samples and Videos/Creating Data Models/Manually` folder.
2. Right click the `Creating a Data Model Manually` folder and select **New File/Directory**. Alternatively, click the the `Creating a Data Model Manually` folder and select **File > New File/Directory** from the menu bar. This opens the **New File/Directory** dialog.
3. Select **New Data Model**. This opens the **Enter name for data model** dialog.

4. Type "Accounts" in the available text box and click **OK**. This causes `Accounts.dod` to be automatically created and displayed in the Project and Explorer windows of the workbench. An `Accounts.dod` tab is also automatically opened in the main window of the workbench, and a dialog box is displayed prompting you to set the target namespace.
 5. Click the  icon in the dialog box to close it.
-

Creating an AccountNumber type

Now that you have created an empty data model, start creating data types for it. First, create an AccountNumber type as follows:

1. In the Explorer window, right click on `Accounts.dod` and select **New Component** from the context menu. This opens the **New Component** dialog.
 2. Click **New Atomic Simple Type**. This opens the **New Atomic Simple Type** dialog.
 3. Type "AccountNumber" in the text box and click **OK**. This opens the **Select Base Type** dialog.
 4. Expand **Text**, click **String**, and then click **OK**. "AccountNumber" is now automatically displayed under `Accounts.dod` in the Explorer window.
 5. Click "AccountNumber" in the Explorer window. This causes properties for the type to be automatically displayed in the Properties window.
 6. In the Properties window, scroll down to the **Validation** section and set the value for both **Min Length** and **Max Length** to 12.
-

Creating an AccountName type

Next create an AccountName type as follows:

1. In the Explorer window, right click on `Accounts.dod` and select **New Component** from the context menu. This opens the **New Component** dialog.
2. Click **New Atomic Simple Type**. This opens the **New Atomic Simple Type** dialog.
3. Type "AccountName" in the text box and click **OK**. This opens the **Select Base Type** dialog.

4. Expand **Text**, click **String**, and then click **OK**. "AccountName" is now automatically displayed under Accounts.dod in the Explorer window. Properties for the type are also automatically displayed in the Properties window.
 5. In the Properties window, scroll down to the **Validation** section and set the value for both **Min Length** and **Max Length** to 20.
-

Creating a Blocked type

Next create a Blocked type as follows:

1. In the Explorer window, right click on Accounts.dod and select **New Component** from the context menu. This opens the **New Component** dialog.
 2. Click **New Atomic Simple Type**. This opens the **New Atomic Simple Type** dialog.
 3. Type "Blocked" in the text box and click **OK**. This opens the **Select Base Type** dialog.
 4. Expand **Text**, click **String**, and then click **OK**. "Blocked" is now automatically displayed under Accounts.dod in the Explorer window. Properties for the type are also automatically displayed in the Properties window.
 5. In the Properties window, scroll down to the **Validation** section and set the value for both **Min Length** and **Max Length** to 1.
-

Creating OpeningBalance and ClosingBalance types

Next create an OpeningBalance type as follows:

1. In the Explorer window, right click on Accounts.dod and select **New Component** from the context menu. This opens the **New Component** dialog.
2. Click **New Atomic Simple Type**. This opens the **New Atomic Simple Type** dialog.
3. Type "OpeningBalance" in the text box and click **OK**. This opens the **Select Base Type** dialog.
4. Expand **Numeric**, click **decimal**, and then click **OK**. "OpeningBalance" is now automatically displayed under Accounts.dod in the Explorer window. Properties for the type are also automatically displayed in the Properties window.

- In the Properties window, scroll down to the **Validation** section and set the values for **Min Total Digits** and **Max Total Digits** to 1 and 16 respectively.

Now repeat steps 1–5 to create a ClosingBalance type. (In this case, make sure that you substitute each occurrence of "OpeningBalance" with "ClosingBalance" in the instructions.)

Creating a Customer type

Next create a Customer type as follows:

- In the Explorer window, right click on Accounts.dod and select **New Component** from the context menu. This opens the **New Component** dialog.
 - Click **New Atomic Simple Type**. This opens the **New Atomic Simple Type** dialog.
 - Type "Customer" in the text box and click **OK**. This opens the **Select Base Type** dialog.
 - Expand **Text**, select **String**, and then click **OK**. "Customer" is now automatically displayed under Accounts.dod in the Explorer window. Properties for the type are also automatically displayed in the Properties window.
 - In the Properties window, scroll down to the **Validation** section and set the values for both **Min Length** and **Max Length** to 6.
-

Creating a Currency Type

Next create a Currency type as follows:

- In the Explorer window, right click on Accounts.dod and select **New Component** from the context menu. This opens the **New Component** dialog.
- Click **New Atomic Simple Type**. This opens the **New Atomic Simple Type** dialog.
- Type "Currency" in the text box and click **OK**. This opens the **Select Base Type** dialog.
- Expand **Text**, select **String**, and click **OK**. "Currency" is now automatically displayed under Accounts.dod in the Explorer window. Properties for the type are also automatically displayed in the Properties window.

- In the Properties window, scroll down to the **Validation** section and set the values for both **Min Length** and **Max Length** to 3.

Creating OpeningBalanceDate, ClosingBalanceDate and LastStatementDate types

Next create an OpeningBalanceDate type as follows:

- In the Explorer window, right click on Accounts.dod and select **New Component** from the context menu. This opens the **New Component** dialog.
- Click **New Atomic Simple Type**. This opens the **New Atomic Simple Type** dialog.
- Type "OpeningBalanceDate" in the text box and click **OK**. This opens the **Select Base Type** dialog.
- Expand **Date & Time**, select **Generic Date**, and click **OK**. "OpeningBalanceDate" is now automatically displayed under Accounts.dod in the Explorer window. Properties for the type are also automatically displayed in the Properties window.

Now repeat steps 1–4 to create a ClosingBalanceDate and LastStatementDate type respectively. (In each case, make sure that you substitute each occurrence of "OpeningBalanceDate" with either "ClosingBalanceDate" or "LastStatementDate", as appropriate.)

Creating a LastStatementNo type

Next create a LastStatementNo type as follows:

- In the Explorer window, right click on Accounts.dod and select **New Component** from the context menu. This opens the **New Component** dialog.
- Click **New Atomic Simple Type**. This opens the **New Atomic Simple Type** dialog.
- Type "LastStatementNo" in the text box and click **OK**. This opens the **Select Base Type** dialog.
- Expand **Numeric**, select **integer**, and click **OK**. "LastStatementNo" is now automatically displayed under Accounts.dod in the Explorer window. Properties for the type are also automatically displayed in the Properties window.
- In the Properties window, scroll down to the **Validation** section and set the values for both **Min Total Digits** and **Max Total Digits** to 12.

Creating a CardNo type



Next create a CardNo type as follows:

1. In the Explorer window, right click on Accounts.dod and select **New Component** from the context menu. This opens the **New Component** dialog.
 2. Click **New Atomic Simple Type**. This opens the **New Atomic Simple Type** dialog.
 3. Type "CardNo" in the text box and click **OK**. This opens the **Select Base Type** dialog.
 4. Expand **Text**, select **String**, and click **OK**. "CardNo" is now automatically displayed under Accounts.dod in the Explorer window. Properties for the type are also automatically displayed in the Properties window.
 5. In the Properties window, scroll down to the **Validation** section and set the values for both **Min Length** and **Max Length** to 16.
-

Creating an Account complex type

Next create an Account complex type that will represent one account record whose fields are based on all the simple types you have already created, as follows:

1. In the Explorer window, right click on Accounts.dod and select **New Component** from the context menu. This opens the **New Component** dialog.
2. Click **New Complex Type**. This opens the **New Complex Type** window.
3. Type "Account" in the text box and click **OK**. The Account complex type is automatically displayed under Accounts.dod in the Explorer window. An Account tab is also automatically opened within the Accounts.dod tab in the main window of the workbench.
4. Select all simple types displayed under Accounts.dod in the Explorer window, by clicking the first simple type displayed and then clicking the last simple type while pressing the Shift key. This causes all simple types to appear highlighted in the Explorer window.
5. Drag and drop the highlighted simple types from the Explorer window over to the Account complex type in the main window of the workbench. This causes all the simple types to be displayed in the main window under the Account complex type.

6. Click the "Account" complex type in the Explorer window. This causes the properties for the complex type to be displayed in the Properties window.
7. For the purposes of this example, the account records are based on data in a fixed-format text file called Accounts.txt. The record format needs to be specified as a property of the "Account" complex type. In the Properties window, scroll down to the **Presentation** section and set the value for **Format Type** to `Fixed`.
8. Each record in the Accounts.txt file ends with a CRLF (carriage return line feed). This needs to be set as another property of the "Account" complex type, so that the data model will know to look for the CRLF at the end of each record it comes across in the text file. In the Properties window, click in the text area beside the **Terminator** field and then click the  icon in the field. This opens the **Insert Character** dialog.
9. Select **CR** and click **Insert**. Then select **LF** and click **Insert**. Then click **OK**. This causes `<CR><LF>` and `0D0A` to be displayed as the value for **Terminator**.
10. Select **File > Save All** from the menu bar, or click the  icon on the toolbar, to save the data model.


Creating an Accounts File complex type

Next create an Accounts File complex type that can consist of multiple instances of the Account complex type (that is, it can contain multiple account records) as follows:

1. In the Explorer window, right click on Accounts.dod and select **New Component** from the context menu. This opens the **New Component** dialog.
2. Click **New Complex Type**. This opens the **New Complex Type** window.
3. Type "Accounts File" in the text box and click **OK**. The Accounts File complex type is automatically displayed under Accounts.dod in the Explorer window. An Accounts File tab is also automatically opened within the Accounts.dod tab in the main window of the workbench.
4. Click the Account complex type in the **Explorer** window, and drag and drop it over to the Accounts File complex type in the main window of the workbench. This causes the Account complex type to be displayed in the main window under the Accounts File complex type.

5. The cardinality value determines how many instances of the Account complex type can pertain to the Accounts File complex type (that is, how many account records can pertain to the accounts file). This is set to 1 by default, which would mean that the accounts file could only contain one account record. For the purposes of this example, the accounts file needs to be able to contain one or more account records, so the cardinality value needs to be changed in this case. Right click the Account complex type in the **Component** column, select **Cardinality**, and then select 1..* instead.
6. Click Accounts.dod in the **Explorer** window. This causes the properties for the data model to be displayed in the Properties window.

Note: If Accounts.dod is not the currently open data model in the **Explorer** window, click the Accounts.dod tab in the main window of the workbench to open it.

7. Remember the tool tip about target namespaces that was displayed upon creating the data model. Now let's specify a target namespace for this data model. In the **General** section of the Properties window, set the value for **Target Namespace** to
<http://www.iona.com/ArtixDataServices/GettingStarted/Account>.
8. Select **File > Save All** from the menu bar, or click the  icon on the toolbar, to save the data model.

At this point, you have finished establishing the framework of your Accounts data model. It now consists of:

- An Accounts File complex type that can represent your accounts file.
- An Account complex type that can represent each record in your accounts file.
- Various simple types that can represent the various fields in each account record.

The next step is to validate the Accounts data model by checking to see if it can parse a valid text file.

Validating your data model and building object instances


You need to ensure that your data model is accurate by checking to see if it can parse some real-world data. You can do this using a feature of the Designer called the Run Wizard, which allows you to read data into a model and in so doing create Java class instances of that model. In this case, you can read the supplied Accounts.txt file into your Accounts data model.

Follow these steps:


1. Ensure that the Accounts.dod data model is currently open.
2. Right-click the Accounts File complex type in the Explorer window and select **Run Component**. This opens the Run Wizard dialog.

In this case, the **Name** field automatically defaults to "Accounts File" (that is, the name of the selected component) and the **Target** field defaults to the path location of the selected component. The **Build Before Running** check box is checked by default.

3. For the purposes of this demonstration, accept all the default values on the Run Wizard dialog and click **Run**.

This opens an Accounts File tab (with a  icon beside its name) within the Accounts.dod tab. This tab will be used to show the structure of the deployed object based on your data model. Because you have not yet loaded any data into the object, it is displayed for now in its empty state with a red X.

In the Messages window, three new tabs have been created at this point. The **Build** tab contains log4j messages relating to the building of Java classes from your data model. The **Run Accounts File** tab is empty at this point. The **Validate AccountsFile** tab displays a validation error at this point, because you have not yet loaded any data to validate the object.

4. Click the  icon in the Accounts File tab in the main window. This opens a Load Dialog window.
5. Click **Browse**. This opens the Load From File dialog.
6. Navigate to the `Getting Started/Samples and Videos/Creating Data Models/Manually` folder and select `Accounts.txt`. Then click **Open**. This reopens the Load Dialog with the full path to the selected file now displayed in the top field.

Notice how the **Format** field is set to "Textual" by default. Because we want to read in a text file in this case, accept the default format.



Note: The default format is based on properties set for the data model in the Properties window.

7. Click **Load**.
8. In the case of this demonstration, a format (parsing) error is now displayed in the **Run Accounts File** tab in the Messages window regarding number validation, and a red X appears beside AccountsFile in the Accounts File tab. This parsing error now needs to be corrected, as described next.

Fixing parsing errors relating to balance amounts

Parsing errors are an indication that a data model is not completely accurate. For the purposes of this demonstration, there is a parsing error relating to the OpeningBalance field. The Accounts.txt file expects the opening balance amount to consist of 14 integer digits and 2 fraction digits, but these have not been set as properties of the OpeningBalance type in the data model.

Follow these steps to fix the parsing error:

1. Click OpeningBalance in the Explorer window. This causes the properties for that type to be displayed in the Properties window.
2. In the Properties window, scroll down to the **Presentation/Advanced** section and type "." (that is, a period) in the **Decimal Separator** field. The value ". [2e]" is then displayed in that field.
3. In the Properties window, scroll down to the **Validation** section and set the values for **Min Integer Digits** and **Max Integer Digits** to 1 and 14 respectively.
4. Set the value for **Min Fraction Digits** and **Max Fraction Digits** to 0 and 2 respectively.
5. Select **File > Save All** from the menu bar, or click the  icon on the toolbar, to save the data model.
6. Close the Accounts File tab (with a  icon beside its name).



- Now repeat steps 1-3 in [“Validating your data model and building object instances” on page 27](#). (Because you have already tried to load the Accounts.txt file, the Designer will automatically try to reload it for you at that point).


The parsing error relating to OpeningBalance is now fixed, but there is now another parsing error relating to ClosingBalance. Again, the Accounts.txt file expects the closing balance amount to consist of 14 integer digits and 2 fraction digits, but these have not been set as properties of the ClosingBalance type in the data model.

Note: At this point, repeat steps 1-7 to fix the parsing error that relates to ClosingBalance.

Fixing parsing errors relating to dates

For the purposes of this demonstration, after you fix the parsing error that relates to ClosingBalance, there will be another parsing error this time relating to OpeningBalanceDate. The Accounts.txt file expects the opening balance date to have a particular date format, but this has not been set as a property of the OpeningBalanceDate type in the data model. Follow these steps to fix the parsing error:

- Click OpeningBalanceDate in the Explorer window. This causes the properties for that type to be displayed in the Properties window.
- In the Properties window, scroll down to the **Presentation** section and click the **Date Format** field. This opens a date format dialog.
- Click the  icon beside the **Pattern** field in the dialog. This opens the **Insert Character** dialog.
- The date format in this case needs to have a format of yyMMdd (note the case sensitivity). Double click "y" twice in the **Char** column, then double click "M" twice, and then double click "d" twice. The **Pattern** field on the **Insert Character** dialog now displays "yyMMdd".
- Click **OK**. The **Pattern** field in the first date format dialog now displays "yyMMdd" also.
- Click **OK**. The **Date Format** field in the Properties window now displays "yyMMdd" also.
- Select **File > Save All** from the menu bar, or click the  icon on the toolbar, to save the data model.

8. Close the Accounts File tab (with a  icon beside its name).
9. Now repeat steps 1-3 in [“Validating your data model and building object instances” on page 27](#). (Because you have already tried to load the Accounts.txt file, the Designer will automatically try to reload it for you at that point).

The parsing error relating to OpeningBalanceDate is now fixed, but there will still be two more parsing errors to be fixed—one relating to ClosingBalanceDate and another relating to LastStatementDate.

Note: At this point, repeat steps 1-9 twice more to fix the parsing errors that relate to ClosingBalanceDate and LastStatementDate respectively.

When the data model is finally accurate and all parsing errors have been fixed, Artix Data Services then creates instances of the model, based on your data, and a green tick appears beside AccountsFile in the Accounts File tab to indicate that parsing has been successful. The **Validate AccountsFile** tab in the Messages window also displays a `Validation passed` message. You can now expand the AccountsFile node in the main window to view all the records in the file.

Creating the Customers Data Model Manually

Overview

This subsection demonstrates how to manually create a Customers data model. The data model is built up from simple types into complex types. Each simple type has its own properties that are specified accordingly. The model contains two complex types—one that represents an individual customer record (called Customer) and another that represents a list of customer records (called Customers File). It then shows how to deploy the Customers model and test its accuracy by parsing a valid text file through it.

Note: An alternative way of creating the Customers data model is to import its contents from the `Customers.txt` file. You may skip this section and follow the instructions in [“Creating a Data Model from a Text File” on page 40](#) instead, if you wish to create the Customers data model from a text file rather than manually.


Note: The information on which this data model is based is contained in the `Customer Data` tab in the `IONA Universal Banking System.xls` file that is supplied within the `Getting Started/Samples and Videos/Creating Data Models/Manually` folder of your Artix Data Services Getting Started material.

Note: Some types, such as dates, also require validation. However, validation rules are outside the scope of this particular demonstration and will be covered later in this chapter.

Creating the empty data model

Follow these steps to start creating your data model:

1. In the Project window of the workbench, ensure that `MyProject.iop` is opened and then navigate to the `My IONA Projects/Getting Started/Samples and Videos/Creating Data Models/Manually` folder
2. Right click the `Creating a Data Model Manually` folder and select **New File/Directory**. Alternatively, click the `Creating a Data Model Manually` folder and select **File > New File/Directory** from the menu bar. This opens the **New File/Directory** dialog.
3. Select **New Data Model**. This opens the **Enter name for data model** dialog.

4. Type "Customers" in the available text box and click **OK**. This causes `Customers.dod` to be automatically created and displayed in the Project and Explorer windows of the workbench. A `Customers.dod` tab is also automatically opened in the main window of the workbench, and a dialog box is displayed prompting you to set the target namespace.
 5. Click the  icon in the dialog box to close it.
-

Creating a CustomerNumber type

Now that you have created an empty data model, start creating data types for it. First, create a `CustomerNumber` type as follows:

1. In the Explorer window, right click on `Customers.dod` and select **New Component** from the context menu. This opens the **New Component** dialog.
 2. Click **New Atomic Simple Type**. This opens the **New Atomic Simple Type** dialog.
 3. Type "CustomerNumber" in the text box and click **OK**. This opens the **Select Base Type** dialog.
 4. Expand **Text**, click **String**, and then click **OK**. "CustomerNumber" is now automatically displayed under `Accounts.dod` in the Explorer window. Properties for the type are also automatically displayed in the Properties window.
 5. In the Properties window, scroll down to the **Validation** section and set the value for both **Min Length** and **Max Length** to 6.
-

Creating a CustomerAcronym type

Next create a `CustomerAcronym` type as follows:

1. In the Explorer window, right click on `Customers.dod` and select **New Component** from the resultant context menu. This opens the **New Component** dialog.
2. Click **New Atomic Simple Type**. This opens the **New Atomic Simple Type** dialog.
3. Type "CustomerAcronym" in the text box and click **OK**. This opens the **Select Base Type** dialog.

4. Expand **Text**, click **String**, and then click **OK**. "CustomerAcronym" is now automatically displayed under Accounts.dod in the Explorer window. Properties for the type are also automatically displayed in the Properties window.
 5. In the Properties window, scroll down to the **Validation** section and set the value for both **Min Length** and **Max Length** to 12.
-

Creating AddressLine types

Next create an AddressLine1 type as follows:

1. In the Explorer window, right click on Customers.dod and select **New Component** from the resultant context menu. This opens the **New Component** dialog.
2. Click **New Atomic Simple Type**. This opens the **New Atomic Simple Type** dialog.
3. Type "AddressLine1" in the text box and click **OK**. This opens the **Select Base Type** dialog.
4. Expand **Text**, click **String**, and then click **OK**. "AddressLine1" is now automatically displayed under Accounts.dod in the Explorer window. Properties for the type are also automatically displayed in the Properties window.
5. In the Properties window, scroll down to the **Validation** section and set the value for **Min Length** to 0 and set the value for **Max Length** to 50.

Now repeat steps 1–5 to create AddressLine2, AddressLine3, AddressLine4, and AddressLine5 types respectively. (In each case, make sure that you substitute each occurrence of "AddressLine1" with the relevant type name.)

Creating other simple types


Now repeat the same steps to create the rest of the simple types that relate to the Customers data model. These types include:


Name	Atomic Type	Min Length	Max Length
PostZip Code	String	8	8
Tel Number	String	20	20
Email Address	String	50	50
BIC	String	11	11
Fax Number	String	20	20
Telex Number	String	20	20
Country of Residence	String	2	2
Fedwire Code	String	9	9
Chips Participant Code	String	4	4
Chips UID	String	6	6
Sort Code	String	0	6
Bankleitzhal Code	String	8	8

For details of these types, including the properties you need to set for them, refer to the `Customer Data` tab in the `IONA Universal Banking System.xls` file that is supplied within the `Getting Started/Samples and Videos/Creating Data Models/Manually` folder of your Artix Data Services Getting Started material.

Creating a Customer complex type

Next create a Customer complex type that will represent one customer record whose fields are based on all the simple types you have already created, as follows:

1. In the Explorer window, right click on Customers.dod and select **New Component** from the context menu. This opens the **New Component** dialog.
2. Click **New Complex Type**. This opens the **New Complex Type** window.
3. Type ""Customer" in the text box and click **OK**. The Customer complex type is automatically displayed under Customers.dod in the Explorer window. A Customer tab is also automatically opened within the Customers.dod tab in the main window of the workbench.
4. Select all simple types displayed under Customers.dod in the Explorer window, by clicking the first simple type displayed and then clicking the last simple type while pressing the Shift key. This causes all simple types to appear highlighted in the **Explorer** window.
5. Drag and drop the highlighted simple types from the **Explorer** window over to the Customer complex type in the main window of the workbench. This causes all the simple types to be displayed in the main window under the Customer complex type.
6. Click the "Customer" complex type in the **Explorer** window. This causes the properties for the complex type to be displayed in the **Properties** window.
7. For the purposes of this example, the customer records are based on data in a fixed-format text file called Customers.txt. The record format needs to be specified as a property of the "Customer" complex type. In the Properties window, scroll down to the **Presentation** section and set the value for **Format Type** to `Fixed`.
8. Each record in the Customers.txt file ends with a CRLF (carriage return line feed). This needs to be set as another property of the "Customer" complex type, so that the data model will know to look for the CRLF at the end of each record it comes across in the text file. In the Properties window, click in the text area beside the **Terminator** field and then click the  icon in the field. This opens the **Insert Character** dialog.

9. Select **CR** and click **Insert**. Then select **LF** and click **Insert**. Then click **OK**. This causes `<CR><LF>` and `OD0A` to be displayed as the value for **Terminator**.
10. Select **File > Save All** from the menu bar, or click the  icon on the toolbar, to save the data model.


Creating a Customers File complex type

Next create a Customers File complex type that can consist of multiple instances of the Customer complex type (that is, it can contain multiple customer records) as follows:

1. In the **Explorer** window, right click on Customers.dod and select **New Component** from the context menu. This opens the **New Component** dialog.
2. Click **New Complex Type**. This opens the **New Complex Type** window.
3. Type "Customers File" in the text box and click **OK**. The Customers File complex type is automatically displayed under Customers.dod in the Explorer window. A Customers File tab is also automatically opened within the Customers.dod tab in the main window of the workbench.
4. Click the Customer complex type in the **Explorer** window, and drag and drop it over to the Customers File complex type in the main window of the workbench. This causes the Customer complex type to be displayed in the main window under the Customers File complex type.
5. The cardinality value determines how many instances of the Customer complex type can pertain to the Customers File complex type (that is, how many customer records can pertain to the customers file). This is set to 1 by default, which would mean that the customers file could only contain one account record. For the purposes of this example, the customers file needs to be able to contain one or more customer records, so the cardinality value needs to be changed in this case. Right click the Customer complex type in the **Component** column, select **Cardinality**, and then select `1..*` instead.

6. Click Customers.dod in the **Explorer** window. This causes the properties for the data model to be displayed in the Properties window.

Note: If Customers.dod is not the currently open data model in the **Explorer** window, click the Customers.dod tab in the main window of the workbench to open it.

7. Remember the tool tip about target namespaces that was displayed upon creating the data model. Now let's specify a target namespace for this data model. In the **General** section of the Properties window, set the value for **Target Namespace** to
`http://www.iona.com/ArtixDataServices/GettingStarted/Customer`
8. Select **File > Save All** from the menu bar, or click the  icon on the toolbar, to save the data model.

At this point, you have finished establishing the framework of your Customers data model. It now consists of:

- A Customers File complex type that can represent your customers file.
- A Customer complex type that can represent each record in your customers file.
- Various simple types that can represent the various fields in each customer record.


The next step is to validate the Customers data model by checking to see if it can parse a valid text file.

Validating your data model and building object instances


You need to ensure that your data model is accurate by checking to see if it can parse some real-world data. You can do this using a feature of the Designer called the Run Wizard, which allows you to read data into a model and in so doing create Java class instances of that model. In this case, you can read the supplied Customers.txt file into your Customers data model.

Follow these steps:

1. Ensure that the Customers.dod data model is currently open.
2. Right-click the Customers File complex type in the Explorer window and select **Run Component**. This opens the Run Wizard dialog.
3. In this case, the **Name** field automatically defaults to "Customers File" (that is, the name of the selected component) and the **Target** field defaults to the path location of the selected component. The **Build Before Running** check box is checked by default.
4. For the purposes of this demonstration, accept all the default values on the Run Wizard dialog and click **Run**.

This opens a Customers File tab (with a  icon beside its name) within the Customers.dod tab. This tab will be used to show the structure of the deployed object based on your data model. Because you have not yet loaded any data into the object, it is displayed for now in its empty state with a red X.

In the Messages window, three new tabs have been created at this point. The **Build** tab contains log4j messages relating to the building of Java classes from your data model. The **Run Customers File** tab is empty at this point. The **Validate CustomersFile** tab displays a validation error at this point, because you have not yet loaded any data to validate the object.

5. Click the  icon in the Customers File tab in the main window. This opens a Load Dialog window.
6. Click **Browse**. This opens the Load from File dialog.
7. **Navigate to the Getting Started/Samples and Videos/Creating Data Models/Manually folder and select Customers.txt. Then click Open.** This reopens the Load Dialog with the full path to the selected file now displayed in the top field.

Notice how the **Format** field is set to "Textual" by default. Because we want to read in a text file in this case, accept the default format.

Note: The default format is based on properties set for the data model in the Properties window.

8. Click **Load**.
9. In the case of this demonstration, there are no parsing errors at this point, so Artix Data Services creates instances of the model, based on your data. A green tick appears beside CustomersFile in the Customers File tab to indicate that parsing has been successful. The **Validate CustomersFile** tab in the Messages window also displays a `Validation passed` message. You can now expand the CustomersFile node in the main window to view four customer records.

Creating a Data Model from a Text File

Overview

This section describes how to create a data model by importing a text file. First, it demonstrates how to create a Transactions data model by importing a Transactions.txt file. Then it demonstrates how to create a Customers data model by importing a Customers.txt file.

In this section

This section discusses the following topics:

Creating the Transactions Data Model from Transactions.txt page 41
--

Creating the Customers Data Model from Customers.txt page 45
--

Creating the Transactions Data Model from Transactions.txt

Overview

This subsection demonstrates how to create a Transactions data model by importing a Transactions.txt file. In the Text File Import Wizard, you can set properties for the fields associated with a model instead of doing so in the Properties window outside the wizard. After creating a model, you can test its validity by parsing a valid text file through it.

Note: This sample data model is based on the Transactions.txt file that is supplied within the Getting Started/Samples and Videos/Creating Data Models/From a Text File folder of your Artix Data Services Getting Started material..

Steps


Follow these steps to start creating your data model:

1. In the Project window of the workbench, ensure that MyProject.iop is opened and then navigate to the My IONA Projects/Getting Started/Samples and Videos/Creating Data Models/From a Text File folder
2. Right click the Creating a Data Model from a Text File folder and select **Import > Choose Importer**. This opens the **Choose Importer** dialog.
3. Select **Import Text File**. This opens the **Import File** panel of the Text File Import Wizard.
4. Navigate to Getting Started/Samples and Videos/Creating Data Models/From a Text File. Then select Transactions.txt and click **Next**. This opens the **Model Directory** panel.
5. Accept the default folder Creating a Data Model from a Text File as the location where you want the data model to be stored. Then click **Next**. This opens the **Profiles** panel.
6. Notice the **Advanced** button that is in the **Steps** section on the left-hand side of the panel. Alternately clicking the **Advanced** button displays and hides some optional items in the list of steps.
7. Click the **Advanced** button to display the optional steps and then click **Next**. This opens the **Mapping File** panel.

8. Click **Next**. This opens the **Model Name & Target Namespace** panel. Notice how the model name defaults to the name of the file that is being imported. Leave the target namespace for now, because it can be specified at a later stage.
9. Click **Next**. This opens the **File Encoding & Text Quotation** panel.
10. Click **Next**. This opens the **Record Types** panel displaying one "Header" row and one "Row" row. The header is separated from rows as displayed here. Notice how the check box in the **Header** column is correctly checked for the "Header" row. (Do not adjust this.)
11. In the **Name** column, double click on "Row", type "Customer Details" as the value instead, and the press Enter. Notice how step 9 in the left-hand pane automatically changes from "Row" to "Customer Details".
12. Click **Next**. This opens the **Header** panel.
13. The text file is a delimited format file and this has been automatically picked up by the wizard. Notice how the delimiter is set as a comma (do not adjust this). Click the various columns in the **Preview** table and notice how the values in the **Selected Column Name** and **Selected Column Data Type** fields change accordingly. In this case, the selected column data type is always "String", because these are header values.
14. Click **Next** to open the **Customer Details** panel.

Note: Notice how the panel name here, "Customer Details", is based on the change that you made on the Record Types panel. If you had not made that change, the panel name here would be called "Row" instead.


15. The text file is a delimited format file and this has been automatically picked up by the wizard. Notice how the delimiter is set as a comma (do not adjust this). Click the various columns in the **Preview** table and notice how the values in the **Selected Column Name** and **Selected Column Data Type** fields change accordingly.
16. Click **Finish**. This causes `Transactions.dod` to be automatically created and displayed in the Project and Explorer windows of the workbench. A `Transactions.dod` tab is also automatically displayed in the main window of the workbench.

17. Click Transactions.dod in the **Explorer** window. This causes the properties for the data model to be displayed in the Properties window.
18. In the **General** section of the Properties window, set the value for **Target Namespace** to
<http://www.iona.com/ArtixDataServices/GettingStarted/Transaction>.
19. Select **File > Save All** from the menu bar, or click the  icon on the toolbar, to save the data model.
20. In the Explorer window, expand "File" and double click the Transactions complex type. This opens a Transactions tab within the Transactions.dod tab in the main window of the workbench. Expand the Header and Customer Details elements to view the contents. Compare the details displayed with those in the Transactions.txt file that you imported.

Validating your data model and building object instances


You need to ensure that your data model is accurate by checking to see if it can parse some real-world data. In this case, you can read the supplied Transactions.txt file into your Transactions data model, as follows:

1. Ensure that the Transactions.dod data model is currently open.
2. Expand "File" and right-click the Transactions complex type in the Explorer window and select **Run Component**. This opens the Run Wizard dialog.
3. In this case, the **Name** field automatically defaults to "Transactions" (that is, the name of the selected component) and the **Target** field defaults to the path location of the selected component. The **Build Before Running** check box is checked by default.
4. For the purposes of this demonstration, accept all the default values on the Run Wizard dialog and click **Run**.

This opens a Transactions tab (with a  icon beside its name) within the Transactions.dod tab. This tab will be used to show the structure of the deployed object based on your data model. Because you have not yet loaded any data into the object, it is displayed for now in its empty state with a red X.

In the Messages window, three new tabs have been created at this point. The **Build** tab contains log4j messages relating to the building of

Java classes from your data model. The **Run Transactions** tab is empty at this point. The **Validate Transactions** tab displays a validation error at this point, because you have not yet loaded any data to validate the object.

5. Click the  icon in the Transactions tab in the main window. This opens a Load Dialog window.
6. Click **Browse**. This opens the Load from File dialog.
7. Navigate to the `Getting Started/Samples and Videos/Creating Data Models/From a Text File` folder and select `Transactions.txt`. Then click **Open**. This reopens the Load Dialog with the full path to the selected file now displayed in the top field.

Notice how the **Format** field is set to "Textual" by default. Because we want to read in a text file in this case, accept the default format.

Note: The default format is based on properties set for the data model in the Properties window.

8. Click **Load**.
9. In the case of this demonstration, there are no parsing errors at this point, so Artix Data Services creates instances of the model, based on your data. A green tick appears beside Transactions in the Transactions tab to indicate that parsing has been successful. The **Validate Transactions** tab in the Messages window also displays a `Validation passed` message. You can now expand the Transactions node in the main window to view a Header and various CustomerDetails records.

Creating the Customers Data Model from Customers.txt

Overview

This subsection demonstrates how to create a Customers data model by importing a Customers.txt file. In the Text File Import Wizard, you can set properties for the fields associated with a model instead of doing so in the Properties window outside the wizard. After creating a model, you can test its validity by parsing a valid text file through it.

Note: You may skip this section if you have already followed the instructions in [“Creating the Customers Data Model Manually” on page 31](#).


Note: This sample data model is based on the Customers.txt file that is supplied within the Getting Started/Samples and Videos/Creating Data Models/From a Text File folder of your Artix Data Services Getting Started material.

Steps

Follow these steps to start creating your data model:

1. In the Project window of the workbench, ensure that MyProject.iop is opened and then navigate to the My IONA Projects/Getting Started/Samples and Videos/Creating Data Models/From a Text File folder.
2. Right click the Creating a Data Model from a Text File folder and select **Import > Choose Importer**. This opens the **Choose Importer** dialog.
3. Select **Import Text File**. This opens the **Import File** panel of the Text File Import Wizard.
4. Navigate to Getting Started/Samples and Videos/Creating Data Models/From a Text File. Then select Customers.txt and click **Next**. This opens the **Model Directory** panel.
5. Accept the default folder Creating a Data Model from a Text File as the location where you want the data model to be stored. Then click **Next**. This opens the **Profiles** panel.

6. Notice the **Advanced** button that is in the **Steps** section on the left-hand side of the panel. Alternately clicking the **Advanced** button displays and hides some optional items in the list of steps.
7. Click **Advanced** to display the optional steps and then click **Next**. This opens the **Mapping File** panel.
8. Click **Next**. This opens the **Model Name & Target Namespace** panel. Notice how the model name defaults to the name of the file that is being imported. Leave the target namespace for now, because it can be specified at a later stage.
9. Click **Next**. This opens the **File Encoding & Text Quotation** panel.
10. Click **Next**. This opens the **Record Types** panel displaying one "Row" row.
11. Click the value in the **Type** column and select "Fixed Length".
12. Click **Next**. This opens the **Row** panel.
13. According to the data in the `Customers Data` tab in the `IONA Universal Banking System.xls` file, the length for Customer Number is 6, so in the **Fixed Offset Properties** section, click the 7th column to automatically place a boundary between the 6th and 7th columns. This causes a new column, to be displayed in the **Preview - Column Data Types** section.
14. Click the first column in the **Preview - Column Data Types** section. Then type "Customer Number" in the **Selected Column Name** field and press Enter. This causes the first column name to change to "Customer Number".
15. Select "String" as the value in the **Selected Column Data Type** field.
16. According to the data in the `Customers Data` tab in the `IONA Universal Banking System.xls` file, the length for Customer Acronym is 12, so in the **Fixed Offset Properties** section, click the 19th column to automatically place a boundary between the 18th and 19th columns. This causes a new column, to be displayed in the **Preview - Column Data Types** section.
17. Click the second column in the **Preview - Column Data Types** section. Then type "Customer Acronym" in the **Selected Column Name** field and press Enter. This causes the second column name to change to "Customer Acronym".


18. Select "String" as the value in the **Selected Column Data Type** field.
19. Repeat steps 16-18 for the rest of the fields in the `Customers Data` tab in the `IONA Universal Banking System.xls` file. Because the fields are of fixed length, boundaries can be easily determined as the last column before the start of the next letter. So, for example, after the "Customer Acronym" column, click "F" in the 19th column to determine the boundary of the "Address Line 1" column. Similarly, click "W" in the 69th column to determine the boundary of the "Address Line 2" column.
20. After all boundaries have been determined and the correct column names have been specified in each case, click **Finish**. This causes `Customers.dod` to be automatically created and displayed in the Project and Explorer windows of the workbench. A `Customers.dod` tab is also automatically displayed in the main window of the workbench.
21. In the Explorer window, expand "File", right-click the "Customers" complex type, select **Rename**, and rename it to "Customers File".
22. In the Explorer window, expand "Records", right-click the "Row" complex type, select **Rename**, and rename it to "Customer".
23. Click `Customers.dod` in the **Explorer** window. This causes the properties for the data model to be displayed in the Properties window.
24. In the **General** section of the Properties window, set the value for **Target Namespace** to
<http://www.iona.com/ArtixDataServices/GettingStarted/Customer>.
25. Select **File > Save All** from the menu bar, or click the  icon on the toolbar, to save the data model.

Validating your data model and building object instances


You need to ensure that your data model is accurate by checking to see if it can parse some real-world data. In this case, you can read the supplied `Customers.txt` file into your `Customers` data model, as follows:

1. Ensure that the `Customers.dod` data model is currently open.
2. Right-click the `Customers File` complex type in the Explorer window and select **Run Component**. This opens the Run Wizard dialog.

3. In this case, the **Name** field automatically defaults to "Customers File" (that is, the name of the selected component) and the **Target** field defaults to the path location of the selected component. The **Build Before Running** check box is checked by default.
4. For the purposes of this demonstration, accept all the default values on the Run Wizard dialog and click **Run**.

This opens a Customers File tab (with a  icon beside its name) within the Customers.dod tab. This tab will be used to show the structure of the deployed object based on your data model. Because you have not yet loaded any data into the object, it is displayed for now in its empty state with a red X.

In the Messages window, three new tabs have been created at this point. The **Build** tab contains log4j messages relating to the building of Java classes from your data model. The **Run Customers File** tab is empty at this point. The **Validate CustomersFile** tab displays a validation error at this point, because you have not yet loaded any data to validate the object.

5. Click the  icon in the Customers File tab in the main window. This opens a Load Dialog window.
6. Click **Browse**. This opens the Load from File dialog.
7. **Navigate to the Getting Started/Samples and Videos/Creating Data Models/From a Text File folder and select Customers.txt. Then click Open.** This reopens the Load Dialog with the full path to the selected file now displayed in the top field.

Notice how the **Format** field is set to "Textual" by default. Because we want to read in a text file in this case, accept the default format.

Note: The default format is based on properties set for the data model in the Properties window.

8. Click **Load**.
9. In the case of this demonstration, there are no parsing errors at this point, so Artix Data Services creates instances of the model, based on your data. A green tick appears beside CustomersFile in the Customers File tab to indicate that parsing has been successful. The **Validate**

CustomersFile tab in the Messages window also displays a `Validation passed` message. You can now expand the `CustomersFile` node in the main window to view four customer records.

Creating a Data Model from an XML Schema

Overview


This section describes how to create a data model by importing an XML schema. It demonstrates how to create a Statements data model by importing a Statements.xsd file. In the XML Schema Import Wizard, you can set properties for the fields associated with the model instead of doing so in the Properties window outside the wizard. After creating the model, you can test its validity by parsing a valid XML file through it.

Note: This sample data model is based on the `Statements.xsd` file that is supplied within the `Getting Started/Samples` and `Videos/Creating Data Models/From an XML Schema` folder of your Artix Data Services Getting Started material..

Steps

Follow these steps to start creating your data model:

1. In the Project window of the workbench, ensure that `MyProject.iop` is opened and then navigate to the `My IONA Projects/Getting Started/Samples` and `Videos/Creating Data Models/From an XML Schema` folder
2. Right click the `Creating a Data Model from an XML Schema` folder and select **Import > Choose Importer**. This opens the **Choose Importer** dialog.
3. Select **Import XML Schema**, This opens the **Files To Import** panel of the XML Schema Import Wizard.
4. Navigate to `Getting Started/Samples` and `Videos/Creating Data Models/From an XML Schema`. Then select `Statements.xsd` and click **Next**. This opens the **Target Directory** panel.
5. Accept the default folder `Creating a Data Model from an XML Schema` as the location where you want the data model to be stored.
6. Notice the **Advanced** button that is in the **Steps** section on the left-hand side of the panel. Alternately clicking the **Advanced** button displays and hides some optional items in the list of steps.

7. Click **Advanced** to display the optional steps and then click **Next**. This opens the **Profiles** panel. Accept the defaults for the purposes of this example.
8. Click **Next**. This opens the **Mapping File** panel. There is no mapping file associated with the XML schema, so you do not need to select a mapping file.
9. Click **Finish**. This causes `Statements.dod` to be automatically created and displayed in the Project and Explorer windows of the workbench. A `Statements.dod` tab is also automatically opened in the main window of the workbench.
10. Select **File > Save All** from the menu bar, or click the  icon on the toolbar, to save the data model.
11. Click `Statements.dod` in the **Explorer** window. This causes the properties for the data model to be displayed in the Properties window.
12. Notice how the value for **Target Namespace** has been automatically populated in this case in the Properties window.

Creating a Data Model from Other Sources

Overview

In Artix Data Services, there are several importers available to create data models. These include text files, XML schemas, XML instance documents, Java classes, and databases. The principle behind creating data models by importing schemas or databases is the same despite the fact that what is being imported is different. This section describes how to create a data model by importing a database or an XML file.

In this section

This section discusses the following topics:

Creating a Data Model from a Set of XML Documents	page 53
Creating a Data Model from a Database	page 56

Creating a Data Model from a Set of XML Documents

Overview


This subsection demonstrates how to create an AccountsXML data model by importing an AccountsXML.xml file. In the XML Instance(s) Import Wizard, you can set properties for the fields associated with a model instead of doing so in the Properties window outside the wizard. After creating the model, you can test its validity by parsing a valid XML file through it.

Note: This sample data model is based on the AccountsXML.xml file that is supplied within the Getting Started/Samples and Videos/Creating Data Models/From Other Sources folder of your Artix Data Services Getting Started material..

Steps

Follow these steps to start creating your data model:


1. In the Project window of the workbench, ensure that MyProject.iop is opened and then navigate to the My IONA Projects/Getting Started/Samples and Videos/Creating Data Models/From Other Sources folder
2. Right click the Creating a Data Model from Other Sources folder and select **Import > Choose Importer**. This opens the **Choose Importer** dialog.
3. Select **Import XML Instance(s)**, This opens the **Files To Import** panel of the XML Instance(s) Import Wizard.
4. Navigate to Getting Started/Samples and Videos/Creating Data Models/From Other Sources. Then select AccountsXML.sml and click **Next**. This opens the **Target Directory** panel.
5. Accept the default folder Creating a Data Model from Other Sources as the location where you want the data model to be stored.
6. Notice the **Advanced** button that is in the **Steps** section on the left-hand side of the panel. Alternately clicking the **Advanced** button displays and hides some optional items in the list of steps.
7. Click **Advanced** to display the optional steps and then click **Next**. This opens the **Profiles** panel. Accept the defaults for the purposes of this example.

8. Click **Next**. This opens the **Mapping File** panel. There is no mapping file associated with the XML instance documents, so you do not need to select a mapping file.
9. Click **Finish**. This causes `AccountXML.dod` to be automatically created and displayed in the Project and Explorer windows of the workbench. The `AccountsXML.dod` tab is also automatically displayed in the main window of the workbench.
10. Select **File > Save All** from the menu bar, or click the  icon on the toolbar, to save the data model.
11. In the Explorer window, expand "AccountsFile" and double click the AccountsFile complex type. This opens an AccountsFile tab within the AccountsXML.dod tab in the main window of the workbench. Expand the Account element to view the contents. Compare the details displayed with those in the original Accounts.dod data model. Notice that each field is prefixed with "account:" (for example, the currency field is `account:currency`).


Validating your data model and building object instances

You need to ensure that your data model is accurate by checking to see if it can parse some real-world data. In this case, you can read the supplied AccountsXML.xml file into your AccountsXML data model, as follows:

1. Ensure that the AccountsXML.dod data model is currently open.
2. Right-click the AccountsFile complex type in the Explorer window and select **Run Component**. This opens the Run Wizard dialog.
3. In this case, the **Name** field automatically defaults to "AccountsFile" (that is, the name of the selected component) and the **Target** field defaults to the path location of the selected component. The **Build Before Running** check box is checked by default.
4. For the purposes of this demonstration, accept all the default values on the Run Wizard dialog and click **Run**.

This opens an AccountsFile tab (with a  icon beside its name) within the AccountsXML.dod tab. This tab will be used to show the structure of the deployed object based on your data model. Because you have not yet loaded any data into the object, it is displayed for now in its empty state with a red X.

In the Messages window, three new tabs have been created at this point. The **Build** tab contains log4j messages relating to the building of Java classes from your data model. The **Run AccountsFile** tab is empty at this point. The **Validate AccountsFile** tab displays a validation error at this point, because you have not yet loaded any data to validate the object.

5. Click the  icon in the AccountsFile tab in the main window. This opens a Load Dialog window.
6. Click **Browse**. This opens the Load from File dialog.
7. Navigate to the `Getting Started/Samples and Videos/Creating Data Models/From Other Sources` folder and select `AccountsXML.xml`. Then click **Open**. This reopens the Load Dialog with the full path to the selected file now displayed in the top field.

Ensure that the **Format** field is set to "XML". Because we want to read in an XML file in this case, XML must be the specified format.

Note: The default format is based on properties set for the data model in the Properties window.

8. Click **Load**.
9. In the case of this demonstration, there are no parsing errors at this point, so Artix Data Services creates instances of the model, based on your data. A green tick appears beside AccountsFile in the AccountsFile tab to indicate that parsing has been successful. The **Validate AccountsFile** tab in the Messages window also displays a `Validation passed` message. You can now expand the AccountsFile node in the main window to view four account records.

Creating a Data Model from a Database

Overview

This subsection describes how to create a data model by importing a database. It demonstrates how to create an IONA Banking System data model by importing a MySQL database called "ionabankingsystem".

Prerequisites

Before you proceed, you must first use MySQL on your machine to create the sample "ionabankingsystem" database. IONA has supplied a text file called `IONAUBS_SQL.txt` that contains the SQL necessary to create the database and its constituent tables. This text file is supplied in the `Getting Started/Samples` and `Videos/Creating Data Models/From Other Sources` folder. Use the "source" option in MySQL to execute the statements in the text file.

Steps


After you have used MySQL to create the "ionabankingsystem" database, follow these steps to start creating your data model:

1. In the Project window of the workbench, ensure that `MyProject.iop` is opened and then navigate to the `My IONA Projects/Getting Started/Samples` and `Videos/Creating Data Models/From Other Sources` folder
2. Right click the `Creating a Data Model from Other Sources` folder and select **Import > Choose Importer**. This opens the **Choose Importer** dialog.
3. Select **Import Database**, This opens the **Model Directory** panel of the Import Database Wizard.
4. Accept the default folder `Creating a Data Model from Other Sources` as the location where you want the data model to be stored. Then click **Next**. This opens the **Connection Properties** panel.
5. Type "Iona banking system" in the **Model Name** field.
6. Type "`http://www.iona.com/ArtixDataServices/GettingStarted/IonaBankingSystem`" in the **Target Namespace** field.

7. For the purposes of this demonstration, select "MySQL" in the **Database Dialect** field. (This indicates the type of database from which you want to import.) The **JDBC Driver Class Name** field is then automatically populated with "com.mysql.jdbc.Driver".
8. Update the value in the **Database URL** field with the name of your database, so make sure that the value reads as "jdbc:mysql://localhost/ionabankingsystem" (where *localhost* represents your machine name).
9. Type a valid user name and password in the **Username** and **Password** fields.
10. Click **Next**. This opens the **Import Type** panel. Notice how the **Automatic table detection** check box is checked by default. (Do not modify this selection).
11. Click **Next**. This opens the **Table selection** panel with a list of all possible tables in your database that may be imported. Notice how all tables in the database are selected for import. Also, notice how the **Import related tables** check box and the **Child only** button are both selected by default. (Do not modify these selections.)
12. Click **Next**. This opens the **Import Options** panel. Notice the various default selections and values on this panel. (Do not modify these.)
13. Click **Next**. This opens the **Types Mapping** panel. At this stage, it is not certain what the mappings should be changed to, and types can be changed later anyway. So you can ignore this panel for now.
14. Click **Next**. This displays all of your database tables in the order in which they were created. In each case, all the fields and their types and the primary keys are displayed. You may change the types at this stage or you can wait until later.

Note: Some characters such as "/", "(" and ")" are incompatible with Artix Data Services Designer. If some of your fields have such characters in them, Artix Designer prompts you to change the name.

15. Click **Finish** after the last table is displayed. This causes `Iona Banking System.dod` to be automatically created and displayed in the Project and Explorer windows of the workbench. In this case, the imported tables are created as complex types.

16. Select **File > Save All** from the menu bar, or click the  icon on the toolbar, to save the data model.

Adding Validation Rules

Overview

Data types such as dates or elements with a type of "double" must be validated to enable them to work in Artix Data Services Designer. Validation is commonly performed in the Properties window. Some properties have lists (that is, enumerations) associated with them which are defined in the Properties window. Elements with a type of "double" require integer and fraction composition to be specified. This demonstration shows how to set up such validation rules for the Accounts and Customers data models.

Note: The data models used in this section have already been created in [“Creating a Data Model Manually” on page 18](#).

In this section

This section discusses the following topics:

Adding Validation Rules for Accounts Data Model	page 60
Adding Validation Rules for Transactions Data Model	page 63

Adding Validation Rules for Accounts Data Model

Overview

This subsection demonstrates how to set up validation rules for the Accounts.dod data model.

Note: The validation values assigned in this demonstration are based on the values specified in the Accounts with Validation tab in the IONA Universal Banking System.xls file.



Opening the Accounts.dod file



Follow these steps to open the Accounts.dod file (if it is not already open):

1. In the Project window of the workbench, ensure that `MyProject.iop` is opened and then navigate to the `My IONA Projects/Getting Started/Samples and Videos/Creating Data Models/Manually` folder
 2. Right click the `Accounts.dod` file and select **Open Selected**. This causes `Accounts.dod` to be automatically displayed in the Explorer window of the workbench. The `Accounts.dod` tab is also automatically displayed in the main window of the workbench.
-

Adding validation rules for Blocked type


Follow these steps to add validation rules for the Blocked type:


1. Click "Blocked" in the Explorer window. This causes the properties for that type to be displayed in the Properties window.
2. In the Properties window, scroll down to the **Validation** section and click in the **Enumeration** field. Then click the down arrow in the field to open the **Select Component** dialog.
3. Click **Enumeration**. This opens the **New Enumeration** dialog.
4. Type "Blocked" as the name of the enumeration and click **OK**. This opens a Blocked tab (with a  icon beside its name) within the Accounts.dod tab.
5. Click the  icon to add a new value to the enumeration. This opens the **New Enumeration Value** dialog.
6. Type "Y" and click **OK**. This causes a new row to be added to the Blocked tab, with "Y" as the displayed value.

7. Double click the **Name** column of the "Y" row, type "Yes" and press Enter. "Yes" is now displayed as the name of the "Y" value.
8. Click the  icon to add a new value to the enumeration. This opens the **New Enumeration Value** dialog.
9. Type "N" and click **OK**. This causes a new row to be added to **Enumeration** with "N" as the displayed value.
10. Double click the **Name** column of the "N" row, type "No" and press Enter. "No" is now displayed as the name of the "N" value.
11. Select **File > Save All** from the menu bar, or click the  icon on the toolbar, to save the data model.

Adding validation rules for Card No type

Follow these steps to add validation rules for the Card No type:

1. Click "Card No" in the Explorer window. This causes the properties for that type to be displayed in the Properties window.
2. In the Properties window, scroll down to the **Validation** section and click in the **Pattern** field.
3. Select **Java Regex** from the drop down list and then click the  icon to the right of the field. This displays "Java Regex" in the first half of the **Pattern** field and also opens the **Insert Character** dialog.
4. Select the following pattern or type it manually in the **Pattern** field on the **Insert Character** dialog:

```
[0-9]{4}[0-9]{4}[0-9]{4}[0-9]{4}
```
5. Click **OK**. The pattern is then displayed in the Properties window.
6. To ensure that all validation is correct, in the Explorer window right click Accounts.dod and select **Verify Components**. This opens a **Verification** tab in the Messages window and the last line should read "Verification passed".
7. Select **File > Save All** from the menu bar, or click the  icon on the toolbar, to save the data model.

Validating your data model

Follow these steps to validate your data model:

1. Ensure that the Accounts.dod data model is currently open.
2. Right-click the Accounts File complex type in the Explorer window and select **Run Component**. This opens the Run Wizard dialog.

In this case, the **Name** field automatically defaults to "Accounts File" (that is, the name of the selected component) and the **Target** field defaults to the path location of the selected component. The **Build Before Running** check box is checked by default.

3. For the purposes of this demonstration, accept all the default values on the Run Wizard dialog and click **Run**.

This opens an Accounts File tab (with a ▶ icon beside its name) within the Accounts.dod tab. This tab shows the structure of the deployed object based on your data model. Because you previously loaded data into the object, the data is automatically reloaded at this point and the AccountsFile node is expanded.

In this case, the four records are showing up in error. Expand the four records and you will notice that the CardNo element is showing an error in each case.

Click the **Validation** tab at the bottom of the Designer workbench to open the Validation window. Expand the node beside the data model name in the Validation window to view the invalid records. Notice how the error details that the card number does not match the Java Regex pattern.

Click the **Validate AccountsFile** tab in the Messages window and notice how the error details that the card number does not match the Java Regex pattern.

This proves that the validation rule for CardNo is working, because it has highlighted as invalid all records whose card numbers do not match the correct pattern.

Adding Validation Rules for Transactions Data Model

Overview

Xpath is predominantly used to apply validation rules to models. This subsection demonstrates how to use Xpath to set up a rule to validate the Commission field in the Transactions.dod data model.

Opening the Transactions.dod file

Follow these steps to open the Transactions.dod file (if it is not already open):




1. In the Project window of the workbench, ensure that `MyProject.iop` is opened and then navigate to the `My IONA Projects/Getting Started/Samples and Videos/Creating Data Models/From a Text File folder`
 2. Right click the `Transactions.dod` file and select **Open Selected**. This causes `Transactions.dod` to be automatically displayed in the Explorer window of the workbench. The `Transactions.dod` tab is also automatically displayed in the main window of the workbench.
-

Adding a rule for Commission type

In this case, the validation rule is going to be created as a global validation rule so that it can be reused: Follow these steps to create the validation rule:

1. Right click `Transactions.dod` in the Explorer window and select **New Component**. This opens the **New Component** dialog.
2. Select **New Validation Rule**. This opens the **New Validation Rule** dialog.
3. Type "Commission Check" in the text box and click **OK**. This automatically opens a Commission Check tab within the `Transactions.dod` tab in the main window of the workbench, with a default type of XPath. In this case, the rule is entered in the left hand pane of the tab and XPath syntax is displayed in the right hand pane

Note: Creating a validation rule directly under the `.dod` file itself means that it is a global validation rule rather than being tied specifically to any one particular element within the data model.

4. In this case, the rule will determine whether the value of commission is greater than the product of 0.02 and the value of amount. Therefore, click in the shaded area at the top of the left-hand pane in the main window and type "Commission > 0.02 * Amount" as the XPath rule.
5. If the validation rule is true, the data model should throw an error. Therefore, type "Commission Error" in the **Error Message** pane.
6. Select **File > Save All** from the menu bar, or click the  icon on the toolbar, to save the data model.
7. In the Explorer window, expand **File** and double click the Transactions complex type. This opens the Transactions complex type in the main window of the workbench.
8. Because the node names used in the Xpath rule do not refer to the parent node in any way, the rule must be applied directly to the Customer Details complex type so that the model can interpret the validation rule correctly. In the **Type** column, click "Customer Details". This displays the properties for the Customer Details type in the Properties window.
9. In the Properties window, scroll down to the **Validation** section and click the field beside **Validation Rules**. This opens a validation rules dialog.
10. Click the  icon. This opens the **Add Validation Rule** dialog.
11. Now apply the global Commission Check validation rule to the Customer Details type. Select the Commission Check global validation rule and click **OK**. This adds Commission Check to the validation rules dialog.
12. Click **OK** to close the validation rules dialog. The **Validation Rules** field in the Properties window now displays "1".
13. Select **File > Save All** from the menu bar, or click the  icon on the toolbar, to save the data model.

Validating your data model

Follow these steps to validate your data model:

1. Ensure that the Transactions.dod data model is currently open.
2. Expand "File", right-click the Transactions complex type in the Explorer window, and select **Run Component**. This opens the Run Wizard dialog.
3. In this case, the **Name** field automatically defaults to "Transactions" (that is, the name of the selected component) and the **Target** field defaults to the path location of the selected component. The **Build Before Running** check box is checked by default.
4. For the purposes of this demonstration, accept all the default values on the Run Wizard dialog and click **Run**.

This opens a Transactions tab (with a ▶ icon beside its name) within the Transactions.dod tab. This tab will be used to show the structure of the deployed object based on your data model. Because you previously loaded data into the object, the data is automatically reloaded at this point and the Transactions node is expanded.

In this case, some Customer Details records show a green (valid) tick and some show a red (invalid) X. Expand the first Customer Details record that is showing a red (invalid) X, and check the value of "Amount" and the value of "Commission". Notice how Amount is -500 and Commission is 8.

Click the **Validation** tab at the bottom of the Designer workbench to open the Validation window. Expand the node beside the data model name in the Validation window to view the invalid records. Notice how "Commission Error" is displayed as the error message in each case.

Click the **Validate Transactions** tab in the Messages window and notice how "Commission Error" is also displayed as the error message there.

This proves that the Commission Check validation rule is working, because it has highlighted records where the value of Commission is greater than the value of Amount * 0.02.

Creating Transformations

This chapter shows how to create transformations in Artix Data Services Designer. Transformations are created within projects and consist of at least two data models that represent input and output data. They allow users to map elements in the input model to elements in the output model for the purposes of transforming your data in some way. A transformation may consist of multiple input and output models. This chapter first describes how to create a simple transformation and then describes how to make it more complex by adding various types of components to it.

In this chapter

This chapter discusses the following topics:

Creating a Simple Transformation	page 68
Making Your Transformation More Complex	page 81

Creating a Simple Transformation

Overview

This section is designed to get you started with creating a simple transformation called StatGen.tfd. The transformation will contain one input model called Transactions and one output model called Statements. Its purpose is to read in a series of Customer Details records and to produce statement lines for various customers. After creating the simple transformation, you can run it in the Run Wizard to test its validity and generate Java class instances from it.

Note: This demonstration is illustrated by the video tutorial within the `Getting Started/Samples` and `Videos/Creating Transformations/Simple Transformation` folder of your Artix Data Services Getting Started material.

In this section

This section discusses the following topics:

Starting to Create a Transformation	page 69
Adding an Input Model	page 70
Adding an Output Model	page 71
Creating a Local Transformation	page 72
Testing the Local Transformation in Your Main Transformation	page 75
Creating a Filter	page 77
Testing the Filter in Your Main Transformation	page 79


Starting to Create a Transformation

Overview

This section describes how to start creating a transformation.

Steps

Follow these steps:

1. In the Project view of the workbench, ensure that `MyProject.iop` is opened and then navigate to the `My IONA Projects/Getting Started/Samples and Videos/Creating Transformations/Simple Transformation` folder
2. Right click the `Simple Transformation` folder and select **New File/Directory**. This opens the **New File/Directory** dialog.
3. Select **New Transform**. This opens the **Enter name for transform** dialog.
4. For the purposes of this example, the transformation is called `StatGen`, because its purpose will be to generate statements based on transaction details. Type "StatGen" in the available text box and click **OK**. This causes `StatGen.tfd` to be automatically created and displayed in the Project and Explorer views of the workbench. A `StatGen.tfd` tab is also automatically opened in the main view of the workbench, and a dialog box is displayed prompting you to set the target namespace.
5. Click the  icon in the dialog box to close it.


Adding an Input Model

Overview

The next step is to add the data model that you want to use as input for the transformation. For the purposes of this example, the Transactions data model will be added as your input.

Steps

Follow these steps:

1. In the **Inputs** section of the main view, click the  (New Global Input) icon. This opens the **Select New Input Data Model** dialog.
2. Navigate to `My IONA Projects/Getting Started/Samples and Videos/Creating Transformations`, select `Transactions.dod` and click **OK**. This opens the **Select New Input Type** dialog.
3. Expand **File**, select the **Transactions** complex type, and click **OK**. The Transactions data model is now added as your input for the transformation, and the Transactions complex type is displayed along with its Header and Customer Details elements in the Inputs section of the MAIN tab.


Adding an Output Model

Overview

The next step is to add the data model that you want to use as output for the transformation. For the purposes of this example, the Statements data model will be added as your output.

Steps

Follow these steps:

1. In the **Outputs** section of the main view, click the  (New Global Output) icon. This opens the **Select New Output Data Model** dialog.
2. Navigate to *My IONA Projects/Getting Started/Samples and Videos/Creating Transformations*, select *Statements.dod* and click **OK**. This opens the **Select New Output Type** dialog.
3. Select the *StatementFile* complex type and click **OK**. The *Statements* data model is now added as your output for the transformation, and the *StatementFile* complex type is displayed along with its *Statement* element in the *Outputs* section of the *MAIN* tab.

Creating a Local Transformation

Overview

A transformation is made functional by adding functions to it. This is done by creating a local transformation that is contained within the main transformation. The local transformation will represent an individual operation and encapsulates functionality that can be reused within the main transformation, to cause an iterative loop effect. Therefore, elements with a cardinality of more than 1 (that is, elements of which there can be multiple instances) must be mapped within a local transformation so that they can be handled correctly. Local transformations work in exactly the same way as other transformations. This section describes how to automatically add a local transformation called "Record to StmtLine" within your main StatGen transformation.

Automatically adding a local transformation


Follow these steps to automatically add a local transformation within your main transformation:

1. Click "Customer Details" in the Inputs section to highlight it.
2. Click "Customer Details" again and drag your mouse across to "StmtLine" in the Outputs section while holding the left mouse key.

Note: You might need to expand "Statement" in the Outputs section to display "StmtLine".

A Warning dialog is now displayed with the following text:

```
The translation requires a mapping between two different complex types. Would you like to create a local transform and proceed with the mapping?
```

3. Click **OK** to automatically create the local transformation. This creates a "CustomerDetails To StmtLine" local transformation which is automatically opened in a new tab (with a  icon beside its name) within the StatGen.tfd tab. The new local transformation has "Customer Details" as its input parameter and "StatementLine" as its output parameter.

Note: For the purposes of this example, rename the local transformation to "Record to StmtLine". To do this, click the MAIN tab, right-click the local transformation in the ALL section, select **Rename**, type "Record to StmtLine" and click **OK**. The new name is automatically reflected in the local transformation and its corresponding tab.

Mapping input "Name" to output "PostingNarrative"

In this case, you want the name in each Customer Details record to be displayed as a posting narrative in your output statements. You therefore need to map "Name" in your input model to "PostingNarrative" in your output model. To do this:

1. Click the Record to StmtLine tab to reopen it.
2. Click "Name" in the Inputs section to highlight it.
3. Click "Name" again and drag your mouse across to "PostingNarrative" while holding the left mouse key.

This displays an arrow going from "Name" to "PostingNarrative". This arrow is an indicator that there is now a mapping between these two elements.

Mapping input "Amount" to output "TxAmount"

In this case, you also want the amount in each Customer Details record to be displayed as a transaction amount in your output statements. You therefore need to map "Amount" in your input model to "TxAmount" in your output model. To do this:

1. Try to connect "Amount" in the Inputs section to "TxAmount" in the Outputs section, again by clicking "Amount" in the Inputs section and dragging your mouse across to "TxAmount" while holding the left mouse key. In this case, you receive the following message:

The translation requires a narrowing of the valid range of numbers. Would you like to create a CAST function and proceed with the mapping?

This message indicates that you cannot set up a straightforward mapping between "Amount" and "TxAmount" because they are not of the same type—one is a double and the other is a float.

Note: The reason why you could set up a direct mapping between "Name" and "PostingNarrative" is because they are both strings.

2. Click **OK** to indicate that you want a CAST function to be automatically created to force a compatible mapping between the "Amount" double type and the "TxAmount" float type.

The CAST function is automatically displayed in the ALL section of the Record to StmtLine tab, with "Amount" in the Inputs section connected to "Arg1" in the CAST function, and "Result" in the CAST function connected to "TxAmount" in the Outputs section.

Testing the Local Transformation in Your Main Transformation


Overview

Now that you have set up a local transformation and its associated functions and mappings, you can check to see how it has made your main transformation more functional.

Running the transformation

You are now ready to run the transformation to see the potential results it will produce. To do this:

1. Click the MAIN tab and you will see that the "Record to StmtLine" local transformation is now displayed in the ALL section, with "Customer Details" in the Inputs section connected to "Customer Details" in the local transformation, and "StatementLine" in the local transformation connected to "StmtLine" in the Outputs section.
2. Right-click StatGen.tfd in the Explorer window and select **Run Component**. This opens the Run Wizard dialog.
3. In this case, the **Name** field automatically defaults to "StatGen" (that is, the name of the selected component) and the **Target** field defaults to the path location of the selected transformation. The **Build Before Running** check box is checked by default.
4. For the purposes of this demonstration, accept all the default values on the Run Wizard dialog and click **Run**.

This opens a Run tab (with a  icon beside its name) within the StatGen.tfd tab. This tab will be used to show the results from running your transformation. In this case, the data that you previously loaded into your Transactions data model is now reloaded.

Expand "Transactions" in the Inputs section to view the various Customer Details records that form your input. Expand "Statement" in the Outputs section to view the various StmtLine records that form your output. Notice how for each Customer Details record there is a corresponding StmtLine record containing both transaction amount and posting narrative details. This proves that your local transformation is working correctly, because it has produced the expected results.

Note: Errors relating to the validation rules on the Transactions input model will be displayed. You do not need to address these errors at this time. Just be aware that they will be displayed.


Creating a Filter

Overview

Suppose that you want to produce statement lines for only one particular customer rather than all customers. In this case, you can add a filter to your transformation to filter out any Customer Details records that you are not interested in. For the purposes of this example, let's assume that you now only want to produce statement lines for the customer Mr. Scrooge.


Starting to create a filter

Follow these steps to start creating a filter within your main transformation:

1. Click the Design tab and then click the MAIN tab to reopen the transformation.
 1. Right click the ALL section in the MAIN tab and select **New Component**. This opens the **New Component** dialog.
 2. Select **New Filter**. This opens the **New Filter** dialog.
 3. Type "JustScrooge" in the text box and click **OK**. This opens a **JustScrooge** tab (with a  icon beside its name) within the StatGen.tfd tab.
-

Adding your local input model

The next step is to add the data model that you want to use as input for the filter. For the purposes of this example, the Customer Details complex type in the Transactions data model will be added as local input. Follow these steps to add your local input model:

1. In the **Inputs** section of the JustScrooge tab, click the  (New Local Input) icon (Alternatively, right click in the **ALL** section, select **New Input/Output**, and then select **New Local Input**.) This opens the **Select New Input Path** dialog.
2. Select "Customer Details" and click **OK**. This displays the Customer Details complex type and its elements in the Inputs section of the JustScrooge tab.

Notice in this case that the Outputs section is divided into a **Condition** pane and a **Value** pane. The purpose of these will be shown in a minute.

Note: You cannot add output models to filters.

Adding the EQUALS function to your local transformation

In this case, you need to specify the logic of the filter that you want to implement. To do this, you can use a logic function called EQUALS. Follow these steps to add the EQUALS function to your local transformation:

1. In the ALL section in the JustScrooge tab, right click and select **New Component**. This opens the **New component** dialog.
2. Select **New Function**. This opens the **New Function** dialog.
3. Expand **Logic**, select **EQUALS** and click **OK**. The EQUALS function is now displayed in the ALL section.
4. Connect "Name" in the Inputs section to "Arg1" in the EQUALS function. This displays an arrow going from "Name" to "Arg1", and Arg1 is now displayed in black.
5. Right-click "Arg2" in the EQUALS function and select **Set Constant Value** from the context menu. This opens the **Set Constant Value** dialog.
6. Type "Mr Scrooge" in the text box and click **OK**. Mr Scrooge is now displayed in the ALL section as a constant value for Arg2.
7. Connect "Result" in the EQUALS function to "boolean" in the Condition part of the Outputs section. This displays an arrow going from "Result" to "boolean", and Result is now displayed in black.
8. Click "Customer Details" in the Inputs section and connect it to "any" in the Value part of the Outputs section. This displays an arrow going from "Customer Details" to "any".

Testing the Filter in Your Main Transformation

Overview

Now that you have set up a filter and its associated functions and mappings, you can check to see what difference it makes to your transformation.

Mapping main inputs and outputs

When you set up a filter, it will be displayed in the ALL section of your main transformation. Click the MAIN tab and you will see that the "JustScrooge" filter is now displayed in the ALL section, with "Customer Details" as its input parameter and "Value" as its output parameter.

Note: You can move components around and change their position in the ALL section if you wish. Simply click the name of a component in the ALL section and drag your mouse while holding the left mouse key. That component will then move position accordingly.

The next step is to set up mappings between inputs and outputs in your main transformation. To do this:

1. Connect "Customer Details" in the Inputs section to "Customer Details" in the JustScrooge filter. This displays an arrow going from "Customer Details" in the Inputs section to "Customer Details" in the filter, and Customer Details in the filter is now displayed in black.
2. Connect "Value" in the JustScrooge filter to "Customer Details" in the Transaction to Statement local transformation. This displays an arrow going from "Value" to "Customer Details" in the local transformation, and Value is now displayed in black.

Note: Notice how Artix Data Services Designer automatically deletes the now redundant arrow between "Customer Details" in the Inputs section and "Customer Details" in the Record to StmtLine local transformation.

Running the transformation

Now that you modified the mappings between your inputs and outputs, you can run your transformation to see the potential results these modifications will produce. To do this:

1. Right-click StatGen.tfd in the Explorer window and select **Run Component**. This opens the Run Wizard dialog.
2. In this case, the **Name** field automatically defaults to "StatGen" (that is, the name of the selected component) and the **Target** field defaults to the path location of the selected transformation. The **Build Before Running** check box is checked by default.
3. For the purposes of this demonstration, accept all the default values on the Run Wizard dialog and click **Run**.

This opens a Run tab (with a ▶ icon beside its name) within the StatGen.tfd tab. This tab will be used to show the results from running your transformation. Expand "Transactions" in the Inputs section to view the various Customer Details records that form your input. Expand "Statement" in the Outputs section to view the various StmtLine records that form your output. In this case, notice how only two StmtLine records have been produced in your output, both of them relating to the two Customer Details records for Mr Scrooge. No StmtLine records have been produced for any other customer. This proves that your newly added filter is working correctly, because it has produced the expected results.

You have now successfully created a simple transformation that includes both a local transformation and a filter with associated functions and mappings. Next let's look at how you can make your transformation more complex by adding more models and components to it.

Making Your Transformation More Complex

Overview

This section expands on what you learned in the previous section. It shows how you can make your transformation more complex by adding various other components to it.

In this section

This section discusses the following topics:

Before You Continue	page 82
Adding More Input Models to Your Main Transformation	page 84
Adding Local Transformations	page 86
Adding Functions	page 90
Adding Nested Local Transformations	page 95
Adding Hash Tables	page 101
Adding Filters	page 104
Adding Java Methods	page 111
Adding Introspect Functions	page 114

Before You Continue

Overview

There are some features and components in the simple transformation you have just created that are not relevant to the more complex example. To make your transformation suitable for continuing with the complex example, you need to make various adjustments to the transformation as outlined next. These modifications are a good way of showing you how you can modify a transformation.

Delete the JustScrooge filter

The JustScrooge filter is not a relevant feature of the more complex demonstration. Please make sure that you delete the JustScrooge filter now as follows:

1. Click the MAIN tab.
 2. Right-click the JustScrooge filter and select **Delete**. This opens a **Confirm Delete** dialog.
 3. Click **OK** to confirm that you want to delete the filter. This opens a **Confirm Component Delete** dialog.
 4. Click **Yes** to confirm that you want to delete the filter. The filter and its associated mappings are then automatically deleted from the MAIN tab.
-

Delete the CAST function

The CAST function is not a relevant feature of the Record to StmtLine local transformation in the more complex demonstration. Please make sure that you delete the CAST function from the Record to StmtLine local transformation as follows:

1. Click the Record to StmtLine tab.
1. Right-click the CAST function and select **Delete**. This opens a **Confirm Delete** dialog.
2. Click **OK** to confirm that you want to delete the function. The function and its associated mappings are then automatically deleted from the Record to StmtLine tab.

Delete the mapping between Name and PostingNarrative

The mapping between Name and PostingNarrative is not a relevant feature of the Record to StmtLine local transformation in the more complex demonstration. Please make sure that you delete the mapping between Name and PostingNarrative from the Record to StmtLine local transformation as follows:

1. Click the Record to StmtLine tab.
 1. Right-click the mapping between Name and PostingNarrative, and select **Delete**. This opens a **Confirm Delete** dialog.
 2. Click **OK** to confirm that you want to delete the mapping. The connection between Name and PostingNarrative is then automatically deleted from the Record to StmtLine tab.
-

Move the Record to StmtLine local transformation

Another modification that is required for the complex demonstration is to move the location of the Record to StmtLine local transformation within the main transformation. However, you are not ready to do this just yet. Instructions on how to do this will be provided in due course.

Adding More Input Models to Your Main Transformation

Overview



The main StatGen transformation already contains one input model called Transactions. Now let's start making it more complex by adding two more input models to it—Customers and Accounts.

Note: Because this section only shows how to create a basic transformation and defines the input and output models without defining any translations, it does not show how to open the Run Wizard. That will be shown in a subsequent section.

Note: Before you continue, ensure that you have created all data models as instructed in chapter 2 of this guide.

Steps

Follow these steps to add the additional input models:

1. Click the MAIN tab.
1. In the **Inputs** section, click the  (New Global Input) icon. This opens the **Select New Input Data Model** dialog.
2. Navigate to *My IONA Projects/Getting Started/Samples and Videos/Creating Transformations*, select *Customers.dod* and click **OK**. This opens the **Select New Input Type** dialog.
3. Select the Customers File complex type and click **OK**. The Customers data model is now added as part of your input for the transformation, and the Customer File complex type is displayed along with its Customer element in the Inputs section of the MAIN tab.
4. In the **Inputs** section, click the  (New Global Input) icon. This opens the **Select New Input Data Model** dialog.
5. Navigate to *My IONA Projects/Getting Started/Samples and Videos/Creating Transformations*, select *Accounts.dod* and click **OK**. This opens the **Select New Input Type** dialog.
6. Select the Accounts File complex type and click **OK**. The Accounts data model is now added as part of your input for the transformation, and the Accounts File complex type is displayed along with its Account element in the Inputs section of the MAIN tab.

You now have three input models and one output model in your transformation. However, the transformation as it stands is not very functional, so the next step is to add a new local transformation to it. See [“Running the transformation” on page 88](#) for more details.

Adding Local Transformations

Overview

The simple demonstration has already shown how to create a local transformation called "Record to StmtLine". For the purposes of this more complex demonstration, you now need to create another local transformation called "AccountTxns to Statement".

Automatically adding the new local transformation


Follow these steps to automatically add the new local transformation within your main transformation:

1. Click the MAIN tab.
2. Connect "Account" (under Accounts File) in the Inputs section to "Statement" in the Outputs section.

A Warning dialog is now displayed with the following text:

```
The translation requires a mapping between two different
complex types. Would you like to create a local transform
and proceed with the mapping?
```



3. Click **OK** to automatically create the local transformation.

This creates an "Account To Statement" local transformation which is automatically opened in a new tab (with a  icon beside its name) within the StatGen.tfd tab. The new local transformation has "Account" as its input parameter and "Statement" as its output parameter.

Note: For the purposes of this example, rename the new local transformation to "AccountTxns to Statement". To do this, click the MAIN tab, right-click the "Account To Statement" local transformation in the ALL section, select **Rename**, type "AccountTxns to Statement" and click **OK**. The new name is automatically reflected in the local transformation and its corresponding tab.

Adding more input models to the new local transformation

For the purposes of this example, two more input models now need to be added to the AccountTxns to Statement local transformation, as follows:

1. Click the AccountTxns to Statement tab to open it.
2. In the Inputs section, click the  (New Local Input) icon (Alternatively, right click in the **ALL** section, select **New Input/Output**, and then select **New Local Input**.) This opens the **Add input** dialog with a list of existing input models.
3. Select **Transactions** and click **OK**. This opens the **Select New Input Path** dialog.
4. Select **Transactions** and click **OK**. This displays the Transactions complex type along with its Header and Customer Details elements in the Inputs section of the AccountTxns to Statement tab.
5. In the Inputs section, click the  (New Local Input) icon (Alternatively, right click in the **ALL** section, select **New Input/Output**, and then select **New Local Input**.) This opens the **Add input** dialog with a list of existing input models.
6. Select **Customers File** and click **OK**. This opens the **Select New Input Path** dialog.
7. Select **Customers File** and click **OK**. This displays the Customers File complex type along with its Customer element in the Inputs section of the AccountTxns to Statement tab.

Setting up main mappings to the new local transformation

When a local transformation contains only one input and output model, Artix Data Services Designer automatically handles the mapping between inputs and outputs for you in the MAIN tab. However, when you add additional input or output models to a local transformation, you must manually set up the additional mappings yourself. For the purposes of this example:

1. Click the MAIN tab.
2. Connect "Transactions" in the Inputs section to "Transactions" in the AccountTxns to Statement local transformation. This displays a second arrow going from the Inputs section to the new local transformation, and Transactions in the local transformation is now displayed in black.

Note: Function parameters are displayed in red to warn you that they have no associated mapping. When you establish a mapping for a function parameter, it is then displayed in black.

3. Connect "Customers File" in the Inputs section to "Customers File" in the Transactions to Statement local transformation. This displays a third arrow going from the Inputs section to the new local transformation, and Customers File in the local transformation is now displayed in black.
4. Select **File > Save> > Save Tab As** and navigate to the `My IONA Projects/Getting Started/Samples and Videos/Creating Transformations/Adding Local Transformations` folder
5. Click **File > Save** from the menu bar or click **Save** to save your changes to the `StatGen.tfd` file.

Running the transformation

Now try running your transformation to see how the elements in the input models translate to elements in the output model. To do this:

1. Right-click `StatGen.tfd` in the Explorer window and select **Run Component**. This opens the Run Wizard dialog.
2. In this case, the **Name** field automatically defaults to "StatGen" (that is, the name of the selected component) and the **Target** field defaults to the path location of the selected transformation. The **Build Before Running** check box is checked by default.

3. For the purposes of this demonstration, accept all the default values on the Run Wizard dialog and click **Run**.

This opens a Run tab (with a ▶ icon beside its name) within the StatGen.tfd tab. This tab will be used to show the results from running your transformation. In this case, the data that you previously loaded into your Transactions, Customers, and Accounts data models is now reloaded. Expand the various elements to view the various records that form your input. Notice how the output Statement model is empty in this case. This is because no functions or translations currently exist within the transformation, so effectively there is nothing yet to be transformed.

Note: Errors relating to the validation rules on the Transactions input model will be displayed. You do not need to address these errors at this time. Just be aware that they will be displayed.

At this point, your transformation is not very functional, so you need to add some functions to it. Let's look at doing this next.

Adding Functions

Overview

Transformations are built up from functions that are chained together to convert one or more values from the input model to a node in the output model. The elements in an input model are translated to that of the output model. These elements are not always compatible and must therefore be "cast" or modified by the use of functions to ensure compatibility.

The purpose of this demonstration is to show how you can use NOW and CONVERTDATE functions to determine the statement date node in the output model. For the purposes of this demonstration, the CONVERTDATE function will be used to translate the generic date that is derived from the NOW function to the ISO8601 statement date node in the output model.

Note: The transformation created in this section is only partially complete, so the transformed statement will be invalid. However, you should look out for the stmtDate node which uses the function at this stage.

Note: Before you continue, ensure that you have completed the instructions in ["Running the transformation" on page 88](#).

Starting to create functions

Follow these steps to start creating functions within your existing transformation:

1. In the Project view of the workbench, ensure that `MyProject.iop` is opened and then navigate to the `My IONA Projects/Getting Started/Samples and Videos/Creating Transformations/Adding Local Transformations` folder
2. Right click the `StatGen.tfd` file and select **Open Selected**. This opens the `StatGen.tfd` transformation in the main view of the workbench.
3. Go to the Transaction to Statement local transformation. The Transaction to Statement local transformation is displayed with no mappings or functions.

Creating NOW and CONVERTDATE functions

First, create an operation to assign the current date to the statement date. Start by creating a date function called NOW. Follow these steps:

1. Click the AccountTxns to Statement tab.
2. In the ALL section, right-click and select **New Component**. This opens the **New component** dialog.
3. Select **New Function**. This opens the **New Function** dialog.
4. Expand **Date & Time**, select **NOW** and click **OK**. The NOW function is displayed in the ALL section.
5. Try to connect "Result" in the NOW function to "StmtDate" in the Outputs section. This displays the following message:

The translation requires a change to the type of date. Would you like to create a CONVERTDATE function and proceed with the mapping?

This message indicates that the NOW function returns a Generic date that is incompatible with the StmtDate type, and is prompting you to automatically create a CONVERTDATE function that will convert the date derived from the NOW function to the correct type.

Note: In this case, the StmtDate is an ISO8601 type of date.

6. Click **OK** to indicate that you want the CONVERTDATE function to be automatically created.

This automatically creates the CONVERTDATE function and displays it in the ALL section of the AccountTxns to Statement tab, with "Result" in the NOW function connected to "Arg1" in the CONVERTDATE function, and "Result" in the CONVERTDATE function connected to "StmtDate" in the Outputs section.

This ensures that the correct ISO8601 type will be returned as the statement date.

Creating the ADD function

Next create an operation to map the LastStatementNo in the Account input model to the StmtNo in the Statement output model, but to increment it by 1 in the process. Start by creating a mathematical function called ADD, which will have the LastStatementNo as its first argument and a constant value of 1 as its second argument. Follow these steps:

1. Click the AccountTxns to Statement tab.
2. In the ALL section, right click and select **New Component**. This opens the **New component** dialog.
3. Select **New Function**. This opens the **New Function** dialog.
4. Expand **Math**, then expand **Arithmetic**, select **ADD** and click **OK**. The ADD function is displayed in the ALL section.
5. Connect "LastStatementNo" in the Account input model to "Arg1" in the ADD function. This displays an arrow going from "LastStatementNo" to "Arg1".
6. Right click "Arg2" in the ADD function and select **Set Constant Value**. This opens the **Set Constant Value** dialog.
7. Type "1" as the constant value and click **OK**. This sets Arg2 to a value of 1.
8. Try to connect "Result" in the ADD function to "StmtNo" in the Statement output model. This raises the following error:

The translation requires a narrowing of the valid range of numbers. Would you like to create a CAST function and proceed with the mapping?

This message indicates that the ADD function returns a number type that is incompatible with the StmtNo, and is prompting you to automatically create a CAST function that will convert the number derived from the ADD function to the correct type.

Note: In this case, the StmtNo is an integer type.

9. Click **OK** to indicate that you want the CAST function to be automatically created.

This automatically creates the CAST function and displays it in the ALL section of the AccountTxns to Statement tab, with "Result" in the ADD

function connected to "Arg1" in the CAST function, and "Result" in the CAST function connected to "StmtNo" in the Outputs section.

This ensures that the correct integer type will be returned as the statement number.


10. Select **File > Save> > Save Tab As** and navigate to the `My IONA Projects/Getting Started/Samples and Videos/Creating Transformations/Adding Functions` folder
11. Click **Save** to save your changes to the `StatGen.tfd` file.

This ensures that the correct integer type will be returned as the statement number.

Running the transformation

Now try running your transformation to see how the elements in the input models translate to elements in the output model. To do this:

1. Right-click `StatGen.tfd` in the Explorer window and select **Run Component**. This opens the Run Wizard dialog.
2. In this case, the **Name** field automatically defaults to "StatGen" (that is, the name of the selected component) and the **Target** field defaults to the path location of the selected transformation. The **Build Before Running** check box is checked by default.
3. For the purposes of this demonstration, accept all the default values on the Run Wizard dialog and click **Run**.

This opens a Run tab (with a  icon beside its name) within the `StatGen.tfd` tab. This tab will be used to show the results from running your transformation. In this case, the data that you previously loaded into your Transactions, Customers, and Accounts data models is now reloaded. Expand the various elements to view the various records that form your input. Notice how the output Statement model is empty in this case. This is because no functions or translations currently exist within the transformation, so effectively there is nothing yet to be transformed.

Note: Errors relating to the validation rules on the Transactions input model will be displayed. You do not need to address these errors at this time. Just be aware that they will be displayed.

You have now added various functions to successfully output the statement date and statement number. However, the transformation still needs further updating. Two more local transformations need to be created at this point, this time within the Transactions to Statement local transformation. So let's look at adding some nested local transformations next. See [“Adding Nested Local Transformations” on page 95](#) for more details.

Adding Nested Local Transformations

Overview

You can nest components within other components. For example, you can nest one or more local transformations within another local transformation. For the purposes of this demonstration, two more local transformations called "Populate NameAndAddress" and "Record to StmtLine" need to be added within the existing 'Transactions to Statement' local transformation.

Note: Remember, you have already created a Record to StmtLine local transformation as part of the simple demonstration. This now needs to be moved, so that it will become a nested local transformation under AccountTxns to Statement.

Moving the "Record to StmtLine" local transformation

Follow these steps to move the "Record to StmtLine" local transformation under "AccountTxns to Statement".

1. Click the MAIN tab.
2. In the ALL section, right-click the Record to StmtLine local transformation and select **Delete**. This opens a **Confirm Delete** dialog.
3. Click **OK**. This opens a **Confirm Component Delete** dialog.
4. Click **No** on the Confirm Component Delete dialog.
5. Add a transform reference to AccountTxns to Statement???

Adding functions to "Record to StmtLine"

Follow these steps to add functions to the "Record to StmtLine" local transformation:

1. Click the Record to StmtLine tab.
1. In the ALL section, right-click and select **New Component**. This opens the **New component** dialog.
2. Select **New Function**. This opens the **New Function** dialog.
3. Expand **Date & Time**, select **CONVERTDATE** and click **OK**. This opens the Select Return Type dialog.
4. Expand **Date & Time**, select **ISO8601 date**, and click **OK**. The CONVERTDATE function is now displayed in the ALL section of the Record to StmtLine tab.




5. Connect "Transaction Date" in the Inputs section to "Arg1" in the CONVERTDATE function. This displays an arrow going from "Transaction Date" to "Arg1", and Arg1 is now displayed in black.
6. Connect "Result" in the CONVERTDATE function to both "PostingDate" and "ValueDate" in the Outputs section. This displays arrows going from "Result" to both "PostingDate" and "ValueDate", and Result is now displayed in black.
7. In the ALL section of the Record to StmtLine tab, right click and select **New Component**. This opens the **New component** dialog.
8. Select **New Function**. This opens the **New Function** dialog.
9. Expand **Logic**, select **GREATERTHAN** and click **OK**. The GREATERTHAN function is now displayed in the ALL section of the Record to StmtLine tab.
10. Connect "Amount" in the Inputs section to "Arg1" in the GREATERTHAN function. This displays an arrow going from "Amount" to "Arg1", and Arg1 is now displayed in black.
11. Right-click "Arg2" in the GREATERTHAN function and select **Set Constant Value** from the context menu. This opens the **Set Constant Value** dialog.
12. Type "0" in the text box and click **OK**. 0 is now displayed in the ALL section as a constant value for Arg2.
13. In the ALL section of the Record to StmtLine tab, right click and select **New Component**. This opens the **New component** dialog.
14. Select **New Function**. This opens the **New Function** dialog.
15. Expand **Logic**, select **IF** and click **OK**. This opens the **Select Return Type** dialog where you can choose the type you want the IF function to return.
16. Expand **Text**, select **String** and click **OK**. The IF function is now displayed in the ALL section of the Record to StmtLine tab. The IF function is now set to return a string type.
17. Connect "Result" in the GREATERTHAN function to "Arg1" in the IF function. This displays an arrow going from "Result" to "Arg1" and "ValueDate", and both parameters are now displayed in black.

18. Right-click "Arg2" in the IF function and select **Set Constant Value** from the context menu. This opens the **Set Constant Value** dialog.
19. Type "DR" in the text box and click **OK**. DR is now displayed in the ALL section as a constant value for Arg2.
20. Right-click "Arg3" in the IF function and select **Set Constant Value** from the context menu. This opens the **Set Constant Value** dialog.
21. Type "CR" in the text box and click **OK**. CR is now displayed in the ALL section as a constant value for Arg3.
22. Connect "Result" in the IF function to "DrCr" in the Outputs section. This displays an arrow going from "Result" to "DrCr", and Result is now displayed in black.
23. Connect "Amount" in the Inputs section to "TxAmount" in the Outputs section. This displays an arrow going from "Amount" to "TxAmount".
24. Connect "Currency" in the Inputs section to the "Ccy" attribute of TxAmount in the Outputs section. This displays an arrow going from "Currency" to "Ccy".
25. Click the AccountTxns to Statement tab to open it.
26. Connect "Customer Details" in the Inputs section to "Customer Details" in the Record to StmtLine local transformation. This displays an arrow going from "Customer Details" In the Inputs section to "Customer Details" in the Record to StmtLine local transformation, and Customer Details in the local transformation is now displayed in black.
27. Connect "StatementLine" in the Record to StmtLine local transformation to "StmtLine" in the Outputs section. This displays an arrow going from "StatementLine" to "StmtLine", and StatementLine is now displayed in black.

Creating a "Populate NameAndAddress" local transformation

Follow these steps to create a "Populate NameAndAddress" local transformation under "AccountTxns to Statement":

1. Click the AccountTxns to Statement tab.
2. Right click the ALL section in the Transactions to Statement tab and select **New Component**. This opens the **New Component** dialog.
3. Select **New Local Transform**. This opens the **New Local Transform** dialog.

4. Type "Populate NameAndAddress" in the text box and click **OK**. This opens a **Populate NameAndAddress** tab (with a  icon beside its name) within the StatGen.tfd tab.
5. In the Inputs section of the Populate NameAndAddress tab, click the  (New Local Input) icon (Alternatively, right click in the ALL section, select **New Input/Output**, and then select **New Local Input**.) This opens the **Add Input** dialog.
6. Select "Customers File" and click **OK**. This opens the **Select New Input Path** dialog.
7. Select "Customer" and click **OK**. This displays the Customer complex type and its elements in the Inputs section of the Populate NameAndAddress tab.
8. In the Outputs section of the Populate NameAndAddress tab, click the  (New Local Output) icon (Alternatively, right click in the ALL section, select **New Input/Output**, and then select **New Local Output**.) This opens the **Select New Output Path** dialog.
9. Expand "Hdr", select "NameAddress", and click **OK**. This displays the PostalAddress1 complex type and its elements in the Outputs section of the Populate NameAndAddress tab.

Adding functions to "Populate NameAndAddress"

Follow these steps to add functions to the "Populate NameAndAddress" local transformation:

1. Click the Populate NameAndAddress tab.
1. In the ALL section, right-click and select **New Component**. This opens the **New component** dialog.
2. Select **New Function**. This opens the **New Function** dialog.
3. Expand **Collections**, select **UNION** and click **OK**. The UNION function is now displayed in the ALL section of the Populate NameAndAddress tab.
4. Connect "Customer Acronym" in the Inputs section to "Arg1" in the UNION function. This displays an arrow going from "Customer Acronym" to "Arg1", and Arg1 is now displayed in black.

5. Connect "addressLine" in the Inputs section to "Arg2" in the UNION function. This displays an arrow going from "addressLine" to "Arg2", and Arg2 is now displayed in black.
6. In the ALL section of the Populate NameAndAddress tab, right click and select **New Component**. This opens the **New component** dialog.
7. Select **New Function**. This opens the **New Function** dialog.
8. Expand **Collections**, select **SUBLIST** and click **OK**. The SUBLIST function is now displayed in the ALL section of the Populate NameAndAddress tab.
9. Connect "Result" in the UNION function to "Arg1" in the SUBLIST function. This displays an arrow going from "Result" to "Arg1", and both parameters are now displayed in black.
10. Right-click "Arg2" in the SUBLIST function and select **Set Constant Value** from the context menu. This opens the **Set Constant Value** dialog.
11. Type "0" in the text box and click **OK**. 0 is now displayed in the ALL section as a constant value for Arg2.
12. Right-click "Arg3" in the SUBLIST function and select **Set Constant Value** from the context menu. This opens the **Set Constant Value** dialog.
13. Type "5" in the text box and click **OK**. 5 is now displayed in the ALL section as a constant value for Arg2.
14. Connect "Result" in the SUBLIST function to "AdrLine" in the Outputs section. This displays an arrow going from "Result" to "AdrLine", and Result is now displayed in black.
15. Connect "Country Of Residence" in the Inputs section to "Ctry" in the Outputs section.
16. Click the Transactions to Statement tab to open it.
17. Connect "Customer" (under Customers File) in the Inputs section to "Customer" in the Populate NameAndAddress local transformation. This displays an arrow going from "Customer" in the Inputs section to "Customer" in the Populate NameAndAddress local transformation, and Customer in the local transformation is now displayed in black.

18. Connect "PostalAddress1" in the Populate NameAndAddress local transformation to "NameAddress" in the Outputs section. This displays an arrow going from "PostalAddress1" to "NameAddress", and PostalAddress1 is now displayed in black.

Next, let's look at adding a hash table to the transformation. See ["Adding Hash Tables" on page 101](#) for more details.

Adding Hash Tables

Overview

The hashtable function allows you to create a hash table of values that can be referenced by the transformation code. This is useful in cases where you want an input string value (for example, "USD") to act as key to an output string (for example, "US Dollar"), so the hash table operates as a simple set of one-to-one mappings. At deployment time, this structure is created as `java.util hashtable`.

The purpose of this demonstration is to show how you can use a currency hash table to assign names and values to different currencies. After the transformation is created, it is then deployed and its validity is tested by transforming a file from the input model to the output model in the Artix Data Services Runner


Note: The transformation created in this section is only partially complete, so the transformed statement will be invalid. However, you should look out for the currency node which uses the hash table at this stage.

Note: Before you continue, ensure that you have completed the instructions in [“Adding Functions” on page 90](#).

Creating a hash table in a transformation

Follow these steps to create a hash table in a transformation:

1. In the Project view of the workbench, ensure that `MyProject.iop` is opened and then navigate to the `My IONA Projects/Getting Started/Samples and Videos/Creating Transformations/Adding Functions` folder
2. Right click the `StatGen.tfd` file and select **Open Selected**. This opens the `StatGen.tfd` transformation in the main view of the workbench.
3. Click the `AccountTxns to Statement` tab.
4. In the ALL section, right click and select **New Component**. This opens the **New component** dialog.
5. Select **New Hashtable**. This opens the Hashtable dialog.

6. For the purposes of this example, the hash table is to be called Currencies. Type "Currencies" in the **Name** field of the **Settings** section.
7. For the purposes of this example, four different currency codes and their names are to be added to the hash table. Type the following inputs and output respectively (click  to add each new row):

Inputs	Outputs
EUR	Euro
GBP	British Pound
JPY	Japenese Yen
USD	US Dollar


The hash table now contains four rows of data.

8. Click **OK**. The Currencies hash table is displayed in the **ALL** section with an invalid Arg 1 and Result.
9. Now you need to specify the mappings between the input and output models. Connect "currency" in the Account input model to "Arg 1" of the Currencies hash table. This displays an arrow going from "currency" to "Arg 1".
10. Connect "Result" in the Currencies hash table to "Ccy" of Startbalance under the Hdr element and "Ccy" of EndBalance under the Tlr element in the Statement output model. This displays arrows going from "Result" to "Ccy" of both StartBalance and EndBalance.
11. Select **File > Save> > Save Tab As** and navigate to the My IONA Projects/Getting Started/Samples and Videos/Creating Transformations/Adding Hash Tables folder
12. Click **Save** to save your changes to the StatGen.tfd file.

Running the transformation

Now try running your transformation to see the effect of the hash table on the results produced. To do this:

1. Right-click StatGen.tfd in the Explorer window and select **Run Component**. This opens the Run Wizard dialog.
2. In this case, the **Name** field automatically defaults to "StatGen" (that is, the name of the selected component) and the **Target** field defaults to the path location of the selected transformation. The **Build Before Running** check box is checked by default.
3. For the purposes of this demonstration, accept all the default values on the Run Wizard dialog and click **Run**.

This opens a Run tab (with a  icon beside its name) within the StatGen.tfd tab. This tab will be used to show the results from running your transformation. In this case, the data that you previously loaded into your Transactions, Customers, and Accounts data models is now reloaded.

4. For the first record listed in the Outputs section, expand Statement, then expand Hdr, StartBalance, and click Ccy; Also for the first record listed in the Outputs section, expand Statement, then expand Tlr, EndBalance, and click Ccy.
5. Select the AccountsFile tab in the Inputs section and expand the first Account. In this case, notice how the "GBP" in the Inputs section maps to two instances of "British Pound" in the Outputs section.
6. For the second record listed in the Outputs section, expand Statement, then expand Hdr, StartBalance, and click Ccy. Also for the second record listed in the Outputs section, expand Statement, then expand Tlr, EndBalance, and click Ccy.
7. Select the AccountsFile tab in the Inputs section and expand the second Account. In this case, notice how the "USD" in the Inputs section maps to two instances of "US Dollar" in the Outputs section.

You have now added a hash table to successfully output the currency name of input currency codes. However, the transformation still needs further updating. Next, let's add a filter that will allow records to be extracted in the transaction file, using the credit card numbers that match the credit card numbers in the accounts file. See ["Adding Filters" on page 104](#) for more details.

Adding Filters

Overview

Filters are used to create mappings for recurring elements, so that only a subset of a group of recurring elements is returned as part of the transformation. A filter will first examine the two fields on which a comparison is based, discard the differences between them, perform the comparison, and return a subset that contains the matching records. The filter does this recursively. In Artix Data Services filters, the Inputs section expects a data model on which the filter logic can operate. The Outputs section is divided in two—the top section is the boolean logic which must be true, and the bottom section specifies what the output should be.

This section describes how to create two different filters within the AccountTxns to Statement local transformation. A "SameAccount" filter will be created to get the records in the transaction file that match the credit card numbers in the accounts file. (The credit card format is different between the accounts file and the transaction file, so it needs to be modified before a comparison is made.) A "FindCustomerRecord" filter will be created to

Note: The transformation created in this section is only partially complete, so the transformed statement will be invalid.

Note: Before you continue, ensure that you have completed the instructions in [“Adding Hash Tables” on page 101](#).

Creating the SameAccount filter

Follow these steps to create the SameAccount filter:

1. Click the AccountTxns to Statement tab.
2. In the ALL section, right click and select **New Component**. This opens the New component dialog.
3. Select **New Filter**. This opens the New Filter dialog.
4. Type "SameAccount" and click **OK**. The SameAccount filter is created as a new tab with Inputs, ALL, and Outputs sections. Please briefly read the instructions in the Inputs panel.

Next select the model that contains the first element to be involved in the comparison. For the purposes of this comparison select the Transactions model, as follows:

1. In the Inputs section, click the **New Local Input** icon (Alternatively, right click in the **ALL** section, select **New Input/Output**, and then select **New Local Input**.) This opens the **Add input** dialog with a list of existing input models.
2. Select **Transactions** and click **OK**. This opens the **Select New Input Path** dialog.
3. Select **Customer Details** and click **OK**. This displays the CustomerDetails complex type in the Inputs section of the SameAccount filter.

Next select the model that contains the second element to be involved in the comparison. For the purposes of this comparison select the Accounts model, as follows:

1. In the Inputs section, click the **New Local Input** icon (Alternatively, right click in the **ALL** section, select **New Input/Output**, and then select **New Local Input**.) This opens the **Add input** dialog with a list of existing input models.
2. Select **Account** and click **OK**. This displays the Account complex type in the Inputs section of the SameAccount filter.

In the Transactions model, the card numbers include hyphens between the numbers. In the Accounts model, the card numbers do not include any hyphens or spaces. Because the card numbers are represented differently between the two models, the elements need to be stripped of anything but numbers so that it will be possible to successfully compare them and continue filtering records. To do this, use a text function called REPLACEALL. Follow these steps to create the REPLACEALL function:

1. In the ALL section, right click and select **New Component**. This opens the **New component** dialog.
2. Select **New Function**. This opens the **New Function** dialog.
3. Expand **Text**, select **REPLACEALL** and click **OK**.
4. Connect "Card Number" in the Customer Details input model to "Arg1" in the REPLACEALL function. This displays an arrow going from "Card Number" to "Arg1".

5. The next step is to set as a constant value what it is you want to be replaced, which in this case is a hyphen. Right click "Arg 2" of REPLACEALL and select **Set Constant Value**. This opens the **Set constant value** dialog.
6. Type "-" and click **OK**. This causes "-" to be displayed as Arg 2.
7. The next step is to set as a constant value what it is you want to replace the hyphen with, which in this case is an empty string. Right click "Arg 3" of REPLACEALL and select **Set Constant Value**. This opens the **Set constant value** dialog.
8. Type "" and click **OK**. This causes "" to be displayed as Arg 3.

Now that the format of the comparable elements has been made to match, you may proceed with enabling the comparison. To do this, use a logic function called EQUALS. Follow these steps to create the EQUALS function:

1. In the ALL section, right click and select **New Component**. This opens the **New component** dialog.
2. Select **New Function**. This opens the **New Function** dialog.
3. Expand **Logic**, select **EQUALS** and click **OK**. The EQUALS function is displayed in the ALL section.
4. Connect "Result" in the REPLACEALL function to "Arg1" in the EQUALS function. This displays an arrow going from "Result" to "Arg1".
5. Connect "cardNo" in the Account input model to "Arg2" in the EQUALS function. This displays an arrow going from "cardNo" to "Arg2".
6. The result of the EQUALS function is the condition on which the filter is based. Connect "Result" in the EQUALS function to the boolean element in the Condition output. This displays an arrow going from "Result" to the boolean element.
7. If the condition is met, that transaction record will be stored in the any element of Value Output. Connect "Result" in the REPLACEALL function to the any element in Value Output. This displays an arrow going from "Result" to the any element.

99% of the filter is now complete. The remaining 1% needs to be completed in the AccountTxns to Statement local transformation where you must map elements of the SameAccount filter to elements of the Transaction input model, the Account input model, and Record to StmtLine local transformation, as follows:

Note: The filter represents an individual statement line in the statement model.

1. Click the AccountTxns to Statement tab.
2. Connect "Customer Details" (under Transactions) in the Inputs section to "Row" in the SameAccount filter. This displays an arrow going from "Customer Details" to "Row".
3. Connect "Account" in the Inputs section to "Account" in the SameAccount filter. This displays an arrow going from the Account input model to "Account" in the SameAccount filter.
4. Connect "Value" in the SameAccount filter to "Row" in the Record to StmtLine local transformation. This displays an arrow going from "Value" to "Row".
5. Select **File > Save> > Save Tab As** and navigate to the `My IONA Projects/Getting Started/Samples and Videos/Creating Transformations/Adding Filters` folder.
6. Click **Save** to save your changes to the `StatGen.tfd` file.

Creating the FindCustomerRecord filter

Follow these steps to create the FindCustomerRecord filter:

1. Click the AccountTxns to Statement tab.
2. In the ALL section, right click and select **New Component**. This opens the New component dialog.
3. Select **New Filter**. This opens the New Filter dialog.
4. Type "FindCustomerRecord" and click **OK**. The FindCustomerRecord filter is created as a new tab with Inputs, ALL, and Outputs sections. Please briefly read the instructions in the Inputs panel.

Next select the model that contains the first element to be involved in the comparison. For the purposes of this comparison select the Transactions model, as follows:

1. In the Inputs section, click the **New Local Input** icon (Alternatively, right click in the **ALL** section, select **New Input/Output**, and then select **New Local Input**.) This opens the **Add input** dialog with a list of existing input models.
2. Select **Transactions** and click **OK**. This opens the **Select New Input Path** dialog.
3. Select **Customer Details** and click **OK**. This displays the CustomerDetails complex type in the Inputs section of the SameAccount filter.

Next select the model that contains the second element to be involved in the comparison. For the purposes of this comparison select the Accounts model, as follows:

1. In the Inputs section, click the **New Local Input** icon (Alternatively, right click in the **ALL** section, select **New Input/Output**, and then select **New Local Input**.) This opens the **Add input** dialog with a list of existing input models.
2. Select **Account** and click **OK**. This displays the Account complex type in the Inputs section of the SameAccount filter.

In the Transactions model, the card numbers include hyphens between the numbers. In the Accounts model, the card numbers do not include any hyphens or spaces. Because the card numbers are represented differently between the two models, the elements need to be stripped of anything but

numbers so that it will be possible to successfully compare them and continue filtering records. To do this, use a text function called REPLACEALL. Follow these steps to create the REPLACEALL function:

1. In the ALL section, right click and select **New Component**. This opens the **New component** dialog.
2. Select **New Function**. This opens the **New Function** dialog.
3. Expand **Text**, select **REPLACEALL** and click **OK**.
4. Connect "Card Number" in the Customer Details input model to "Arg1" in the REPLACEALL function. This displays an arrow going from "Card Number" to "Arg1".
5. The next step is to set as a constant value what it is you want to be replaced, which in this case is a hyphen. Right click "Arg 2" of REPLACEALL and select **Set Constant Value**. This opens the **Set constant value** dialog.
6. Type "-" and click **OK**. This causes "-" to be displayed as Arg 2.
7. The next step is to set as a constant value what it is you want to replace the hyphen with, which in this case is an empty string. Right click "Arg 3" of REPLACEALL and select **Set Constant Value**. This opens the **Set constant value** dialog.
8. Type "" and click **OK**. This causes "" to be displayed as Arg 3.

Now that the format of the comparable elements has been made to match, you may proceed with enabling the comparison. To do this, use a logic function called EQUALS. Follow these steps to create the EQUALS function:

1. In the ALL section, right click and select **New Component**. This opens the **New component** dialog.
2. Select **New Function**. This opens the **New Function** dialog.
3. Expand **Logic**, select **EQUALS** and click **OK**. The EQUALS function is displayed in the ALL section.
4. Connect "Result" in the REPLACEALL function to "Arg1" in the EQUALS function. This displays an arrow going from "Result" to "Arg1".
5. Connect "cardNo" in the Account input model to "Arg2" in the EQUALS function. This displays an arrow going from "cardNo" to "Arg2".

6. The result of the EQUALS function is the condition on which the filter is based. Connect "Result" in the EQUALS function to the boolean element in the Condition output. This displays an arrow going from "Result" to the boolean element.
7. If the condition is met, that transaction record will be stored in the any element of Value Output. Connect "Result" in the REPLACEALL function to the any element in Value Output. This displays an arrow going from "Result" to the any element.

Running the transformation

Now try running your transformation to see the effect of the filter on the results produced. To do this:

1. Right-click StatGen.tfd in the Explorer window and select **Run Component**. This opens the Run Wizard dialog.
2. In this case, the **Name** field automatically defaults to "StatGen" (that is, the name of the selected component) and the **Target** field defaults to the path location of the selected transformation. The **Build Before Running** check box is checked by default.
3. For the purposes of this demonstration, accept all the default values on the Run Wizard dialog and click **Run**.
This opens a Run tab (with a ▶ icon beside its name) within the StatGen.tfd tab. This tab will be used to show the results from running your transformation. In this case, the data that you previously loaded into your Transactions, Customers, and Accounts data models is now reloaded.
4. It is invalid because some of the mandatory elements have not been mapped at this stage.)

Note: There is a second filter to be created in this case. It is a FindCustomerRecord filter for the Populate NameAndAddress local transformation. See the completed StatGen.tfd within *Getting Started/Samples and Videos/Creating Transformations/Adding Filters/Completed* for details of how to set up that filter.

Adding Java Methods

Overview

Java methods can be used to write new methods that will be embedded in the class representing the transformation in deployment time.

The purpose of this demonstration is to show how you can use a Java method to look up a transaction from a vendor and then assign it to a vendorID. The input parameter type is defined as "long", because the vendor ID that is passed in is of type "long". The return type is a string, so that it can be displayed as such in the output model.

Note: The transformation created in this section is only partially complete, so the transformed statement will be invalid.

Note: Before you continue, ensure that you have completed the instructions in [“Adding Filters” on page 104](#).

Steps

Follow these steps to use Java methods in a transformation:


1. In the Project view of the workbench, ensure that `MyProject.iop` is opened and then navigate to the `My IONA Projects/Getting Started/Samples` and `Videos/Creating Transformations/Adding Filters` folder
2. Right click the `StatGen.tfd` file and select **Open Selected**. This opens the `StatGen.tfd` transformation in the main view of the workbench.
3. In the `AccountTxns to Statement` local transformation, double click the `Record to StmtLine` local transformation.
4. Right click in the ALL section of the `Record to StmtLine` local transformation and select **New Component**. This opens the **New Components** dialog.
5. Select **New Java Method**. This opens the **Java Method** dialog.
6. In the Signature tab, under the Details section, type "CreateNarrative" in the **Method Name** field.
7. In the Parameters section, click + to add a new parameter row.
8. Type "vendorID" in the **Name** column.

9. Click anyType in the **Type** column. This opens the **Select Argument Type** dialog.
10. Expand **Numeric**, select **long** and click **OK**.
11. In the Return Type section, click **Select**. This opens the **Select Return Type** dialog.
12. Expand **Text**, select **String** and click **OK**.
13. Click the Code tab to open it. The method declaration is displayed.
14. Type in `' return "Transaction from vendor:"+vendorID;'` and click **OK**. The CreateNarrative method is displayed in the ALL section.
15. Connect "Vendor ID" in the Customer Details input model to "vendorID" in the CreateNarrative method. This displays an arrow going from "Vendor ID" to "vendorID".
16. Connect "Result" in the CreateNarrative method to "PostingNarrative" under the StatementLine element in the Statement output model. This displays an arrow going from "Result" to "PostingNarrative".
17. Select **File > Save> > Save Tab As** and navigate to the `My IONA Projects/Getting Started/Samples and Videos/Creating Transformations/Adding Java Methods` folder.
18. Click **Save** to save your changes to the `StatGen.tfd` file.

Running the transformation

Now run your transformation to see the effect of the Java method on the results produced. To do this:

1. Right-click `StatGen.tfd` in the Explorer window and select **Run Component**. This opens the Run Wizard dialog.
2. In this case, the **Name** field automatically defaults to "StatGen" (that is, the name of the selected component) and the **Target** field defaults to the path location of the selected transformation. The **Build Before Running** check box is checked by default.
3. For the purposes of this demonstration, accept all the default values on the Run Wizard dialog and click **Run**.

This opens a Run tab (with a  icon beside its name) within the `StatGen.tfd` tab. This tab will be used to show the results from running your transformation. In this case, the data that you previously loaded

into your Transactions, Customers, and Accounts data models is now reloaded.

4. Invalid StatementFile is displayed in the output section. (It is invalid because some of the mandatory elements have not been mapped at this stage.)
5. Expand **Statement** and then expand **StmtLine** for one or all records available. PostingNarrative should be displayed for that record.

Adding Introspect Functions

Overview

This section describes how to use introspect functions in transformations. Introspect functions return a value of the part of a complex type value which a user can then map to an output data model.

The purpose of this demonstration is to show how you can use an introspect function to extract country of residence from the Customer model, and concatenate it with an account number to identify the location of a customers account.

Note: The transformation created in this section is only partially complete, so the transformed statement will be invalid. However, you should look out for the Account node which uses the introspect function at this stage.

Note: Before you continue, ensure that you have completed the instructions in [“Adding Java Methods” on page 111](#).

Steps

Follow these steps to use filters in a transformation:

1. In the Project view of the workbench, ensure that `MyProject.iop` is opened and then navigate to the `My IONA Projects/Getting Started/Samples and Videos/Creating Transformations/Adding Introspect Functions` folder
2. Right click the `StatGen.tfd` file and select **Open Selected**. This opens the `StatGen.tfd` transformation in the main view of the workbench.
3. In the Transactions to Statement local transformation, right click in the ALL section and select **New Component**. This opens the **New Components** dialog.
4. Select **New Introspector**. The Introspect function is displayed in the **ALL** section, with "Arg1" as its input and "Result" as output.

If you have not disabled tool tips, a tool tip is also displayed prompting you to first map the input type of the introspect function and to then double click on it in order to specify the return type which will then

enable you to map its output. The Introspect function is displayed with "Arg1" as its input and "Result" as output.

5. Connect "Value" in the FindCustomerRecord filter to "Arg 1" in the Introspect function. This displays an arrow going from "Value" to "Arg 1".
6. Double click on "Arg 1" of Introspector. This opens the **Select Path** dialog.
7. Select the Customer complex type and click **OK**. This displays "Customer" as Arg 1 of Introspector.
8. Double click on "Result" of Introspector. This opens the **Select Path** dialog.
9. Select the "countryofResidence" element and click **OK**. "CountryOfResidence" is now displayed as Result of Introspector.
10. Right click in the **ALL** section and select New Component. This opens the New Component dialog.
11. Select **New Function**, expand **Text**, select **CONCAT**, and click **OK**. The CONCAT function is displayed in the **ALL** section.
12. Connect "countryOfResidence" in Introspector to "Arg 1" in the CONCAT function. This displays an arrow going from "countryOfResidence" to "Arg 1".
13. Connect "accountNumber" in the Account input model to "Arg 2" in the CONCAT function. This displays an arrow going from "accountNumber" to "Arg 2".
14. Connect "Result" in the CONCAT function to "Account" in the Statement output model. This displays an arrow going from "Result" to "Account".
15. Select **File > Save > > Save Tab As** and navigate to the `My IONA Projects/Getting Started/Samples and Videos/Creating Transformations/Adding Introspect Functions` folder.
16. Click **Save** to save your changes to the `StatGen.tfd` file.

Running the transformation

Now try running your transformation to see the effect of the introspect function on the results produced. To do this:

1. Right-click StatGen.tfd in the Explorer window and select **Run Component**. This opens the Run Wizard dialog.
2. In this case, the **Name** field automatically defaults to "StatGen" (that is, the name of the selected component) and the **Target** field defaults to the path location of the selected transformation. The **Build Before Running** check box is checked by default.
3. For the purposes of this demonstration, accept all the default values on the Run Wizard dialog and click **Run**.

This opens a Run tab (with a ▶ icon beside its name) within the StatGen.tfd tab. This tab will be used to show the results from running your transformation. In this case, the data that you previously loaded into your Transactions, Customers, and Accounts data models is now reloaded.

4. Invalid StatementFile is displayed in the output section. (It is invalid because some of the mandatory elements have not been mapped at this stage.)
5. Expand **Statement** and then expand **Hdr** for one or all records available. Account should now be different from what it was before. It should have a 2-character country of residence code at the start.

Overview of ANT Tasks

A number of Apache ANT (<http://ant.apache.org/>) tasks specific to Artix Data Services are packaged within the `artix-ds-designerXXX.jar` file. These enable deployment and exports to be automated with an ANT script. This is useful where the build of Artix Data Services generated components are to be included within overall project builds, without any requirement to manually deploy the components from within the Artix Data Services Designer.

In this chapter

This chapter discusses the following topics:

Using the supplied ANT tasks	page 118
Deployment	page 118
Deployments directory	page 118

Using the supplied ANT tasks

To use these tasks, you will need to include task definitions such as the following at the top of your ANT file (where the classpath reference includes the `artix-ds-designerXXX.jar` and `artix-commonX.jar` files):

```
<taskdef name="deploy" classname="biz.c24.io.ant.DeployTask"
  classpathref="classpath" loaderref="java.lang.ClassLoader"/>
```

Note: The `loaderref` attribute is required for full compatibility with versions of Ant prior to 1.6.0.

Deployment

Regarding deployment, an Ant build file is used to construct individual build files for each deployment. The `build-template.xml` file is delivered with the toolkit. At deployment time, namespace-specific build files are constructed by replacing various placeholders with the specific values for the deployment. The following replacements will occur at deployment time:

- `@namespace@` is replaced by the namespace.
 - `@package@` is replaced by the deployment package.
 - `@directory@` is replaced by the deployment directory (the deployment package with `'.'` replaced by `'/'`).
 - `@date@` is replaced by the deployment date in the format `yy/MM/dd`.
 - `@time@` is replaced by the deployment time in the format `hh/mm/ss`.
 - `@javadoc.link@` is replaced by the `'build.javadoc.link'` property taken from the `system.properties` file.
 - `@cvshheader@` is replaced by the default CVS header.
-

Deployments directory

The directory named "Deployments" is the directory where data models and transformations are deployed to. Under this directory you can find all Ant build files, Java source code, compiled Java classes, and jar files created at deployment time. You can specify the location of this deployment directory by altering the profile settings of the Artix Data Services Designer.