



Artix™ Data Services

Getting Started

Version 3.8, September 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: September 27, 2008

Contents

Preface	5
Chapter 1 Creating Projects	7
Before You Begin	8
Starting Artix Data Services Designer	9
Downloading Sample Getting Started Data	10
Creating a Project with the Project Wizard	11
Creating a Project Manually	14
Chapter 2 Creating Data Models	17
Creating a Data Model from a Text File	19
Creating the Transactions Data Model from Transactions.txt	20
Creating the Customers Data Model from Customers.txt	28
Creating a Data Model from an XML Schema	34
Creating a Data Model from Other Sources	38
Creating a Data Model from a Set of XML Documents	39
Creating a Data Model from a Database	42
Creating a Data Model Manually	47
Creating the Accounts Data Model Manually	48
Creating the Customers Data Model Manually	60
Adding Validation Rules	69
Adding Validation Rules for Accounts Data Model	70
Adding Validation Rules for Transactions Data Model	74
Chapter 3 Creating Transformations	79
Creating a Simple Transformation	80
Starting to Create a Transformation	81
Creating a Local Transformation	83
Testing the Local Transformation in Your Main Transformation	86
Creating a Filter	88
Testing the Filter in Your Main Transformation	90

Making Your Transformation More Complex	92
Before You Continue	93
Adding More Input Models to Your Main Transformation	95
Adding Local Transformations	96
Adding Functions	99
Adding Nested Local Transformations	104
Adding Hash Tables	112
Adding Filters	116
Adding Java Methods	122
Adding Introspect Functions	125
Chapter 4 Overview of ANT Tasks	129

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:

Before You Begin	page 8
Creating a Project with the Project Wizard	page 11
Creating a Project Manually	page 14

Before You Begin

Overview

Before you start working through the demonstrations, this section provides details of how to start the Artix Data Services Designer and download the supplied Getting Started plug-in to your product installation.

In this section

This section discusses the following topics:

Starting Artix Data Services Designer	page 9
Downloading Sample Getting Started Data	page 10

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 the product, see the Artix Data Services *Installation Guide* for details of the installation steps.

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.

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 - Not Installed** link. This opens the **Confirm Download** dialog.
 2. Click **OK** to proceed with the download. You will then be prompted when the download has completed successfully.
-

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 folder contains the following subfolders:

- | | |
|----------|--|
| /Guide | This contains a PDF copy of this Getting Started guide. |
| /Samples | This contains a series of subfolders that correspond to the various chapters in this guide. Each subfolder 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 demonstrations, you will be prompted to work with particular sample files. |
| /Videos | This contains an HTML file with a link to various video tutorials that will help you to familiarise with using Artix Data Services. |

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 the 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 done so. 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 display or hide optional panels within the wizard. For the purposes of this demonstration, click the **Advanced** button to view the 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 **Next** to open the **Aliases** panel of the Project Wizard. This allows you to set up various preferred aliases that will enable you to choose between seeing different sets of names for the same components within your data models. By default, there is only one "[default]" alias defined. For the purposes of this demonstration, simply accept the default setting.
13. Click **Finish**. If you are prompted to open the project in a new frame, click **Yes**. (This prompt only appears if you have already created another project.)

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 done so. 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`, 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 only appears if you have already created another project.)

If it is displayed, uncheck the **Show Tips on startup** check box and click **Close** to cancel the **Tip Of The Day** dialog.

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, click **Default**, 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** on the **Profile Settings** panel to reopen the **Project Properties** panel.
11. Click the **Preferred Aliases** icon to set up various aliases that will enable you to choose between seeing different sets of names for the same components within your data models. By default, there is only one "[default]" alias defined. For the purposes of this demonstration, simply accept the default setting.
12. 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.

In this chapter

This chapter discusses the following topics:

Creating a Data Model from a Text File	page 19
Creating a Data Model from an XML Schema	page 34
Creating a Data Model from Other Sources	page 38
Creating a Data Model Manually	page 47

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 20
Creating the Customers Data Model from Customers.txt page 28

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 the model, you can test its accuracy 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/Creating Data Models/From a Text File folder of your Artix Data Services Getting Started material. This demonstration is illustrated by the Creating a Data Model from a Text File video tutorial which you can access from the Getting Started/Videos folder.

Steps

Follow these steps to start creating your data model:

1. In the Project window of the workbench, ensure that `MyProject.iop` is opened. If you need to open it, you may do so by selecting **File > Open Project** from the menu bar.
2. Navigate to the `My IONA Projects/Getting Started/Samples/Creating Data Models/From a Text File` folder
3. Right-click the `From a Text File` folder and select **Import > Import Text File**. This opens the **Import File** panel of the Text File Import Wizard.
4. Navigate to `Getting Started/Samples/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 `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 "Header" row and two "Row" rows. 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 1", type "Customer Details" as the value instead, and then press Enter. Then double click on "Row 2", type "Row Count" as the value instead, and press Enter again. Notice how steps 9 and 10 in the left-hand pane automatically change from "Row 1" and "Row 2" to "Customer Details" and "Row Count" respectively.
12. Click the **Type** column for Row Count and select "Fixed Length" instead.
13. Click **Next**. This opens the **Header** panel.
14. The Header record is a delimited format type 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.
15. 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 1" instead.

16. The Customer Details records are a delimited format type 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.
17. Click **Next** to open the **Row Count** panel.

Note: Notice how the panel name here, "Row Count", 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 2" instead.

18. The Row Count record is a fixed format type and this has been automatically picked up by the wizard. In the **Fixed Offset Properties** section, click the final column to automatically place a boundary between the "=" and "7". This causes a new column to be displayed in the **Preview - Column Data Types** section.
19. Click the first column in the **Preview - Column Data Types** section. Then type "Prefix" in the **Selected Column Name** field and press Enter. This causes the first column name to change to "Prefix".
20. Select "String" as the value in the **Selected Column Data Type** field.
21. Click the second column in the **Preview - Column Data Types** section. Then type "Value" in the **Selected Column Name** field and press Enter. This causes the second column name to change to "Value".
22. Select "Long" as the value in the **Selected Column Data Type** field.
23. 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.
In the Messages window, an **Importing Text File** tab is opened to indicate that the import has been successful.
24. Click `Transactions.dod` in the **Explorer** window. This causes the properties for the data model to be displayed in the Properties window.

25. In the **General** section of the Properties window, set the value for **Target Namespace** to
<http://www.iona.com/ArtixDataServices/GettingStarted/Transaction>.
26. Select **File > Save All** from the menu bar, or click the  icon on the toolbar, to save the data model.
27. 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, Customer Details, and Row Count elements to view the contents. Compare the details displayed with those in the Transactions.txt file that you imported.

Setting up a validation rule for the row count

Let's set up a validation rule that will determine whether the value of the Row Count record is equal to the number of Customer Details records. If it is not, a validation error should be raised. Follow these steps to set up this validation rule against the row count record:

1. Right-click `Transactions.dod` in the Explorer window and select **New > Validation Rule**. This opens the **New Validation Rule** dialog.
2. Type "rowCheckRule" in the text box and click **OK**. This automatically opens a rowCheckRule 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.

3. First, let's add the XPath syntax for the Value element to the XPath rule. To do this, click the Transactions tab to open it, expand Row Count, right-click Value in the **Component** column, and select **Copy XPath**. Then click the rowCheckRule tab to reopen it, click in the shaded text area in the left hand pane in the tab, and select **Edit>Paste** from the menu bar. This copies the XPath syntax for the Value element to the XPath rule.

4. Click at the end of the XPath rule for the purposes of ensuring the cursor is in the correct position for adding the next part of it.
5. Next let's check whether the Value element does not match the number of Customer Details records. To do this, scroll down in the right-hand pane and double click the != (Not Equal) operator to select it. This adds != to the XPath rule in the left-hand pane.
6. Click at the end of the XPath rule for the purposes of ensuring the cursor is in the correct position for adding the next part of it.
7. Next let's specify that we are dealing with a number count in this case. To do this, scroll up in the right-hand pane and double click the "number count(node-set)" function to select it. This adds count () to the XPath rule in the left-hand pane.
8. Next let's specify that we want to count the number of Customer Details records. To do this, click the Transactions tab to reopen it, right-click Customer Details in the Component column, and select **Copy XPath** from the context menu. Then click the rowCheckRule tab to reopen it, make sure that you click within the parentheses for the count() function in the left-hand pane, and select **Edit>Paste** from the menu bar. This copies the XPath syntax for the Customer Details element to the XPath rule.



Note: The XPath rule should now look as follows:

```
/Transactions/RowCount/Value!=count (Transactions/  
CustomerDetails)
```

9. If the validation rule is true, the data model should raise a validation error. Therefore, type "Invalid row count" in the **Error Message** pane.
10. Uncheck the Ignore Document Node check box, to enable the imported XPath syntax to be read successfully.
11. Select **File > Save All** from the menu bar, or click the  icon on the toolbar, to save the data model.

Applying the validation rule to the data model


Follow these steps to apply the validation rule to the data model:


1. Click the Transactions tab to reopen it in the main window of the workbench.
 2. In the **Type** column, click "Transactions". This displays the properties for the Transactions complex type in the Properties window.
 3. In the Properties window, scroll down to the **Validation** section and click the field beside **Validation Rules**. This opens a validation rules dialog.
 4. Click the  icon. This opens the **Add Validation Rule** dialog.
 5. Now apply the global rowCheckRule validation rule to the Transactions type. Expand "Local", select the "rowCheckRule" global validation rule, and click **OK**. This adds rowCheckRule to the validation rules dialog.
 6. Click **OK** to close the validation rules dialog. The **Validation Rules** field in the Properties window now displays "1".
 7. Select **File > Save All** from the menu bar, or click the  icon on the toolbar, to save the data model.
-

Testing the accuracy of your data model


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 Transactions.txt file into your Transactions data model, as follows:

1. Ensure that the Transactions.dod data model is open in the Explorer window.
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**.

A dialog box is displayed prompting you to load the data you want to parse. Click the  icon in the dialog box to close it.

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, notice that an empty Run Transactions tab has been created at this point.


5. Click the  (**Load**) icon in the Transactions tab in the main window. This opens the Select Input File/Directory dialog.
6. Navigate to the `Getting Started/Samples/Creating Data Models/From a Text File` folder and select `Transactions.txt`. Then click **Open**.
7. A Note dialog is displayed prompting you that files in subdirectories will be parsed by default. Click **OK** on this dialog.
8. A Confirm dialog is displayed prompting you that changing the URI will allow your data to be overwritten. Click **Yes** on this dialog.

In the case of this demonstration, there are no parsing errors, 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. You can now expand the Transactions node in the main window to view a Header record, seven CustomerDetails records, and a RowCount record.

At this point, the Run Transactions tab in the Messages window displays a message indicating that parsing has been successful.

Checking the validation rule against row count

Follow these steps to test the validation rule that you have set up against Row Count:

1. Click the Validation tab at the bottom of the workbench to open the Validation window. In this case, no validation errors are currently being reported. This is because the value of Row Count currently matches the number of Customer Details records loaded (that is, 7).
2. Expand Row Count in the Transactions tab (with a  icon beside its name) .

3. Change the value for the Value row to, for example, "5".
4. Click anywhere else in the tab and a validation error is now automatically reported in the Validation window.
5. Expand the validation error and it displays the "Invalid Row Count" error that has been set up.
6. Now change the value for Value back to "7" in the Transactions tab.
7. Click anywhere else in the tab and the validation error that was reported in the Validation window now automatically disappears.

This proves that the validation rule we have set up is working, because it is raising a validation error only when expected.

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 the model, you can test its accuracy by parsing a valid text file through it.

Note: You may skip this section if you are going to follow the instructions in [“Creating the Customers Data Model Manually” on page 60](#).

Note: This sample data model is based on the Customers.txt file that is supplied within the Getting Started/Samples/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. If you need to open it, you may do so by selecting **File > Open Project** from the menu bar.
2. Navigate to the `My IONA Projects/Getting Started/Samples/Creating Data Models/From a Text File` folder.
3. Right-click the `From a Text File` folder and select **Import > Import Text File**. This opens the **Import File** panel of the Text File Import Wizard.
4. Navigate to `Getting Started/Samples/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 `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. In the **Name** column, double click on "Row", type "Customer" as the value instead, and then press Enter. Notice how step 8 in the left-hand pane automatically changes from "Row" to "Customer".
12. Click the value in the **Type** column and select "Fixed Length".
13. Click **Next**. This opens the **Customer** panel.

Note: Notice how the panel name here, "Customer", 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.

14. According to the data in the `Customers.xls` file that is supplied within the `Getting Started/Samples/Creating Data Models/From a Text File` folder of your Artix Data Services Getting Started material, the length for Customer Number is 6, so in the **Fixed Offset Properties** section, click column 6 to automatically place a boundary between columns 5 and 6. This causes a new column, to be displayed in the **Preview - Column Data Types** section.

Note: The boundary comes after column 5 in this case, because the column numbers are starting at 0 rather than 1.


15. 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".
16. Accept "String" as the value in the **Selected Column Data Type** field.

17. According to the data in the `Customers.xls` file, the length for Customer Acronym is 12, so in the **Fixed Offset Properties** section, click column 18 to automatically place a boundary between columns 17 and 18. This causes a new column, to be displayed in the **Preview - Column Data Types** section.
18. 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".
19. Accept "String" as the value in the **Selected Column Data Type** field.
20. Repeat steps 17-19 in a similar fashion for the remaining fields. The following is a table of all the fields (columns) that need to be set up:

Column Name	Column Data Type	Start Column	End Column
Customer Number	String	0	5
Customer Acronym	String	6	17
Address Line 1	String	18	67
Address Line 2	String	68	117
Address Line 3	String	118	167
Address Line 4	String	168	217
Address Line 5	String	218	267
Post Zip Code	String	268	275
Telephone Number	String	276	295
Email Address	String	296	345
BIC	String	346	356
Fax Number	String	357	376
Telex Number	String	377	396
Country of Residence	String	397	398

Column Name	Column Data Type	Start Column	End Column
Fedwire Code	String	399	407
Chips Participant Code	String	408	411
Chips UID	String	412	417
Sort Code	String	418	423
Bankleitzhal Code	String	424	431

Note: 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 column 18 to determine the boundary of the "Address Line 1" column. Similarly, click "W" in column 68 to determine the boundary of the "Address Line 2" column, and so on.


21. 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. In the Messages window, an **Importing Text File** tab is opened to indicate that the import has been successful.
22. In the Explorer window, expand "File", right-click the "Customers" complex type, select **Rename**, and rename it to "Customers File".
23. In the Explorer window, right-click the "Customers" element (under the complex type), select **Rename**, and rename it to "Customers File" also.
24. Click `Customers.dod` in the Explorer window. This causes the properties for the data model to be displayed in the Properties window.
25. In the **General** section of the Properties window, set the value for **Target Namespace** to
<http://www.iona.com/ArtixDataServices/GettingStarted/Customer>.
26. Select **File > Save All** from the menu bar, or click the  icon on the toolbar, to save the data model.


27. In the Explorer window, expand "File" and double click the Customers File complex type. This opens a Customers File tab within the Customers.dod tab in the main window of the workbench. Expand the Customer element to view the contents. Compare the details displayed with those in the Customers.txt file that you imported.

Testing the accuracy of your data model


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, as follows:

1. Ensure that the Customers.dod data model is currently open.
2. Expand "File", 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**.

A dialog box is displayed prompting you to load the data you want to parse. Click the  icon in the dialog box to close it.

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, notice that an empty Run Customers File tab has been created at this point.

5. Click the  (**Load**) icon in the Customers File tab in the main window. This opens the Select Input File/Directory dialog.

6. Navigate to the Getting Started/Samples/Creating Data Models/From a Text File folder and select Customers.txt. Then click **Open**.
7. A Note dialog is displayed prompting you that files in subdirectories will be parsed by default. Click **OK** on this dialog.
8. A Confirm dialog is displayed prompting you that changing the URI will allow your data to be overwritten. Click **Yes** on this dialog.

In the case of this demonstration, there are no parsing errors, 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. You can now expand the CustomersFile node in the main window to view all the records in the file.

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 accuracy 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/Creating Data Models/From an XML Schema` folder of your Artix Data Services Getting Started material. This demonstration is illustrated by the `Creating a Data Model from an XML Schema` video tutorial which you can access from the `Getting Started/Videos` folder.

Steps

Follow these steps to start creating your data model:

1. In the Project window of the workbench, ensure that `MyProject.iop` is opened. If you need to open it, you may do so by selecting **File > Open Project** from the menu bar.
2. Navigate to the `My IONA Projects/Getting Started/Samples/Creating Data Models/From an XML Schema` folder
3. Right-click the `From an XML Schema` folder and select **Import > Import XML Schema**. This opens the **Files To Import** panel of the XML Schema Import Wizard.
4. Navigate to `Getting Started/Samples/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 `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.


In the Messages window, an **Importing XML Schema** tab is opened to indicate that the import has been successful.
10. Click `Statements.dod` in the Explorer window. This causes the properties for the data model to be displayed in the Properties window. Notice how the **Target Namespace** field has been automatically populated in this case with a namespace of `http://www.iona.com/ArtixDataServices/Training/Statements`, based on the imported schema.
11. Select **File > Save All** from the menu bar, or click the  icon on the toolbar, to save the data model.
12. In the Explorer window, double click the `StatementFile` complex type. This opens a `StatementFile` tab within the `Statements.dod` tab in the main window of the workbench. Expand the `Statement` element to view the contents. Compare the details displayed with those in the `Statements.xsd` file that you imported.

Testing the accuracy of your data model


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 StatementsXML.xml file into your Statements data model, as follows:

1. Ensure that the Statements.dod data model is currently open in the Explorer window.
2. Right-click the StatementFile 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 "StatementFile" (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**.

A dialog box is displayed prompting you to load the data you want to parse. Click the  icon in the dialog box to close it.

This opens a StatementFile tab (with a  icon beside its name) within the Statements.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.

In the Messages window, notice that an empty Run StatementFile tab has been created at this point.

5. Click the  (**Load**) icon in the StatementFile tab in the main window. This opens the Select Input File/Directory dialog.
6. Navigate to the `Getting Started/Samples/Creating Data Models/From an XML Schema` folder and select `StatementsXML.xml`. Then click **Open**.
7. A Note dialog is displayed prompting you that files in subdirectories will be parsed by default. Click **OK** on this dialog.
8. A Confirm dialog is displayed prompting you that changing the URI will allow your data to be overwritten. Click **Yes** on this dialog.

In the case of this demonstration, there are no parsing errors, so Artix Data Services creates instances of the model, based on your data. A green tick appears beside StatementFile in the StatementFile tab to indicate that parsing has been successful. You may now expand the StatementFile node in the main window to view all the records in the file.

Creating a Data Model from Other Sources

Overview

In Artix Data Services, there are several importers available to create data models from other formats. 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 39
Creating a Data Model from a Database	page 42

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 accuracy 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/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. If you need to open it, you may do so by selecting **File > Open Project** from the menu bar.
2. Navigate to the `My IONA Projects/Getting Started/Samples/Creating Data Models/From Other Sources` folder.
3. Right-click the `From Other Sources` folder and select **Import > Import XML Instance(s)**. This opens the **Files To Import** panel of the XML Instance(s) Import Wizard.
4. Navigate to `Getting Started/Samples/Creating Data Models/From Other Sources`. Then select `AccountsXML.xml` and click **Next**. This opens the **Target Directory** panel.
5. Accept the default folder `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. An `AccountsXML.dod` tab is also automatically opened in the main window of the workbench.
In the Messages window, an **Importing XML Instance(s)** tab is opened to indicate that the import has been successful.
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.

Testing the accuracy of your data model


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 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**.

A dialog box is displayed prompting you to load the data you want to parse. Click the  icon in the dialog box to close it.

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, notice that an empty Run AccountsFile tab has been created at this point.

5. Click the  (**Load**) icon in the AccountsFile tab in the main window. This opens the Select Input File dialog.
6. Navigate to the `Getting Started/Samples/Creating Data Models/From Other Sources` folder and select `AccountsXML.xml`. Then click **Open**.
7. A Note dialog is displayed prompting you that files in subdirectories will be parsed by default. Click **OK** on this dialog.
8. A Confirm dialog is displayed prompting you that changing the URI will allow your data to be overwritten. Click **Yes** on this dialog.

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. You may now expand the AccountsFile node in the main window to view all the records in the file. Compare the details with those in the original Accounts.txt and you will see that the two are exactly the same.

Creating a Data Model from a Database

Overview

This subsection demonstrates how to create a data model by importing a MySQL database called "adsubs" (Artix Data Services Universal Banking System). After creating the model, you can test its validity by parsing valid database entries through it.

Prerequisites

Before you proceed, you must first use MySQL on your machine to create the sample "adsubs" database. A text file called ADSUBS_SQL is supplied in the `Getting Started/Samples/Creating Data Models/From Other Sources` folder in your Artix Data Services Getting Started material. This text file contains the SQL necessary to create the database and its constituent tables. Use the "source" option in MySQL to execute the statements in the text file.

Steps

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

1. In the Project window of the workbench, ensure that `MyProject.iop` is opened. If you need to open it, you may do so by selecting `File > Open Project` from the menu bar.
2. Navigate to the `My IONA Projects/Getting Started/Samples/Creating Data Models/From Other Sources` folder.
3. Right-click the `From Other Sources` folder and select **Import > Import Database**. This opens the **Model Directory** panel of the Import Database Wizard.
4. Accept the default folder `From Other Sources` as the location where you want the data model to be stored.
5. Click **Next**. This opens the **Connection Properties** panel.
6. Type "ADSUBS" in the **Model Name** field.
7. Type "`http://www.iona.com/ArtixDataServices/GettingStarted/ADSUBS`" in the **Target Namespace** field.


8. For the purposes of this demonstration, select "MySQL" in the **Database Dialect** field. (This indicates the type of database from which you wish to import.) The **JDBC Driver Class Name** field is then automatically populated with "com.mysql.jdbc.Driver".
9. Update the value in the **Database URL** field with the name of your database, so make sure that the value reads as follows:

```
jdbc:mysql://localhost:3306/adsubs
```

Note: The default port for MySQL is 3306. If you are using an alternative port, replace 3306 in the preceding URL with whatever port your installation of MySQL is using.


10. Type a valid user name for connecting to the database in the **Username** field.

Note: For the purposes of connecting to a MySQL database, you might need to type a user name of `root` in this case.

11. Type a valid password for connecting to the database in the **Password** field.
12. Click **Edit Classpath**, click the  icon, and navigate to and select the `mysql-connector-java-x.x.x-bin.jar` file (where `x.x.x` represents the version number) in your MySQL Connector folder. This then adds this `.jar` file to the classpath.
13. Click **Next**. This opens the **Import Type** panel. Notice how the **Automatic table detection** check box is checked by default. (Do not adjust this.)
14. 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 the tables in the database are selected for import by default. Also, notice how the **Import related tables** check box and the **Child only** button are both selected by default. (Do not adjust these settings.)
15. Click **Next**. This opens the **Import Options** panel. Notice the various default selections and values on this panel. (Do not adjust these.)

16. 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.
17. Click **Next** repeatedly to display each of your database tables in turn. 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.

18. Click **Finish** when you are ready. This causes ADSUBS.dod to be automatically created and displayed in the Project and Explorer windows of the workbench. In this case, each imported table is created as a complex type.
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, double click each complex type in turn to open it in its own tab, and compare the details displayed with those in the original database table.


Testing the accuracy of your data model

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 contents of the adsubs database into your ADSUBS data model, as follows:


Note: It is assumed that you have already used MySQL to populate the various tables in the adsubs database with relevant data.

1. Ensure that the ADSUBS.dod data model is currently open.
2. Right-click (for example) the accounts 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 "accounts" (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 accounts tab (with a  icon beside its name) within the ADSUBS.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, note that an empty Run accounts tab has been created at this point.

5. Click the  (**Advanced**) icon in the accounts tab in the main window. This opens the Advanced dialog.
6. Ensure that the Input icon is selected.
7. Select "(Database)" in the **Format** field.
8. Click **Yes** on the Confirm dialog to continue.
9. Type "com.mysql.jdbc.Driver" in the **JDBC Driver Class Name** field.
10. Type "jdbc:mysql://localhost:3306/adsubs" in the **Database URL** field.

Note: If you are using an alternative port, replace 3306 in the preceding URL with whatever port your installation of MySQL is using.

11. Type a valid user name for connecting to the database in the **Username** field.

Note: For the purposes of connecting to a MySQL database, you might need to type a user name of `root` in this case.

12. Type a valid password for connecting to the database in the **Password** field.
13. Click **OK**.

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 accounts in the accounts tab to indicate that parsing has been successful. You may now expand the accounts node in the main window to view all the records in the table. Compare the details with those in the original database table and you will see that the two are exactly the same.

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 48
Creating the Customers Data Model Manually	page 60

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 sample data model is based on the information in the `Accounts.xls` file that is supplied within the `Getting Started/Samples/Creating Data Models/Manually` folder of your Artix Data Services Getting Started material. This demonstration is illustrated by the `Creating a Project` video tutorial which you can access from the `Getting Started/Videos` folder.

Note: Some types, such as dates, also require validation. However, validation rules are outside the scope of this particular demonstration.

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. If you need to open it, you may do so by selecting **File > Open Project** from the menu bar.
2. Navigate to the `My IONA Projects/Getting Started/Samples/Creating Data Models/Manually` folder.
3. Right-click the `Manually` folder and select **New > Data Model** from the context menu. Alternatively, click the `Manually` folder and select **File > New > Data Model** from the menu bar. This opens the **Setup** panel of the New Data Model Wizard.
4. Ensure that the **Create new empty data model** button is selected.
5. Type "Accounts" in the **Data Model name** field.

- For the purposes of this demonstration, type the following in the **Namespace** field:

<http://www.iona.com/ArtixDataServices/GettingStarted/Account>

Note: This will be the target namespace for this data model.

- For the purposes of this demonstration, accept the default location in the **Location** field.
- Click **Finish**. 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.

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:

- In the Explorer window, right-click on `Accounts.dod` and select **New > Atomic Simple Type** from the context menu. This opens the Atomic Simple Type Wizard.
- Type "AccountNumber" in the **Type name** field.
- Click **Next**. This opens the Base Type panel.
- Select **String** and then click **Finish**. "AccountNumber" is now automatically displayed under `Accounts.dod` in the Explorer window.
- Click "AccountNumber" in the Explorer window. This causes properties for the type to be automatically displayed in the Properties window.
- 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:

- In the Explorer window, right-click on `Accounts.dod` and select **New > Atomic Simple Type** from the context menu. This opens the Atomic Simple Type Wizard.
- Type "AccountName" in the **Type name** field.
- Click **Next**. This opens the Base Type panel.

4. Select **String** and then click **Finish**. "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 > Atomic Simple Type** from the context menu. This opens the Atomic Simple Type Wizard.
 2. Type "Blocked" in the **Type name** field.
 3. Click **Next**. This opens the Base Type panel.
 4. Select **String** and then click **Finish**. "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 > Atomic Simple Type** from the context menu. This opens the Atomic Simple Type Wizard.
2. Type "OpeningBalance" in the **Type name** field.
3. Click **Next**. This opens the Base Type panel.
4. Expand **Built-in**, expand **Numeric**, click **decimal**, and then click **Finish**. "OpeningBalance" 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 **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:

1. In the Explorer window, right-click on Accounts.dod and select **New > Atomic Simple Type** from the context menu. This opens the Atomic Simple Type Wizard.
2. Type "Customer" in the **Type name** field.
3. Click **Next**. This opens the Base Type panel.
4. Select **String** and then click **Finish**. "Customer" 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 6.

Creating a Currency Type

Next create a Currency type as follows:

1. In the Explorer window, right-click on Accounts.dod and select **New > Atomic Simple Type** from the context menu. This opens the Atomic Simple Type Wizard.
2. Type "Currency" in the **Type name** field.
3. Click **Next**. This opens the Base Type panel.
4. Select **String** and then click **Finish**. "Currency" 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 3.

Creating OpeningBalanceDate, ClosingBalanceDate and LastStatementDate types

Next create an OpeningBalanceDate type as follows:

1. In the Explorer window, right-click on Accounts.dod and select **New > Atomic Simple Type** from the context menu. This opens the Atomic Simple Type Wizard.
2. Type "OpeningBalanceDate" in the **Type name** field.
3. Click **Next**. This opens the Base Type panel.

4. Select **Generic Date** and then click **Finish**. "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:

1. In the Explorer window, right-click on Accounts.dod and select **New > Atomic Simple Type** from the context menu. This opens the Atomic Simple Type Wizard.
 2. Type "LastStatementNo" in the **Type name** field.
 3. Click **Next**. This opens the Base Type panel.
 4. Select **int** and then click **Finish**. "LastStatementNo" 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 Total Digits** and **Max Total Digits** to 12.
-

Creating a CardNumber type



Next create a CardNumber type as follows:


1. In the Explorer window, right-click on Accounts.dod and select **New > Atomic Simple Type** from the context menu. This opens the Atomic Simple Type Wizard.
2. Type "CardNumber" in the **Type name** field.
3. Click **Next**. This opens the Base Type panel.
4. Select **String** and then click **Finish**. "CardNumber" 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 > Complex Type** from the context menu. This opens the **New Complex Type** dialog.
2. 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.


A dialog box is displayed prompting you how you may add components to the complex type. Click the  icon in the dialog box to close it.
3. 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.
4. 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.
5. Click the "Account" complex type in the Explorer window. This causes the properties for the complex type to be displayed in the Properties window.
6. 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*.
7. 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.

8. 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**.
9. 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 > Complex Type** from the context menu. This opens the **New Complex Type** dialog.
2. 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.
3. 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.
4. 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.
5. Click Accounts.dod in the Explorer window. This causes the properties for the data model to be displayed in the Properties window. Notice how the namespace you specified in the New Data Model Wizard is now displayed in the **Target Namespace** field in the Properties window.

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

Creating an Accounts File element

To enable the model to be subsequently used in code, you must also create an element for the Accounts File complex type as follows:

1. In the Explorer window, right-click on Accounts.dod and select **New > Element** from the context menu. This opens the **New Element** dialog.
2. Type "Accounts File" in the text box and click **OK**. This opens the **Select Type** dialog.
3. Expand "Local", click the Accounts File complex type, and click **OK**. This displays a dialog box prompting you to open the type for the element.
4. Click **Yes** on the dialog box. The Accounts File element is automatically displayed under Accounts.dod in the Explorer window.
5. 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 and element 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 test the accuracy of the Accounts data model by checking to see if it can parse a valid text file.


Testing the accuracy of your data model


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 create Java class instances of that model. In this case, you can read the supplied Accounts.txt file into your Accounts data model, as follows:

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**.

A dialog box is displayed prompting you to load the data you want to parse. Click the  icon in the dialog box to close it.

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, note that an empty Run Accounts File tab has been created at this point.


4. Click the  (**Load**) icon in the Accounts File tab in the main window. This opens the Select Input File/Directory dialog.
5. Navigate to the `Getting Started/Samples/Creating Data Models/Manually` folder and select `Accounts.txt`. Then click **Open**.
6. A Note dialog is displayed prompting you that files in subdirectories will be parsed by default. Click **OK** on this dialog.
7. A Confirm dialog is displayed prompting you that changing the URI will allow your data to be overwritten. Click **Yes** on this dialog.

In the case of this demonstration, a dialog box is opened indicating that there is a parsing error on the OpeningBalance type. The error is also displayed in the **Run Accounts File** tab in the Messages window. 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. Click the  (**Reload Active Run Configuration**) icon on the toolbar to automatically reload the Accounts.txt file into the updated model.

In this case, another dialog box is opened indicating that there is now a parsing error on the ClosingBalance type. 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. The error is also displayed in the **Run Accounts File** tab in the Messages window.

6. Click ClosingBalance in the Explorer window and repeat steps 2-5. In this case, another dialog box is opened indicating that there is now a parsing error on the OpeningBalanceDate type. This parsing error now needs to be corrected, as described next.

Fixing parsing errors relating to dates

The Accounts.txt file expects the opening balance date to have a date format of yyMMdd, but this has not been set as a property of the OpeningBalanceDate type in the data model. Follow these steps to fix this parsing error:

1. Click OpeningBalanceDate 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** section and click the **Date Format** field. This opens a date format dialog.
3. Click the  icon beside the **Pattern** field in the dialog. This opens the **Insert Character** dialog.
4. 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".
5. Click **OK**. The **Pattern** field in the first date format dialog now displays "yyMMdd" also.
6. Click **OK**. The **Date Format** field in the Properties window now displays "yyMMdd" also.
7. Click the  (**Reload Active Run Configuration**) icon to automatically reload the Accounts.txt file into the updated model.

In this case, another dialog box is opened indicating that there is now a parsing error on the ClosingBalanceDate type. Again, the Accounts.txt file expects the closing balance date to have a date format of yyMMdd, but this has not been set as a property of the ClosingBalanceDate type in the data model. The error is also displayed in the **Run Accounts File** tab in the Messages window.

8. Click ClosingBalanceDate in the Explorer window and repeat steps 2-7.

In this case, another dialog box is opened indicating that there is now a parsing error on the LastStatementDate type. Again, the Accounts.txt file expects the last statement date to have a date format of yyMMdd, but this has not been set as a property of the LastStatementDate type in the data model. The error is also displayed in the **Run Accounts File** tab in the Messages window.

9. Click LastStatementDate in the Explorer window and repeat steps 2-7.

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. In this case, a green tick appears beside AccountsFile in the Accounts File tab to indicate that parsing has been successful. The **Run AccountsFile** tab in the Messages window also displays a message that parsing has been successful. You may 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 if you have already followed the instructions in “[Creating a Data Model from a Text File](#)” on page 19 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 `Customers.xls` file that is supplied within the `Getting Started/Samples/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.

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. If you need to open it, you may do so by selecting **File > Open Project** from the menu bar.
2. Navigate to the `My IONA Projects/Getting Started/Samples/Creating Data Models/Manually` folder
3. Right-click the `Manually` folder and select **New > Data Model** from the context menu. Alternatively, click the `Manually` folder and select **File > New > Data Model** from the menu bar. This opens the **Setup** panel of the New Data Model Wizard.
4. Ensure that the **Create new empty data model** button is selected.

5. Type "Customers" in the **Data Model name** field.
6. For the purposes of this demonstration, type the following in the **Namespace** field:

`http://www.iona.com/ArtixDataServices/GettingStarted/Customer`

Note: This will be the target namespace for this data model.

7. For the purposes of this demonstration, accept the default location in the **Location** field.
8. 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 opened in the main window of the workbench.

Creating a Customer Number type

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

1. In the Explorer window, right-click on `Customers.dod` and select **New > Atomic Simple Type** from the context menu. This opens the Atomic Simple Type Wizard.
2. Type "Customer Number" in the **Type name** field.
3. Click **Next**. This opens the Base Type panel.
4. Select **String** and then click **Finish**. "Customer Number" is now automatically displayed under `Accounts.dod` in the Explorer window.
5. Click "Customer Number" 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 6.

Creating a Customer Acronym type

Next create a Customer Acronym type as follows:

1. In the Explorer window, right-click on `Customers.dod` and select **New > Atomic Simple Type** from the context menu. This opens the Atomic Simple Type Wizard.
2. Type "Customer Acronym" in the **Type name** field.

3. Click **Next**. This opens the Base Type panel.
 4. Select **String** and then click **Finish**. "Customer Acronym" 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 an Address Line type


Next create an AddressLine type as follows:

1. In the Explorer window, right-click on Customers.dod and select **New > Atomic Simple Type** from the context menu. This opens the Atomic Simple Type Wizard.
 2. Type "Address Line" in the **Type name** field.
 3. Click **Next**. This opens the Base Type panel.
 4. Select **String** and then click **OK**. "Address Line" 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.
-

Creating an Address complex type

Next create an Address complex type that will be able to hold multiple address lines, as follows:

1. In the Explorer window, right-click on Customers.dod and select **New > Complex Type** from the context menu. This opens the **New Complex Type** dialog.
2. Type "Address" in the text box and click **OK**. The Address complex type is automatically displayed under Customers.dod in the Explorer window. An Address tab is also automatically opened within the Customers.dod tab in the main window of the workbench.

A dialog box is displayed prompting you how you may add components to the complex type. Click the  icon in the dialog box to close it.

3. Click the Address Line type in the Explorer window. Then drag and drop it over to the Address complex type in the main window of the workbench. This causes Address Line to be displayed in the main window under the Address complex type.
4. For the purposes of this example, the address needs to contain 5 address lines, so the cardinality value needs to be changed in this case. Right-click Address Line in the **Component** column, select **Cardinality**, select **n**, type "5" as the fixed cardinality value, and click **OK**.

Creating other simple types

Now create the rest of the simple types that relate to the Customers data model. In each case, make sure that you right-click on Customers.dod and select **New > Atomic Simple Type** to start creating the type. These types include:


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

For details of these types, refer to the `Customers.xls` file that is supplied within the `Getting Started/Samples/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 > Complex Type** from the context menu. This opens the **New Complex Type** dialog.
2. 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.

A dialog box is displayed prompting you how you may add components to the complex type. Click the  icon in the dialog box to close it.

3. 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 and the Address complex type to appear highlighted in the Explorer window.
4. Drag and drop the highlighted types from the Explorer window over to the Customer complex type in the main window of the workbench. This causes all the simple types and Address complex type to be displayed in the main window under the Customer complex type.
5. Right-click the Address Line simple type in the main window and select **Delete**. This opens the **Confirm delete** dialog. Then click **OK** to delete Address Line from the list of components.

Note: You need to delete this from the list of components, because the Address complex type is already set up to pull in the Address Line type with a cardinality of 5.

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 > Complex Type** from the context menu. This opens the **New Complex Type** dialog.
2. 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.
3. 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.
4. 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.

5. Click Customers.dod in the Explorer window. This causes the properties for the data model to be displayed in the Properties window. Notice how the namespace you specified in the New Data Model Wizard is now displayed in the **Target Namespace** field in the Properties window.
6. Select **File > Save All** from the menu bar, or click the  icon on the toolbar, to save the data model.

Creating a Customers File element

To enable the model to be subsequently used in code, you must also create an element for the Customers File complex type as follows:

1. In the Explorer window, right-click on Customers.dod and select **New > Element** from the context menu. This opens the **New Element** dialog.
2. Type "Customers File" in the text box and click **OK**. This opens the **Select Type** dialog.
3. Expand "Local", click the Customers File complex type, and click **OK**. This displays a dialog box prompting you to open the type for the element.
4. Click **Yes** on the dialog box. The Customers File element is automatically displayed under Customers.dod in the Explorer window.
5. 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 and element 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 test the accuracy of the Customers data model by checking to see if it can parse a valid text file.

Testing the accuracy of your data model


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, as follows:

1. Ensure that the Customers.dod data model is open in the Explorer window.
2. Right-click the Customers 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 "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.
3. For the purposes of this demonstration, accept all the default values on the Run Wizard dialog and click **Run**.

A dialog box is displayed prompting you to load the data you want to parse. Click the  icon in the dialog box to close it.

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, note that an empty Run Customers File tab has been created at this point.

4. Click the  (**Load**) icon in the Customers File tab in the main window. This opens the Select Input File/Directory dialog.
5. Navigate to the `Getting Started/Samples/Creating Data Models/Manually` folder and select `Customers.txt`. Then click **Open**.

In the case of this demonstration, there are no parsing errors, 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. You may now expand the CustomersFile node in the main window to view all the records in the file.

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 Transactions data models.

In this section

This section discusses the following topics:

Adding Validation Rules for Accounts Data Model	page 70
Adding Validation Rules for Transactions Data Model	page 74

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_validation.xls` file that is supplied within the `Getting Started/Samples/Creating Data Models/Adding Validation Rules` folder of your Artix Data Services Getting Started material.


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. If you need to open it, you may do so by selecting **File > Open Project** from the menu bar.
2. Navigate to the `My IONA Projects/Getting Started/Samples/Creating Data Models/Manually` folder.
3. 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. Click the  icon to add a new value to the enumeration. This opens the **New Enumeration Value** dialog.
8. Type "N" and click **OK**. This causes a new row to be added to **Enumeration** with "N" as the displayed value.
9. 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.
10. 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.
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 CardNumber type

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

1. Click "CardNumber" 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 Component(s)**. 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 open in the Explorer window.
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, there are no validation errors, because the format of the card number details in the imported text file exactly matches the format of the CardNumber type in your data model.

4. Now try loading some invalid data for the Blocked type, to see what happens. To do this, click the  (**Load**) icon to open the Select Input File/Directory dialog. Then navigate to the `Getting Started/Samples/Creating Data Models/Adding Validation Rules` folder, select `Accounts_invalid.txt`, and click **Open**.

5. A Note dialog is displayed prompting you that files in subdirectories will be parsed by default. Click **OK** on this dialog.
6. A Confirm dialog is displayed prompting you that changing the URI will allow your data to be overwritten. Click **Yes** on this dialog.
7. Notice how the first Account record is now automatically marked with a red X. Expand it and you will see that the Blocked element is also marked in red. This is because the Blocked type is only meant to accept a value of "Y" or "N", but it is currently displaying an invalid value of "A" for the first record.

8. Click the Validation tab at the bottom of the workbench to open the Validation window. In this case, a validation failure is now being reported against that Blocked element.
9. Now try making the format of one of the CardNumber elements deliberately invalid. To do this, expand a particular Account record in the tab and click the value for its constituent CardNumber. Click the down arrow that is displayed in the CardNumber field and update its value on the Multiline textual value dialog, by inserting a hyphen after every fourth digit, as follows:

4325-6486-3757-2678

10. Click **OK** to close the dialog and then click anywhere in the workbench. Notice how that CardNumber element and its parent Account component are now automatically marked in red with an X. The Validation window is now reporting additional validation errors against that CardNumber element.
11. Now try loading valid data again, to see what happens. To do this, click the  (**Load**) icon to open the Select Input File/Directory dialog. Then navigate to the Getting Started/Samples/Creating Data Models/Manually folder, select Accounts.txt, and click **Open**. Notice how all Account records are now automatically displayed as valid again.

This proves that the validation rules for the Blocked and CardNumber types are working, because validation failures are being correctly reported against invalid data.

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. If you need to open it, you may do so by selecting **File > Open Project** from the menu bar.
 2. Navigate to the `My IONA Projects/Getting Started/Samples/Creating Data Models/From a Text File` folder.
 3. 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 > Validation Rule**. This opens the **New Validation Rule** dialog.
2. 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.

3. 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.
4. If the validation rule is true, the data model should throw an error. Therefore, type "Commission Error" in the **Error Message** pane.
5. Select **File > Save All** from the menu bar, or click the  icon on the toolbar, to save the data model.
6. 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.
7. 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.
8. In the Properties window, scroll down to the **Validation** section and click the field beside **Validation Rules**. This opens a validation rules dialog.
9. Click the  icon. This opens the **Add Validation Rule** dialog.
10. Now apply the global Commission Check validation rule to the Customer Details type. Expand "Local", select the "Commission Check" global validation rule, and click **OK**. This adds Commission Check to the validation rules dialog.
11. Click **OK** to close the validation rules dialog. The **Validation Rules** field in the Properties window now displays "1".
12. 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 in the Explorer window.
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, there are no validation errors, because none of the data you previously loaded will fail this validation check.

5. Now try loading some invalid data to see what happens. To do this, click the  (**Load**) icon to open the Select Input File dialog. Then navigate to the Getting Started/Samples/Creating Data Models/Adding Validation Rules folder, select Transactions_invalid.txt, and click **Open**.
6. A Note dialog is displayed prompting you that files in subdirectories will be parsed by default. Click **OK** on this dialog.
7. A Confirm dialog is displayed prompting you that changing the URI will allow your data to be overwritten. Click **Yes** on this dialog.
8. Notice how some Customer Details records now 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.

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

This proves that the Commission Check validation rule is working, because validation failures are being correctly reported against 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 80
Making Your Transformation More Complex	page 92

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 `Creating a Simple Transformation` video tutorial which you can access from the `Getting Started/Videos` folder of your Artix Data Services Getting Started material. A completed version of this transformation is supplied in the `Getting Started/Samples/Creating Transformations/Simple Transformation/Completed Transformation` folder.

In this section

This section discusses the following topics:

Starting to Create a Transformation	page 81
Creating a Local Transformation	page 83
Testing the Local Transformation in Your Main Transformation	page 86
Creating a Filter	page 88
Testing the Filter in Your Main Transformation	page 90


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. If you need to open it, you may do so by selecting **File > Open Project** from the menu bar.
2. Navigate to the `My IONA Projects/Getting Started/Samples/Creating Transformations/Simple Transformation` folder.
3. Right-click the `Simple Transformation` folder and select **New > Transform**. This opens the **Setup** panel of the New Transform wizard.
4. For the purposes of this example, the transformation is called StatGen, because its purpose will be to generate statements based on transaction details. So type "StatGen" in the **Transform name** field.
5. Accept the default location in the **Location** field.
6. Click the **Advanced** button to display some optional panels and then click **Next**. This opens the Select New Input Data Type panel, which allows you 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 needs to be added as your input.
7. Click the  icon. This opens the Select New Input Data Model dialog.
8. Navigate to `My IONA Projects/Getting Started/Samples/Creating Transformations`, select `Transactions.dod` and click **OK**. This opens the Select New Input Type dialog.
9. Expand **Local**, expand **File**, select the **Transactions** complex type, and click **OK**. The Transactions data model is now added to the Select New Input Data Type panel.

10. Click **Next**. This opens the Select New Output Data Type panel, which allows you 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 needs to be added as your output.
11. Click the  icon. This opens the Select New Input Data Model dialog.
12. Navigate to `My IONA Projects/Getting Started/Samples/Creating Transformations`, select `Statements.dod` and click **OK**. This opens the Select New Output Type dialog.
13. Expand Local, select the **StatementFile** complex type, and click **OK**. The Statements data model is now added to the Select New Output Data Type panel.
14. Click **Finish**. 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. Notice how the Transactions complex type is displayed along with its Header and Customer Details elements in the Inputs section of the MAIN tab. Notice also how 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. Expand "Statement" in the Outputs section to display its three sub-elements.
2. Click "Customer Details" in the Inputs section to highlight it.
3. Click "Customer Details" again and drag your mouse across to "StmtLine" in the Outputs section while holding the left mouse key.

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?
```

4. 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 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 StatGen 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.

A dialog box is displayed prompting you to load the data you want to parse. Click the  icon in the dialog box to close it.

5. Click the  (**Load**) icon in the Inputs section to open the Select Input File/Directory dialog. Then navigate to the `Getting Started/Samples/Creating Transformations` folder, select `Transactions.txt`, and click **Open**.
6. A Note dialog is displayed prompting you that files in subdirectories will be parsed by default. Click **OK** on this dialog.

7. A Confirm dialog is displayed prompting you that changing the URI will allow your data to be overwritten. Click **Yes** on this dialog.
8. This loads the relevant data records into your input model.
9. Expand "Transactions" in the Inputs section to view the various Customer Details records that form your input. Notice how an arrow is now automatically mapped from each CustomerDetails record to "Customer Details" in the Record to StmtLine local transformation.
10. Click the  (**Perform Transformation**) icon on the toolbar. This automatically sets up a connection between "StatementLine" in the Record to StmtLine local transformation and the output model. Relevant data from the input model is now automatically loaded in the output model.

In this case, expand StatementFile and Statement and you will see seven StmtLine records corresponding to the seven CustomerDetails records in the Inputs section. Expand each StmtLine record and you will see that it includes values for TxAmount and PostingNarrative.

This proves that your local transformation is working correctly, because it has produced the expected results.

Note: The errors being reported in the Outputs section at this point are validation errors. These are due to the fact that various other mandatory elements (that is, elements with a cardinality of 1) within StatementFile are not currently being mapped to. Ignore these validation errors for the purposes of this demonstration.


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.
2. Click the arrow that is between the Inputs section and the local transformation, to highlight it.
3. Right-click the highlighted arrow and select **Add Filter** from the context menu. This opens a **Filter Customer Details** tab for the filter (with a  icon beside its name) within the StatGen.tfd tab.

Notice how the Inputs section of the filter tab is automatically populated with the relevant input type. Notice also how it is automatically mapped to the Value pane in the Outputs section.

Note: The Outputs section for a filter is divided into a **Condition** pane and a **Value** pane. The purpose of these will be shown in a minute. You cannot add output models to filters.

4. For the purposes of this example, rename the filter to "JustScrooge". To do this, click the MAIN tab, right-click the filter in the ALL section, select **Rename**, type "JustScrooge" and click **OK**. The new name is automatically reflected in the filter and its corresponding tab.

Adding the EQUALS function to your filter

You now need to specify the logic of the filter that you want to implement. For the purposes of this example, let's use a logic function called EQUALS. Follow these steps to add the EQUALS function to your filter:

1. Click the JustScrooge tab to reopen it.
2. In the ALL section, right-click and 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.

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 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.

Notice how "Customer Details" in the Inputs section is now automatically mapping to "Customer Details" in the JustScrooge filter. Notice also how "Value" in the JustScrooge filter is automatically mapping to "Customer Details" in the Record to StmtLine local transformation.

Running the transformation

You may now run your transformation to see the potential results that the new filter 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 reopens the 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. The relevant data records are automatically reloaded into your input model.

4. Expand "Transactions" in the Inputs section to view the various Customer Details records that form your input. Notice how an arrow is now automatically mapped from each CustomerDetails record to "Customer Details" in the JustScrooge filter.
5. Relevant data from the input model is automatically loaded in the output model. In this case, expand StatementFile and Statement and you will now see only two StmtLine records. Expand each StmtLine record and you will see that they are based on the two CustomerDetails 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.

Note: Again, the errors being reported in the Outputs section at this point are validation errors due to the fact that various other mandatory elements (that is, elements with a cardinality of 1) within StatementFile are not currently being mapped to. Ignore these validation errors for the purposes of this demonstration.

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 93
Adding More Input Models to Your Main Transformation	page 95
Adding Local Transformations	page 96
Adding Functions	page 99
Adding Nested Local Transformations	page 104
Adding Hash Tables	page 112
Adding Filters	page 116
Adding Java Methods	page 122
Adding Introspect Functions	page 125

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 Design tab and then 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.

Note: Notice how "Customer Details" in the Record to StmtLine local transformation is now displayed in red, because you have removed its corresponding input mapping.

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 later.

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: 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/Creating Transformations*, select *Customers.dod* and click **OK**. This opens the **Select New Input Type** dialog.
3. Expand "Local", 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 Customers 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/Creating Transformations*, select *Accounts.dod* and click **OK**. This opens the **Select New Input Type** dialog.
6. Expand "Local", 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 [“Adding Local Transformations” on page 96](#) 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 and 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 and 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 click `StatGen.tfd` to populate it in the **File name** field. Then navigate to the `My IONA Projects/Getting Started/Samples/Creating Transformations` folder, double click `Adding Local Transformations`, and click **Save**. This saves the updated transformation into the `Adding Local Transformations` folder.

At this point, your transformation is not very functional, so you need to add some functions to it. See [“Adding Functions” on page 99](#) for more details.

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 [“Adding Local Transformations” on page 96](#).

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. If you need to open it, you may do so by selecting **File > Open Project** from the menu bar.
2. Navigate to the `My IONA Projects/Getting Started/Samples/Creating Transformations/Adding Local Transformations` folder
3. Right-click the `StatGen.tfd` file and select **Open Selected**. This opens the `StatGen.tfd` transformation in the main view of the workbench.

Mapping input "OpeningBalance" to output "StartBalance"

In this case, you want the opening balance in each Account record to be displayed as a start balance in your output statements. You therefore need to map "OpeningBalance" in your Account input model to "StartBalance" in your output model. To do this:

1. Click the AccountTxns to Statement tab.
2. Try to connect "OpeningBalance" (under Account) in the Inputs section to "StartBalance" (under Hdr) in the Outputs section. 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 "OpeningBalance" and "StartBalance" because they are not of the same type—one is a decimal and the other is a float.

3. Click **OK** to indicate that you want a CAST function to be automatically created to force a compatible mapping between the "OpeningBalance" decimal type and the "StartBalance" float type.

The CAST function is automatically displayed in the ALL section of the AccountTxns to Statement tab, with "OpeningBalance" in the Inputs section connected to "Arg1" in the CAST function, and "Result" in the CAST function connected to "StartBalance" in the Outputs section.

Mapping input "ClosingBalance" to output "EndBalance"

You also want the closing balance in each Account record to be displayed as an end balance in your output statements. You therefore need to map "ClosingBalance" in your Account input model to "EndBalance" in your output model. To do this:

1. Click the AccountTxns to Statement tab.
2. Try to connect "ClosingBalance" (under Account) in the Inputs section to "EndBalance" (under Tlr) in the Outputs section. 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 "ClosingBalance" and "EndBalance" because they are not of the same type—one is a decimal and the other is a float.

3. Click **OK** to indicate that you want a CAST function to be automatically created to force a compatible mapping between the "ClosingBalance" decimal type and the "EndBalance" float type.

The CAST function is automatically displayed in the ALL section of the AccountTxns to Statement tab, with "ClosingBalance" in the Inputs section connected to "Arg1" in the CAST function, and "Result" in the CAST function connected to "EndBalance" in the Outputs section.

Creating NOW and CONVERTDATE functions

Next 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 > Function**. This opens the **New Function** dialog.
3. Expand **Date & Time**, select **NOW** and click **OK**. The NOW function is displayed in the ALL section.
4. 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.

5. 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 > Function**. This opens the **New Function** dialog.
3. Expand **Math**, then expand **Arithmetic**, select **ADD** and click **OK**. The ADD function is displayed in the ALL section.
4. Connect "LastStatementNo" in the Account input model to "Arg1" in the ADD function. This displays an arrow going from "LastStatementNo" to "Arg1".
5. Right-click "Arg2" in the ADD function and select **Set Constant Value**. This opens the **Set Constant Value** dialog.
6. Type "1" as the constant value and click **OK**. This sets Arg2 to a value of 1.
7. 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.

8. 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 "StmntNo" in the Outputs section.

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

9. Select **File > Save > Save Tab As** and click `StatGen.tfd` to populate it in the **File name** field. Then navigate to the `My IONA Projects/Getting Started/Samples/Creating Transformations` folder, double click `Adding Functions`, and click **Save**. This saves the updated transformation into the `Adding Functions` folder.

You have now added various functions and mappings to successfully output the starting balance, ending balance, 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 AccountTxns to Statement local transformation. So let's look at adding some nested local transformations next. See ["Adding Nested Local Transformations" on page 104](#) 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 'AccountTxns 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.

Note: Before you continue, ensure that you have completed the instructions in [“Adding Functions” on page 99](#).

Moving the "Record to StmtLine" local transformation

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

1. In the Project view of the workbench, ensure that MyProject.iop is opened. If you need to open it, you may do so by selecting **File > Open Project** from the menu bar.
2. Navigate to the `My IONA Projects/Getting Started/Samples/Creating Transformations/Adding Functions` folder.
3. Right-click the `StatGen.tfd` file and selected **Open Selected**. This opens the `StatGen.tfd` transformation in the main view of the workbench.
4. Click the MAIN tab.
5. In the ALL section, right-click the Record to StmtLine local transformation and select **Delete**. This opens a **Confirm Delete** dialog.
6. Click **OK**. This opens a **Confirm Component Delete** dialog.
7. Click **No** on the Confirm Component Delete dialog.
8. Click the AccountTxns to Statement tab to open it.

9. Right-click in the ALL section and select **New > Transform Reference**. This opens the New Transform Reference dialog.
10. Expand My IONA Projects/Getting Started/Samples/Creating Transformations/Adding Functions and click StatGen.tfd.
11. Click **OK**. This opens the Select Component dialog.
12. Select Record to StmtLine and click **OK**. This adds the Record to StmtLine local transformation to the AccountTxns to Statement tab.

Adding SIZE and DIVIDE functions within "AccountTxns to Statement"

Now that the Record to StmtLine local transformation has been moved, let's add functions that will allow the output from "Record to StmtLine" to be mapped to the StmtPage in the Statement output model. This means that we can take a series of individual records and use them as a collection to determine an overall count of the records.

In this case, a SIZE function will be used to take the output from Record to StmtLine and return the size of the input list. Then a DIVIDE function will take the size of the input list and divide it by 10, to output the correct value for Statement Page (that is, there will be 10 statement records per page).

Follow these steps:

1. Click the AccountTxns to Statement tab.
2. In the ALL section, right-click and select **New > Function**. This opens the **New Function** dialog.
3. Expand **Collections**, select **SIZE** and click **OK**. The SIZE function is displayed in the ALL section.
4. In the ALL section, right-click and select **New > Function**. This opens the **New Function** dialog.
5. Expand **Math**, then **Arithmetic**, select **DIVIDE** and click **OK**. The DIVIDE function is displayed in the ALL section.
6. Connect "StatementLine" in the Record to StmtLine local transformation to "Arg1" in the SIZE function. This displays an arrow going from "StatementLine" "Arg1" in SIZE.
7. Connect "Result" in the SIZE function to "Arg1" in the DIVIDE function. This displays an arrow going from "Result" in SIZE to "Arg1" in DIVIDE.
8. Right-click "Arg2" in the DIVIDE function and select **Set Constant Value** from the context menu. This opens the **Set Constant Value** dialog.

9. Type "10" in the text box and click **OK**. 10 is now displayed in the ALL section as a constant value for Arg2 in DIVIDE.
10. Expand "Hdr" in the Statement output model.
11. Try to connect "Result" in the DIVIDE function to "StmtPage" 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 DIVIDE function returns a number type that is incompatible with the StmtPage, and is prompting you to automatically create a CAST function that will convert the number derived from the DIVIDE function to the correct type.

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

12. 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 DIVIDE function connected to "Arg1" in the CAST function, and "Result" in the CAST function connected to "StmtPage" in the Outputs section.

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

Adding functions within "Record to StmtLine"

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




1. Click the Record to StmtLine tab to open it.
1. In the ALL section, right-click and select **New > Function**. This opens the **New Function** dialog.
2. Expand **Date & Time**, select **CONVERTDATE** and click **OK**. This opens the Select Return Type dialog.
3. 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.

4. 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.
5. 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.
6. In the ALL section of the Record to StmtLine tab, right-click and select **New > Function**. This opens the **New Function** dialog.
7. Expand **Logic**, select **GREATERTHAN** and click **OK**. The GREATERTHAN function is now displayed in the ALL section of the Record to StmtLine tab.
8. 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.
9. Right-click "Arg2" in the GREATERTHAN function and select **Set Constant Value** from the context menu. This opens the **Set Constant Value** dialog.
10. Type "0" in the text box and click **OK**. 0 is now displayed in the ALL section as a constant value for Arg2.
11. In the ALL section of the Record to StmtLine tab, right-click and select **New > Function**. This opens the **New Function** dialog.
12. 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.
13. 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.
14. Connect "Result" in the GREATERTHAN function to "Condition" in the IF function. This displays an arrow going from "Result" to "Condition", and Condition is now displayed in black.
15. Right-click "WhenTrue" in the IF function and select **Set Constant Value** from the context menu. This opens the **Set Constant Value** dialog.

16. Type "DR" in the text box and click **OK**. DR is now displayed in the ALL section as a constant value for WhenTrue.
17. Right-click "WhenFalse" in the IF function and select **Set Constant Value** from the context menu. This opens the **Set Constant Value** dialog.
18. Type "CR" in the text box and click **OK**. CR is now displayed in the ALL section as a constant value for WhenFalse.
19. 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.
20. Connect "Amount" in the Inputs section to "TxAmount" in the Outputs section. This automatically prompts you to set up a CAST function between the two types. Click **OK** to add the CAST function.
21. 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".
22. Click the AccountTxns to Statement tab to open it.
23. Connect "Customer Details" (under Transactions) 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.
24. 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. In the ALL section, right-click and select **New > Local Transform**. This opens the **New Local Transform** dialog.
3. 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.
4. In the Inputs section of the Populate NameAndAddress tab, click the  (**New Local Input**) icon (Alternatively, right-click in the ALL section and select **New > Local Input**.) This opens the **Add Input** dialog.
5. Select "Customers File" and click **OK**. This opens the **Select New Input Path** dialog.
6. Select "Customer" and click **OK**. This displays the Customer complex type and its elements in the Inputs section of the Populate NameAndAddress tab.
7. In the Outputs section of the Populate NameAndAddress tab, click the  (**New Local Output**) icon (Alternatively, right-click in the ALL section and select **New > Local Output**.) This opens the **Select New Output Path** dialog.
8. 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 within "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 > Function**. This opens the **New Function** dialog.
2. Expand **Collections**, select **UNION** and click **OK**. The UNION function is now displayed in the ALL section of the Populate NameAndAddress tab.

3. 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.
4. Expand "Address" in the Inputs section and connect its constituent "AddressLine" to "Arg2" in the UNION function. This displays an arrow going from "AddressLine" to "Arg2", and Arg2 is now displayed in black.
5. In the ALL section of the Populate NameAndAddress tab, right-click and select **New > Function**. This opens the **New Function** dialog.
6. Expand **Collections**, select **SUBLIST** and click **OK**. The SUBLIST function is now displayed in the ALL section of the Populate NameAndAddress tab.
7. Connect "Result" in the UNION function to "List" in the SUBLIST function. This displays an arrow going from "Result" to "List", and both parameters are now displayed in black.
8. Right-click "BeginIndex" in the SUBLIST function and select **Set Constant Value** from the context menu. This opens the **Set Constant Value** dialog.
9. Type "0" in the text box and click **OK**. 0 is now displayed in the ALL section as a constant value for BeginIndex.
10. Right-click "EndIndex" in the SUBLIST function and select **Set Constant Value** from the context menu. This opens the **Set Constant Value** dialog.
11. Type "5" in the text box and click **OK**. 5 is now displayed in the ALL section as a constant value for EndIndex.
12. 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.
13. Connect "Country Of Residence" in the Inputs section to "Ctry" in the Outputs section.
14. Click the AccountTxns to Statement tab to open it.
15. 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.

16. 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.
17. Select **File > Save > Save Tab As** and click StatGen.tfd to populate it in the **File name** field. Then navigate to the My IONA Projects/Getting Started/Samples/Creating Transformations folder, double click Adding Nested Local Transformations, and click **Save**. This saves the updated transformation into the Adding Nested Local Transformations folder.

Next, let's look at adding a hash table to the transformation. See ["Adding Hash Tables" on page 112](#) 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 using the Run Wizard by transforming a file from the input model to 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 currency node which uses the hash table at this stage.

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

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. If you need to open it, you may do so by selecting **File > Open Project** from the menu bar.
2. Navigate to the `My IONA Projects/Getting Started/Samples/Creating Transformations/Adding Nested Local Transformations` folder
3. Right-click the `StatGen.tfd` file and select **Open Selected**. This opens the `StatGen.tfd` transformation in the main view of the workbench.
4. Click the AccountTxns to Statement tab.
5. In the ALL section, right-click and 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.
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):

Input	Output
EUR	Euro
GBP	British Pound
JPY	Japanese 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 click `StatGen.tfd` to populate it in the **File name** field. Then navigate to the `My IONA Projects/Getting Started/Samples/Creating Transformations` folder, double click `Adding Hash Tables`, and click **Save**. This saves the updated transformation into the `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. For the same record, also 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. For the same record, also 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.
8. Select **File > Save > Save Tab As** and click StatGen.tfd to populate it in the **File name** field. Then navigate to the My IONA

Projects/Getting Started/Samples/Creating Transformations folder, double click Adding Hash Tables, and click **Save**. This saves the updated transformation into the Adding Hash Tables folder.

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 116](#) 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 112](#).


Creating the SameAccount filter

Follow these steps to create the SameAccount filter:

1. Click the AccountTxns to Statement tab.
2. Click the arrow that is between "Customer Details" (under Transactions) in the Inputs section and "Customer Details" in the Record to StmtLine local transformation. This highlights the arrow.
3. Right-click the highlighted arrow and select **Add Filter** from the context menu. This opens a **Filter Customer Details** tab for the filter (with a  icon beside its name) within the AccountTxns to Statement tab.

Notice how the Inputs section of the filter tab is automatically populated with the Customer Details input type. Notice also how it is automatically mapped to the Value pane in the Outputs section.

Note: Customer Details will be the first input element to be involved in the comparison.

4. For the purposes of this example, rename the filter to "SameAccount". To do this, click the AccountTxns to Statement tab, right-click the filter in the ALL section, select **Rename**, type "SameAccount" and click **OK**. The new name is automatically reflected in the filter and its corresponding tab.
5. Now select the input model that will contain the second element to be involved in the comparison. For the purposes of this comparison select the Accounts model, as follows:
 - i. Click the  (**New Local Input**) icon. (Alternatively, right-click in the **ALL** section and select **New > Local Input**.) This opens the **Add input** dialog with a list of existing input models.
 - ii. Select **Account** and click **OK**. This displays the Account complex type in the Inputs section of the SameAccount filter.

Adding a REPLACEALL function to 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 > Function**. This opens the **New Function** dialog.
2. Expand **Text**, select **REPLACEALL** and click **OK**.
3. Connect "Card Number" in the Customer Details input model to "String" in the REPLACEALL function. This displays an arrow going from "Card Number" to "String".

4. 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 "Regex" of REPLACEALL and select **Set Constant Value**. This opens the **Set constant value** dialog.
 5. Type "-" and click **OK**. This causes "-" to be displayed for Regex.
 6. 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 "Replacement" of REPLACEALL and select **Set Constant Value**. This opens the **Set constant value** dialog.
 7. Type "" and click **OK**. This causes "" to be displayed for Replacement.
-

Adding an EQUALS function to the SameAccount filter

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 > Function**. This opens the **New Function** dialog.
2. Expand **Logic**, select **EQUALS** and click **OK**. The EQUALS function is displayed in the ALL section.
3. Connect "Result" in the REPLACEALL function to "Arg1" in the EQUALS function. This displays an arrow going from "Result" to "Arg1".
4. Connect "CardNumber" in the Account input model to "Arg2" in the EQUALS function. This displays an arrow going from "CardNumber" to "Arg2".
5. 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 pane. This displays an arrow going from "Result" to the boolean element.
6. If the condition is met, that transaction record will be stored in the any element of Value Output. Notice how "Customer Details" in the Inputs section is already automatically mapped to the any element in the Value output pane. Do not adjust this.

Completing mappings for the SameAccount filter


99% of the filter is now complete. In the AccountTxns to Statement local transformation, "Customer Details" (under Transactions) in the Inputs section is already automatically mapped to "Customer Details" in the SameAccount filter. Similarly, "Value" in the SameAccount filter is already automatically mapped to "Customer Details" in the Record to StmtLine local transformation.

To complete the filter mappings, 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.

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

Creating the FindCustomerRecord filter


Follow these steps to create the FindCustomerRecord filter:

1. Click the AccountTxns to Statement tab.
2. Click the arrow that is between "Customer" (under Customers File) in the Inputs section and "Customer" in the Populate NameAndAddress local transformation. This highlights the arrow.
3. Right-click the highlighted arrow and select **Add Filter** from the context menu. This opens a **Filter Customer** tab for the filter (with a  icon beside its name) within the AccountTxns to Statement tab.

Notice how the Inputs section of the filter tab is automatically populated with the Customer input type. Notice also how it is automatically mapped to the Value pane in the Outputs section.

Note: Customer will be the first input element to be involved in the comparison.

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

5. Now select the input model that will contain the second element to be involved in the comparison. For the purposes of this comparison select the Accounts model, as follows:
 - iii. Click the  (**New Local Input**) icon. (Alternatively, right-click in the **ALL** section and select **New > Local Input**.) This opens the **Add input** dialog with a list of existing input models.
 - iv. Select **Account** and click **OK**. This displays the Account complex type in the Inputs section of the FindCustomerRecord filter.

Adding an EQUALS function to the FindCustomerRecord filter

To enable the comparison between the two input models, use a logic function called EQUALS. Follow these steps to create the EQUALS function:

1. In the ALL section, right-click and select **New > Function**. This opens the **New Function** dialog.
2. Expand **Logic**, select **EQUALS** and click **OK**. The EQUALS function is displayed in the ALL section.
3. Connect "Customer Number" in the Customer input model to "Arg1" in the EQUALS function. This displays an arrow going from "Customer Number" to "Arg1".
4. Connect "Customer" in the Account input model to "Arg2" in the EQUALS function. This displays an arrow going from "Customer" to "Arg2".
5. 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 pane. This displays an arrow going from "Result" to the boolean element.
6. If the condition is met, that customer record will be stored in the any element of Value Output. Notice how "Customer" in the Inputs section is already automatically mapped to the any element in the Value output pane. Do not adjust this.

Completing mappings for the FindCustomerRecord filter

99% of the filter is now complete. In the AccountTxns to Statement local transformation, "Customer" (under Customers File) in the Inputs section is already automatically mapped to "Customer" in the FindCustomerRecord filter. Similarly, "Value" in the FindCustomerRecord filter is already automatically mapped to "Customer" in the Populate NameAndAddress local transformation.

To complete the filter mappings, connect "Account" in the Inputs section to "Account" in the FindCustomerRecord filter. This displays an arrow going from the Account input model to "Account" in the FindCustomerRecord filter

Select **File > Save > Save Tab As** and click `StatGen.tfd` to populate it in the **File name** field. Then navigate to the `My IONA Projects/Getting Started/Samples/Creating Transformations` folder, double click `Adding Filters`, and click **Save**. This saves the updated transformation into the `Adding Filters` folder.

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 116](#).

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. If you need to open it, you may do so by selecting **File > Open Project** from the menu bar.
2. Navigate to the `My IONA Projects/Getting Started/Samples/Creating Transformations/Adding Filters` folder
3. Right-click the `StatGen.tfd` file and select **Open Selected**. This opens the `StatGen.tfd` transformation in the main view of the workbench.
4. Click the Record to StmtLine tab.
5. Right-click in the ALL section of the Record to StmtLine local transformation and select **New > Java Method**. This opens the **Java Method** dialog.
6. Click the Signature tab.
7. Type "CreateNarrative" in the **Method Name** field.
8. In the Parameters section, click the  icon to add a new parameter row.
9. Type "vendorID" in the **Name** column.

10. Click anyType in the **Type** column. This opens the **Select Argument Type** dialog.
11. Expand **Numeric**, select **long** and click **OK**.
12. In the Return Type section, click **Select**. This opens the **Select Return Type** dialog.
13. Expand **Text**, select **String** and click **OK**.
14. Click the Code tab at the top of the dialog. The method declaration is displayed.
15. Type `return "Transaction from vendor:"+vendorID;` and click **OK**. The CreateNarrative method is displayed in the ALL section.
16. 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".
17. 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".
18. Select **File > Save > Save Tab As** and navigate to the `My IONA Projects/Getting Started/Samples/Creating Transformations/Adding Java Methods` folder.
19. 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 **StmntLine** 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 will be finally complete, so the transformed statement will be valid. 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 122](#).

Steps

Follow these steps to use introspect functions in a transformation:

1. In the Project view of the workbench, ensure that `MyProject.iop` is opened. If you need to open it, you may do so by selecting **File > Open Project** from the menu bar.
2. Navigate to the `My IONA Projects/Getting Started/Samples/Creating Transformations/Adding Java Methods` folder
3. Right-click the `StatGen.tfd` file and select **Open Selected**. This opens the `StatGen.tfd` transformation in the main view of the workbench.
4. Click the AccountTxns to Statement tab.
5. Right-click in the ALL section and select **New > Introspector**.

Note: If you have not disabled tool tips, a tool tip is displayed at this point. It will prompt you to first map the input type of the introspect function and then double click on it to specify the return type, which will enable you to map its output.


The Introspect function is displayed in the ALL section, with "Arg1" as its input and "Result" as its output.

6. Connect "Value" in the FindCustomerRecord filter to "Arg 1" in the Introspect function. This displays an arrow going from "Value" to "Arg 1".
7. Double click on "Arg 1" of Introspector. This opens the **Select Path** dialog.
8. Select the Customer complex type and click **OK**. This displays "Customer" as Arg 1 of the Introspect function.
9. Double click on "Result" of Introspector. This opens the **Select Path** dialog.
10. Select the "Country Of Residence" element and click **OK**. "Country Of Residence" is now displayed as Result of the Introspect function.
11. Right-click in the **ALL** section and select **New > Function**. This opens the **New Function** dialog.
12. Expand **Text**, select **CONCAT**, and click **OK**. The CONCAT function is displayed in the ALL section.
13. Connect "Country Of Residence" in the Introspect function to "Arg 1" in the CONCAT function. This displays an arrow going from "Country Of Residence" to "Arg 1".
14. Connect "AccountNumber" in the Account input model to "Arg 2" in the CONCAT function. This displays an arrow going from "AccountNumber" to "Arg 2".
15. Connect "Result" in the CONCAT function to "Account" (under Hdr) in the Statement output model. This displays an arrow going from "Result" to "Account".
16. Select **File > Save > Save Tab As** and click `StatGen.tfd` to populate it in the **File name** field. Then navigate to the `My IONA Projects/Getting Started/Samples/Creating Transformations` folder, double click `Adding Introspect Functions`, and click **Save**. This saves the updated transformation into the `Adding Introspect Functions` folder.

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. Expand **Statement** and then expand **Hdr** for one or all records available. Notice how Account is now different from what it was before. It now has 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 130
Deployment	page 130
Deployments directory	page 130

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.