

Progress[®] Artix[®] Data Services

User Guide

Version 3.9, May 2009

© 2009 Progress Software Corporation and/or its affiliates or subsidiaries. All rights reserved.

These materials and all Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation and/or its affiliates or subsidiaries. The information in these materials is subject to change without notice, and Progress Software Corporation and/or its affiliates or subsidiaries assume no responsibility for any errors that may appear therein. The references in these materials to specific platforms supported are subject to change.

Actional, Actional (and design), Allegrix, Allegrix (and design), Apama, Apama (and Design), Artix, Business Empowerment, DataDirect (and design), DataDirect Connect, DataDirect Connect64, DataDirect Technologies, DataDirect XML Converters, DataDirect XQuery, DataXtend, Dynamic Routing Architecture, EasyAsk, EdgeXtend, Empowerment Center, Fathom, IntelliStream, IONA, IONA (and design), Mindreef, Neon, Neon New Era of Networks, ObjectStore, OpenEdge, Orbix, PeerDirect, Persistence, POSSENET, Powered by Progress, PowerTier, Progress, Progress DataXtend, Progress Dynamics, Progress Business Empowerment, Progress Empowerment Center, Progress Empowerment Program, Progress OpenEdge, Progress Profiles, Progress Results, Progress Software Developers Network, Progress Sonic, ProVision, PS Select, SequeLink, Shadow, ShadowDirect, Shadow Interface, Shadow Web Interface, SOAPscope, SOAPStation, Sonic, Sonic ESB, SonicMQ, Sonic Orchestration Server, Sonic Software (and design), SonicSynergy, SpeedScript, Stylus Studio, Technical Empowerment, WebSpeed, Xcalia (and design), and Your Software, Our Technology-Experience the Connection are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries.

AccelEvent, Apama Dashboard Studio, Apama Event Manager, Apama Event Modeler, Apama Event Store, Apama Risk Firewall, AppAlive, AppServer, ASPen, ASP-in-a-Box, BusinessEdge, Cache-Forward, DataDirect Spy, DataDirect SupportLink, FUSE, FUSE Mediation Router, FUSE Message Broker, FUSE Services Framework, Future Proof, Ghost Agents, GVAC, High Performance Integration, Looking Glass, ObjectCache, ObjectStore Inspector, ObjectStore Performance Expert, OpenAccess, Orbacus, Pantero, POSSE, ProDataSet, Progress ESP Event Manager, Progress ESP Event Modeler, Progress Event Engine, Progress RFID, PSE Pro, SectorAlliance, SeeThinkAct, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, Sonic Business Integration Suite, Sonic Process Manager, Sonic Collaboration Server, Sonic Continuous Availability Architecture, Sonic Database Service, Sonic Workbench, Sonic XML Server, StormGlass, The Brains Behind BAM, WebClient, Who Makes Progress, and Your World. Your SOA. are trademarks or service marks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and other countries.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. Any other trademarks contained herein are the property of their respective owners.

Updated: May 13, 2009

Table of Contents

Welcome to Artix Data Services	6
What's New in ADS 3.9	7
Quick Start	8
The Basics	8
Launching ADS Designer	10
Using Designer for the First Time	11
Creating Projects	13
Downloading Getting Started Samples	16
Opening a Sample Data Model	17
Building Instances of a Data Model	19
Finding Generated Source and Compiled Code	21
Using Ant	24
Finding Components In ADS Designer	26
Designer Preferences	29
Windows	36
The Home Page	36
The Main Window	38
The Project Window	41
The Explorer Window	42
The Properties Window	44
The Messages Window	46
The Ant Build Window	47
The Find Window	48
The Linkage Errors Window	51
The Overview Window	52
Menus and Toolbars	54
Toolbar	54
Data Model Palette	55
Transform Palette	56
File Menu	57
Edit Menu	61
View Menu	65
Find Menu	69
Tools Menu	72
Deploy Menu	79
Window Menu	81
Help Menu	82

Projects	85
Creating a Project	86
Paths	87
Properties	88
Profiles	91
Profile Settings	92
Profile Properties	93
Namespaces	105
Classpath	107
Preferred Aliases	108
Data Models	109
Creating a Data Model Manually	110
Adding Components to a Data Model	112
Model Components	113
Simple Data Types	114
Atomic Simple Types	115
List Simple Types	118
Union Simple Types	120
Built-in Data Types	121
Complex Data Types	127
Data Components	129
Elements	130
Element Groups	132
Any Elements	134
Attributes	135
Attribute Groups	137
Any Attributes	139
References	140
Validation Components	142
Enumerations	143
Factory Lookups	145
Validation Rules	146
Cardinality	153
Includes, Imports and Redefines	154
Namespaces	155
Preferred Aliases	156
Classification Groups	157
Annotations	158
SWIFT Field Data Types	161
Importing Data Models from Other Formats	162
Importing from an XML Schema	163
Importing from a DTD	165
Importing from a WSDL File	166
Importing from an XML Instance	167
Importing from a Text File	167

Importing from a RELAX NG (XML) Schema	172
Importing from a RELAX NG (Compact) Schema	173
Importing from a Java Class	174
Importing from a Database	176
Importing from an Excel Spreadsheet	186
Using a Mapping File	189
Exporting Models to Other Formats	190
Exporting to an XML Schema	191
Exporting to a DTD	193
Exporting to a RELAX NG (XML) Schema	196
Exporting to a RELAX NG (Compact) Schema	198
Exporting to HTML	200
Document Roots	204

Transformations **206**

Creating a Transformation	207
Adding Models to a Transformation	212
Adding Components to a Transformation	213
Adding Functions to a Transform	214
Adding a Transformation Reference	216
Adding a Local Transformation	217
Adding Filters to a Transform	218
Adding Java Methods to a Transform	221
Adding Hashtable Functions to a Transform	224
Adding Introspect Functions to a Transform	226
Adding Instantiate Functions to a Transform	228
Setting up translations	229
Using Curved Connectors	230
Finding Unmapped Components	231
Automatically Aligning Transformation Components	233
Finding Components in a Transformation	234
Globalizing a Local Transformation	235
Moving Transformation Mappings	236
Exporting HTML from Transformations	237
Collaborating on Transformations	238

Model and Transformation Properties **239**

General Properties	242
General Advanced Properties	248
Presentation Properties	255
Presentation Advanced Properties	258
Validation Properties	263
Validation Advanced Properties	268
XML Properties	270
Database Properties	275
SWIFT Properties	277
Transform Properties	278

Setting Properties using XPath	280
Verifying, Building, and Running	281
Verifying Data Models and Transformations	282
Building Models, Components, and Transformations	283
Running Model Components and Transformations	284
Creating a Run Configuration	285
Working with Data Models in the Run Tab	288
Loading Data into an Object	289
Creating Object Instances	293
Loading Model Changes	296
Setting Logging Levels	297
Using the Text View	298
Using Advanced Settings	304
Working with Transformations in the Run Tab	309
Packaging	312
Creating a Package	313
Unpacking a Package	314
The Diff Tool	315
Loading and Working with the Diff Tool	316
Menus and Toolbars	317
File Menu	317
Options Menu	320
Go To Menu	321
Window Menu	322
Help Menu	323
Context Menu	324
Samples and Standards Libraries	326
Examples and Reference Implementations	326
Running the Samples	327
Demonstrated Functionality	328
Standards Libraries	330
Frequently Asked Questions	331
API	331
How Do I Configure API Logging?	331
How Do I Read XML Data in the API?	333
How Do I Validate a Complex Data Object?	334
How Do I Use Artix DS And XPath?	335
How Do I Write XML Data in the API?	337
How Does the SWIFT Pre-Parser Work?	338
What Are Bean Classes and Interfaces?	339
Why Do I Get an OutOfMemoryError in Log4J?	341

Data Models	343
How Do I Edit a Model's Namespaces?	343
How Do I Set the Target Namespace of a Data Model?	344
Building	345
How Can I Build Via Apache Ant?	345
Why Does Windows Not Like the Length of My Generated Class Names?	346
Examples	347
Where Are the Examples Located?	347
General	348
How Do I Configure Java System Properties?	348
How Do I Configure the Designer Classpath?	349
How Do I increase Designer's Memory?	350
How Is The ADS_HOME Environment Variable Set Up?	351
Is There a Getting Started Guide for ADS?	352
Metadata	353
How Do I Create a Data Model from an XML Schema?	353
How Do I Generate an XML Schema from a Data Model?	354
Projects	355
How Do I Add a New Path to a Project?	355
Using ADS with Other Products	356
DataXtend SI Integration	356
Apache Camel Integration	357
Index	359

Welcome to Artix Data Services

This user guide aims to provide a technical introduction to Progress Artix Data Services (ADS).

ADS is an integration tool for developers, technical architects, and metadata management architects. It provides a development platform that enables you to model complex data structures and build Java code directly from them.

ADS is mainly used in integration and messaging projects where you need to manipulate and extend complex data structures based on industry-standard message formats, proprietary APIs, XML schema and RDBMS. It generates Java code components that can read, write, parse, validate and transform 'instances' of the data structures that are described by the originating models.

In addition, ADS provides pre-built and maintained standards libraries. These are packages of industry-standard data model implementation libraries and support services. Each library consists of syntactic/semantic implementations of, and data model constraint rules for, standards such as SWIFT FIN, FpML, FIX, CREST DEX and ISO, and TRAX. The standards libraries include a full ADS Enterprise license with support services, including remedial product support, upgrades and updates, and data model updates/maintenance based on the published standards by the relevant standards bodies.

If you are new to ADS, you should take a few moments to read through the [Quick Start](#) section.

If your main interest is in using the compiled code components within your own applications, you should concentrate on the [Examples](#) section.

Contact Us

Contact details can be found in the **Help > Contact Us** menu in the ADS Designer.

Support Services

Support services can be found in the **Help > Technical Support** menu in the ADS Designer.

What's New in ADS 3.9

The following features are new in ADS 3.9:

Progress DataXtend SI Support

ADS now ships with a plug-in that you can add to Progress [DataXtend Semantic Integrator](#) 8.4, so allowing you to import an ADS data model into the DataXtend SI Workbench.

Additional Metadata Added to HTML Export Feature

The output that results from exporting a data model to [HTML](#) has been improved to include the content of validation rules.

Excel Spreadsheet Support

You can import a data model from a Microsoft [Excel](#) spreadsheet created in the Office Open XML format using Excel 2007 or later.

XQuery Added as Validation Rule Option

You can now specify an XQuery as a [validation rule](#) in ADS Designer.

Transformation Collaboration

Developers can now [collaborate](#) more easily on the same transformation file. The XML in transformations has been changed to allow for easier merging using source control management tools.

The Basics

This topic provides an overview of the following concepts:

- [Artix Data Services](#)
- [Data Models](#)
- [Data Components](#)

Artix Data Services

Artix Data Services (ADS) is a Java developer toolkit for defining, managing and maintaining complex data models and generating Java code directly from them. Models are defined in terms of simple and complex data types, in much the same way as XML Schema, and are used to generate code that can be used to parse, validate and transform conformant data.

ADS provides a powerful GUI where you can define models from scratch, or acquire them automatically from existing XML DTDs or schemas, relational database catalogs, or Java objects. It also ships with several pre-built libraries that represent ready-made implementations of common financial messaging and data standards, such as SWIFT, FIX, CREST, and TRAX.

You can also use ADS as an abstract, format-agnostic repository for your data models and then express them in a variety of ways for external consumption - for example by exporting them as XML Schema or DTD, RDBMS DDL, or as an XML document.

You use ADS to generate code from data models as follows:

1. Model your data, by creating or importing data models that describe the characteristics of the different data objects and their relationships to each other. Data models are organized within projects.
2. Within a project, create one or more sets of deployment characteristics (profile settings) that govern the behavior of the resulting code.
3. Deploy the model, or elements of the model. This generates Java code that describes all or part of your data model and knows how to read and write (parse and format) instances of compliant data.
4. Use the generated code in your own application to perform the required tasks; for example to read XML documents from files and write them to database tables, or to parse incoming SWIFT messages and present them in application-specific formats on a JMS queue.

There are also reference implementations that show how to use deployed ADS code as the basis for dynamically creating fully validating user interfaces, either as Java applications or using JSP/servlets. You can download these from the ADS Designer Home page.

Data Models

An ADS data model is an object-oriented model of a collection of data. It is analogous to a W3C XML Schema, and many of the terms and concepts correspond almost directly with those used in XML Schema. However, ADS also includes items covered by Schema, such as semantic validation rules.

Data models are mostly abstracted away from the specifics of presentation (format/syntax) and transport, although these layers exist in the ADS architecture in their own right.

Data Components

Data models are composed of *data components*, of which there are ten basic types:

Simple Data Types

Built-in types such as strings, integers, dates, booleans, and types that are derived from them. Simple types are the basic building blocks from which other types are constructed and derived, and are further subdivided according to whether they are 'atomic', or represent a list or union.

Complex Data Types

Aggregations of other types (simple and/or complex) adhering to a specified content model, and derivations thereof.

Elements

Occurrences or instances of data types.

Attributes

Objects associated with a complex type that are not children of that type.

Element Groups

Reusable, anonymous collections of previously declared elements.

Attribute Groups

Reusable, anonymous collections of previously declared attributes.

Annotations

Textual information that you want to associate with a specific component.

Validation Rules

Semantic constraints that apply to the structures being described. They usually take the form of a conditional expression over the values of specific parts of the model.

Enumerations

The set of allowable values applicable to a particular data object. They are typically used where content needs to be validated against a static list of values, for example, currency codes.

Classification Groups


Branch nodes in the classification hierarchy that allow other components to be grouped in a logical way.

The rest of this Quick Start section walks through the essential features and functions available within ADS Designer.

Launching ADS Designer

To launch ADS Designer:

Windows:

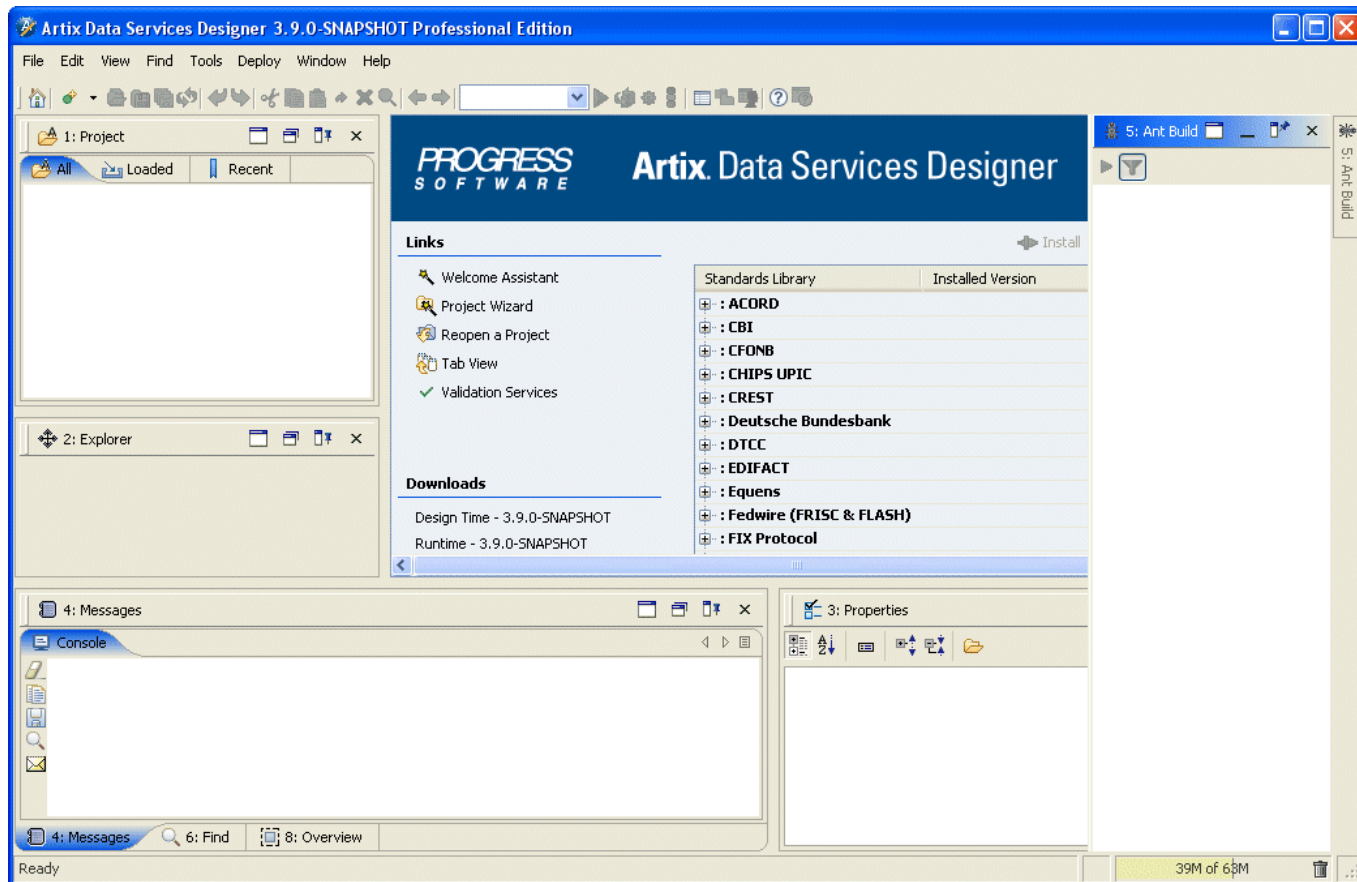
- Select **Programs > Progress > Artix Data Services > ADS Designer** from the Windows **Start** menu
- Double-click the  icon on your desktop (if you chose to add one during installation)
- Use Windows Explorer to navigate to your ADS installation directory and double-click **artix-ds-designer.exe**.

UNIX:

- Run the `artix-ds-designer` command from your ADS installation directory.

Using Designer for the First Time

On opening the ADS Designer for the first time, you will see the following screen:



The Designer user interface is divided into the following panels or windows:

Note: Where a window name is prefixed by a number, you can jump that that window by pressing Alt and pressing the corresponding number key.


- The central, untitled area is the **Main** window , where components are displayed when you open them.
- The **Project (Alt+1)** window displays the currently open project and contains a tree showing all the directory paths specified for that project.
- The **Explorer (Alt+2)** window is empty until a data model or transformation is selected in the Project window. It shows details of the various components that constitute the selected data model or transformation.
- The **Properties (Alt+3)** window is context-sensitive. It tabulates the characteristics of whatever ADS object is currently selected in the Project window, Explorer window or Tab.
- The **Messages (Alt+4)** window contains tabs for displaying console messages, deployment and compilation feedback.
- The **Ant Build (Alt+5)** window displays a tree containing details of build files generated by the tool as a result of deploying projects/models/components.
- The **Find (Alt+6)** window displays the results of search operations.

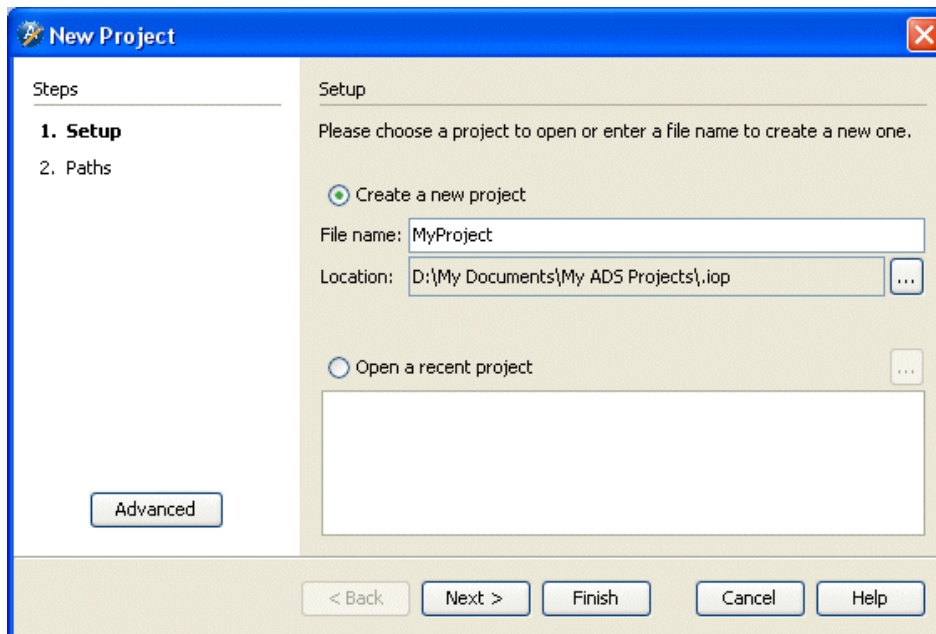
All windows can be docked or undocked, repositioned, and hidden or revealed according to preference.



Creating Projects

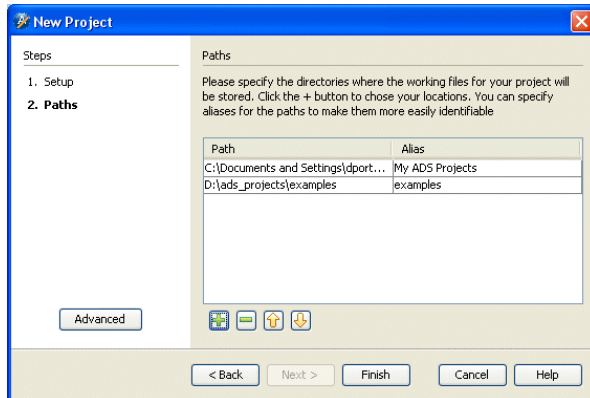
In ADS Designer, projects are used to store the data models, transformations and other working files for the tasks that you want to perform. You must create a project before you can perform any other task.

To create a project:

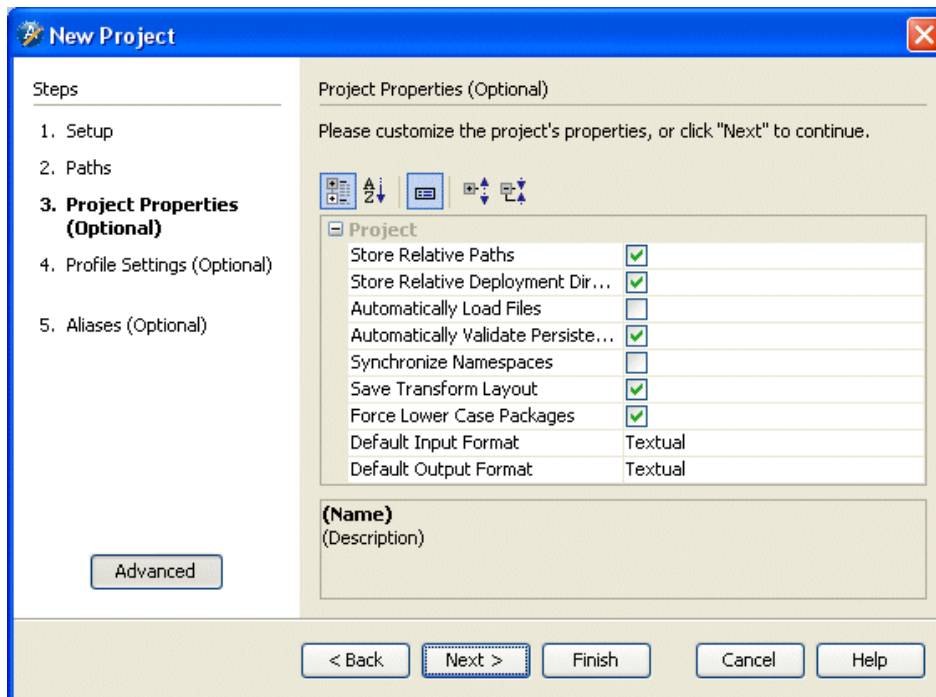
- Do one of the following:
 - Click the **New** drop-down icon in the toolbar and select **New Project**.
 - Select **File > New > Project** from the menu bar.
- In the Setup panel, type a unique project name in the **File name** field. The project name is added to the **Location** field and is assigned an .iop extension.
To select an alternative location for your project file, click the  button. Click **Next**.



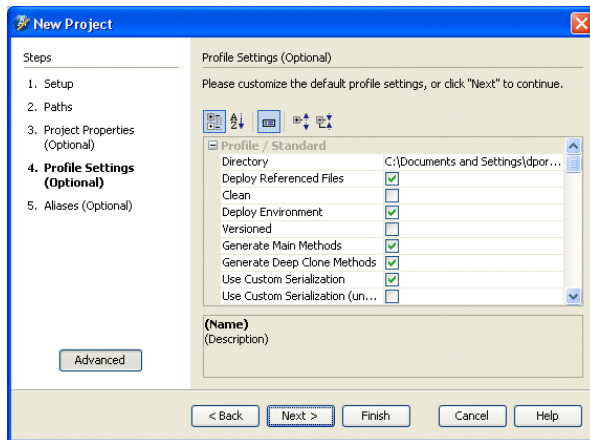
- In the Paths panel, set one or more file system locations where your ADS data models and transformations will be stored. You can set as many paths as you wish. Use the  and  icons to add or delete paths. Each path is assigned an alias to make it more easily identifiable in Designer. In the example shown, the second path relates to the examples directory that holds the various examples you can download as part of your ADS installation. In this case, paths to the My ADS Projects and examples folders will be displayed in the Project window under the project name.



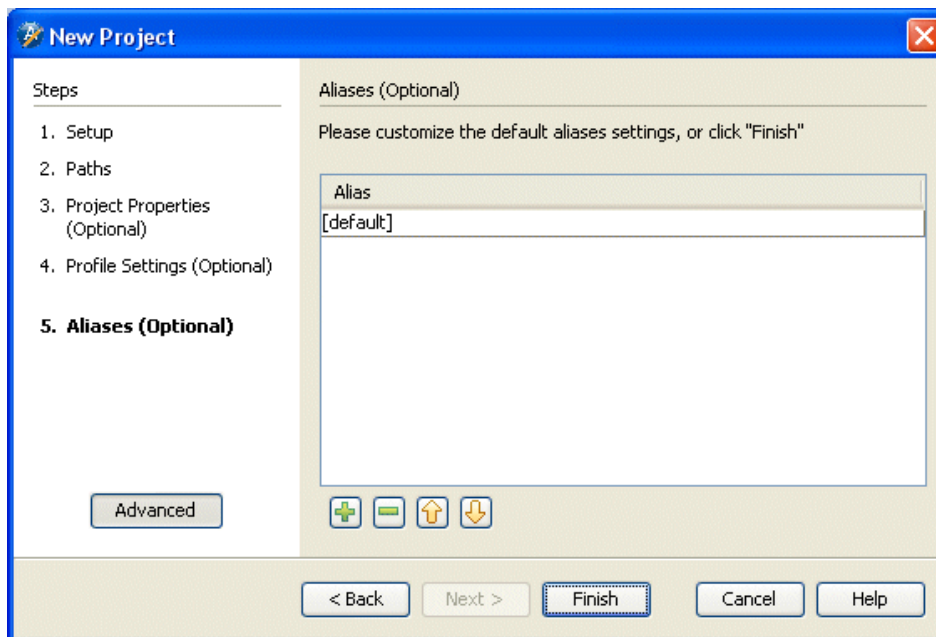
4. If you clicked the **Advanced** button, the Project Properties panel allows you to customize your project properties. You are not bound by the properties you set at this stage. You can accept the default values and edit them later within Designer. Click **Next**.



5. The Profile Settings panel allows you to set code style, versioning and the location where generated code is deployed. You are not bound by the profile settings you set at this stage. If you prefer, accept the default values and edit them later. Click **Next**.



6. The Aliases panel allows you to use different language versions of the same models. This is useful if you want to create internationalized models where different sets of names in different languages are defined for the same components. Using preferred aliases allows you to easily switch from one set of names to another. For example, you might want to display English or French, or technical or business, names in a data model. You can change these aliases in Designer whenever you like and are not bound by any aliases you set at this stage.



7. Click **Finish** to create your project. The project opens using the default layout.

Downloading Getting Started Samples

ADS provides a series of sample models and transformations to help you become familiar with the product. You can download these sample files as a plug-in to your ADS installation.

Click the **Getting Started** link in the Downloads section of the ADS Designer [Home page](#) to download the sample files to your machine. The files are downloaded to the following default locations:

Windows

C:\Documents and Settings*username*\My Documents\My ADS Projects\Getting Started\Samples

UNIX

/userhome/MyADSProjects/Getting Started/Samples

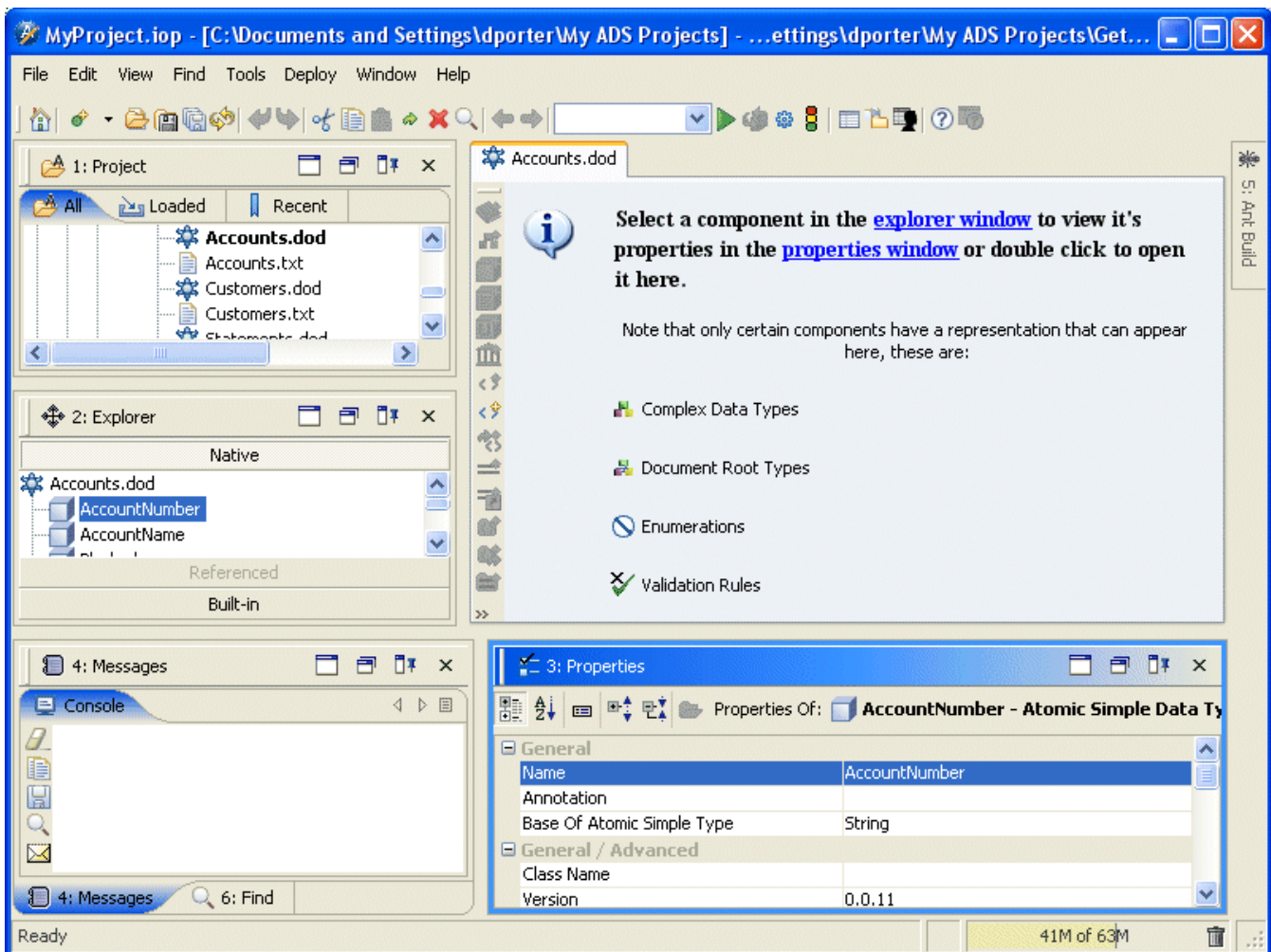
Opening a Sample Data Model

Double-click a data model (.dod file) in the Project window to load the model and display it in the Explorer window.

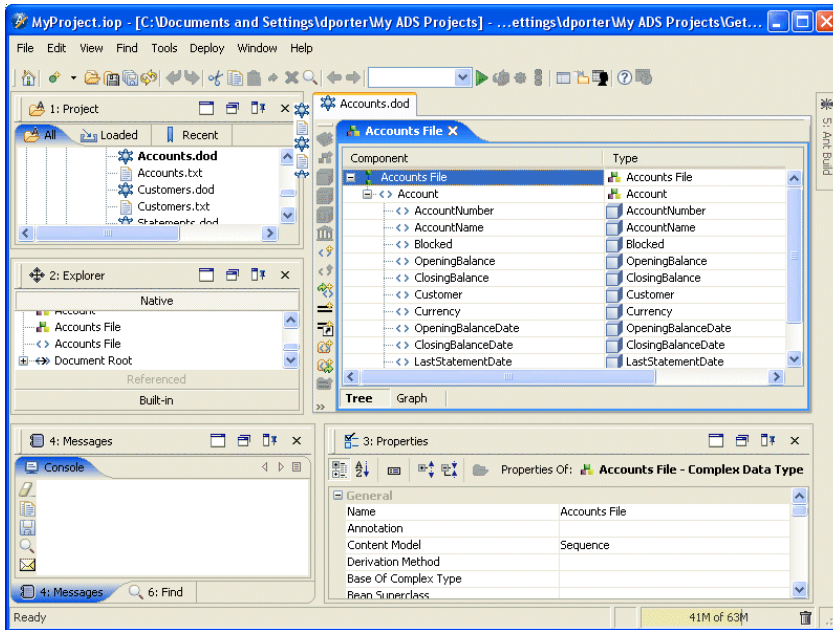
The components that constitute the model are shown in the Explorer window, which is divided into the following sections:

- Native
- Referenced (relating to types imported or included from other models)
- Built-in (which lists all available built-in data types).

In the example below, the simple data type 'AccountNumber' is selected in the Explorer window and its properties are displayed in the Properties window.



In the following example, the complex data type **Accounts File** is selected in the Explorer window. When you double-click a complex type, a tab opens containing the structure of the complex type within the data model tab. You can see an **Accounts File** tab opened within the **Accounts.dod** tab:



This complex type has a structure consisting of one to many **Account** objects, each of which contains a series of fields such as **Account Number**, **Account Name**, and so on.

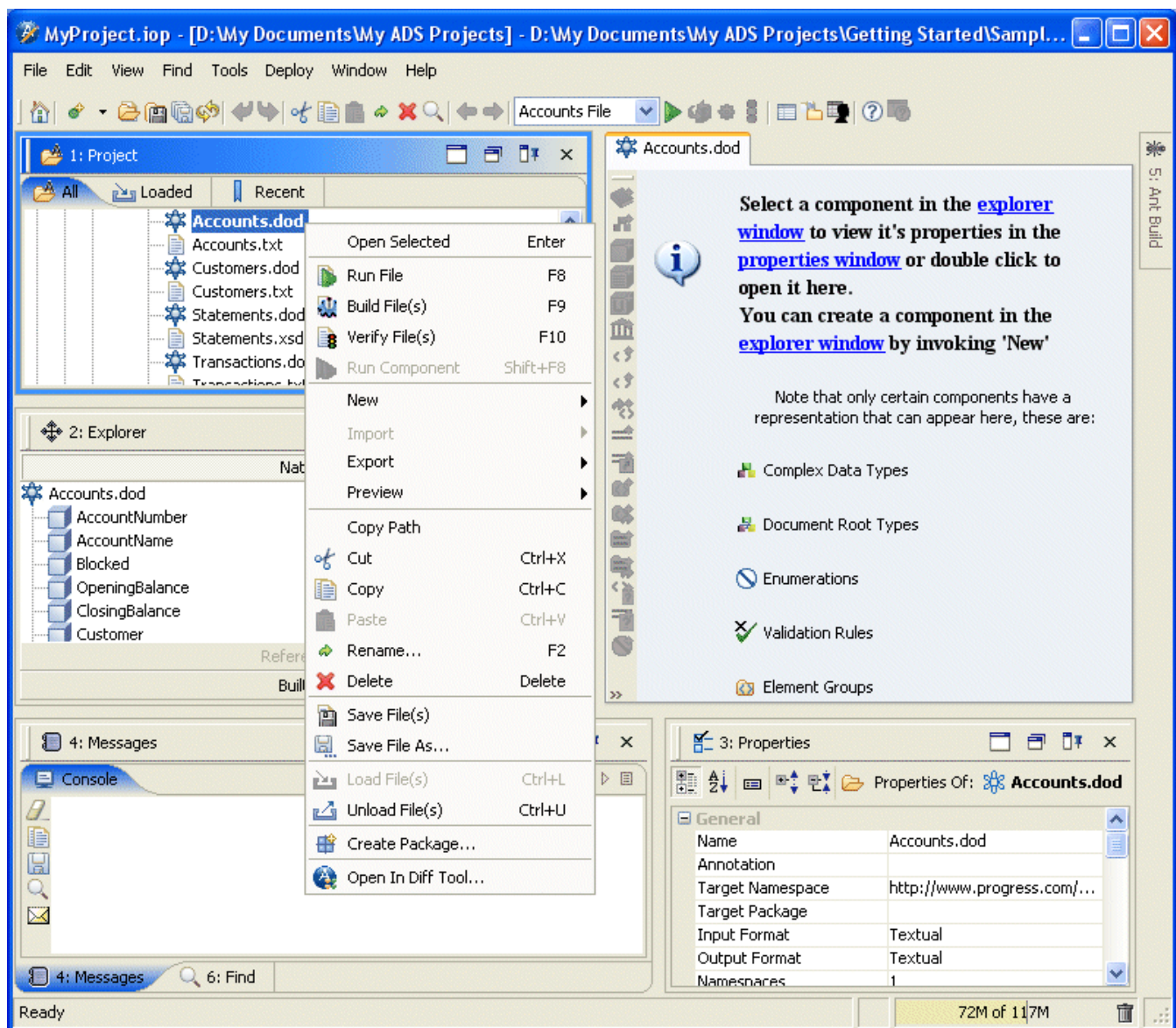
The next step is to build Java class instances of your data model.

Building Instances of a Data Model

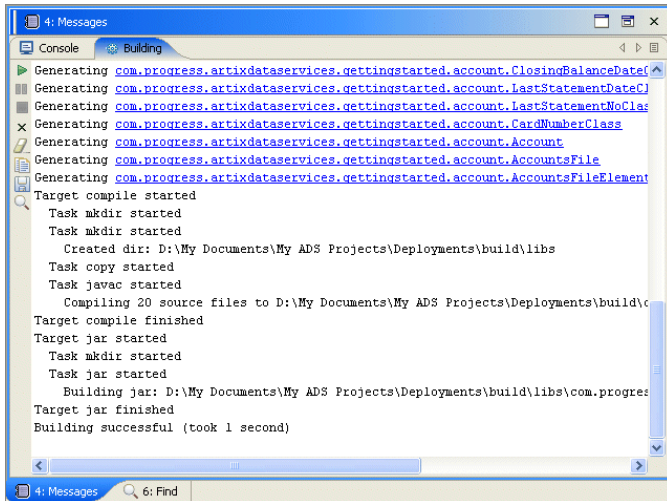
Data models are abstract descriptions of real-world data structures. You can generate code components from them that do useful things, such as read, write and validate your data. In ADS, this is known as [building](#) (or deploying) Java class instances of your data models.

To build the Java class instances of a data model, do one of the following:

- Right-click the data model in the Project window and select **Build File**.
- Right-click the open data model in the Explorer window and select **Build Component**.
- Select the open data model in the Explorer window and select **Deploy > Build Component** from the menu bar.



The Java code is generated and compiled for you. You can see the results in the Messages window.



```
4: Messages
Console Building
Generating com.progress.artixdataservices.gettingstarted.account.ClosingBalanceDate[
Generating com.progress.artixdataservices.gettingstarted.account.LastStatementDate[
Generating com.progress.artixdataservices.gettingstarted.account.LastStatementNoCla
Generating com.progress.artixdataservices.gettingstarted.account.CardNumberClass
Generating com.progress.artixdataservices.gettingstarted.account.Account
Generating com.progress.artixdataservices.gettingstarted.account.AccountsFile
Generating com.progress.artixdataservices.gettingstarted.account.AccountsFileElement
Target compile started
Task mkdir started
Task mkdir started
Created dir: D:\My Documents\My ADS Projects\Deployments\build\libs
Task copy started
Task javac started
Compiling 20 source files to D:\My Documents\My ADS Projects\Deployments\build\c
Target compile finished
Target jar started
Task mkdir started
Task jar started
Building jar: D:\My Documents\My ADS Projects\Deployments\build\libs\com.progres
Target jar finished
Building successful (took 1 second)
```

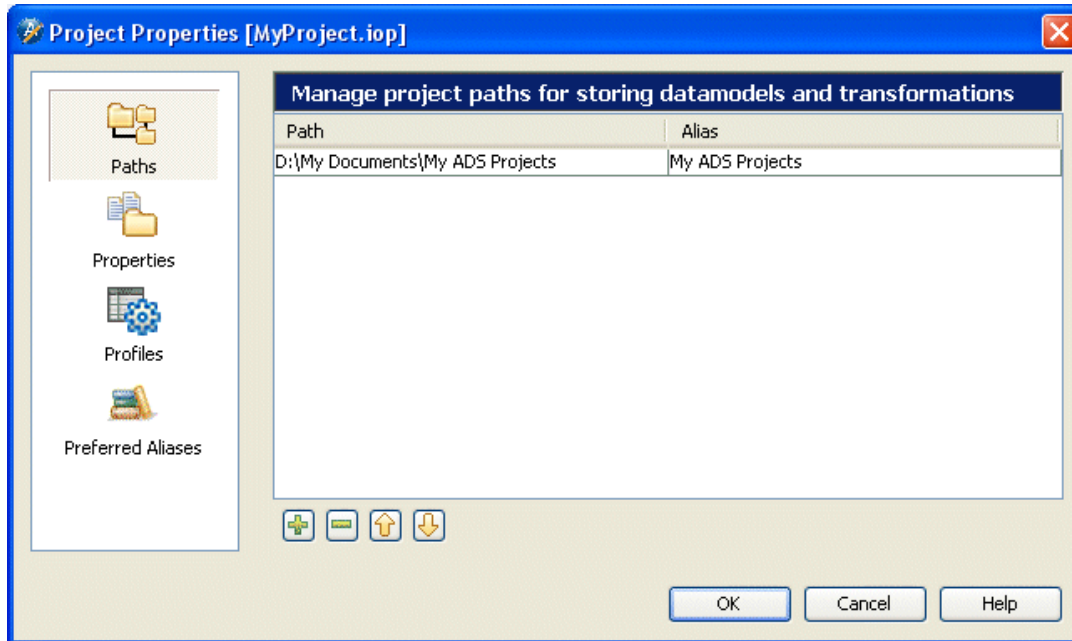
And an Ant build script is created and displayed in the [Ant Build window](#).

The next step is to [find the code](#) that you have built.

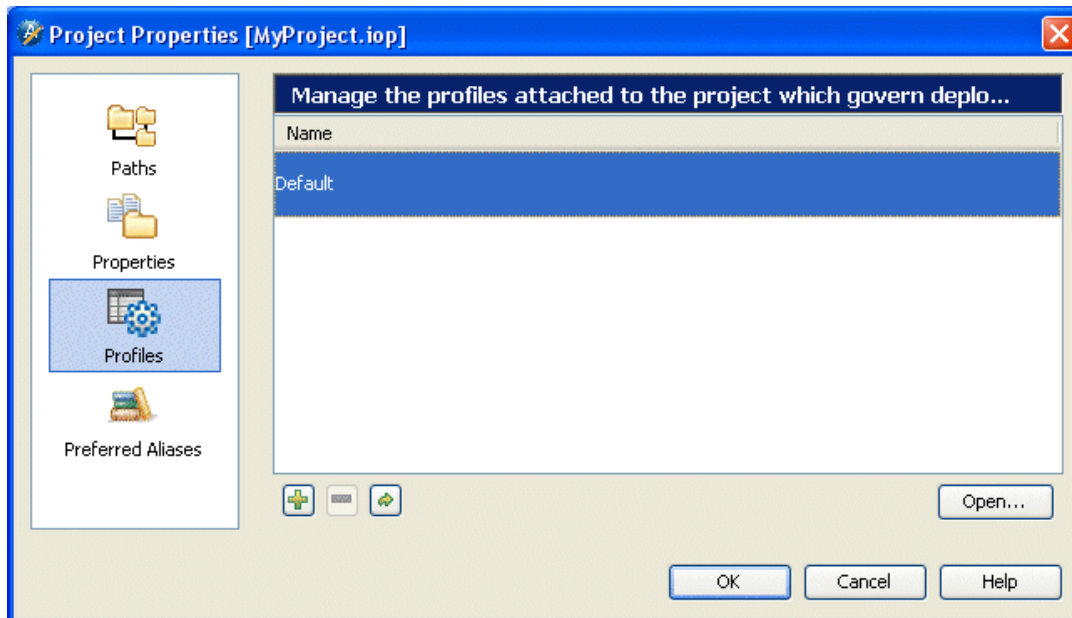
Finding Generated Source and Compiled Code

To find where the generated source and compiled code are written:

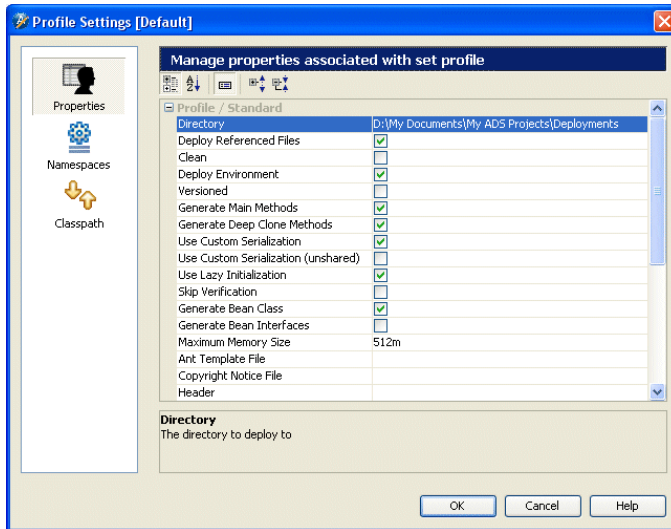
1. Select **Edit > Project Properties** from the menu bar or right-click root node of the tree in the Project window.
2. In the Project Properties dialog, click **Profiles**.



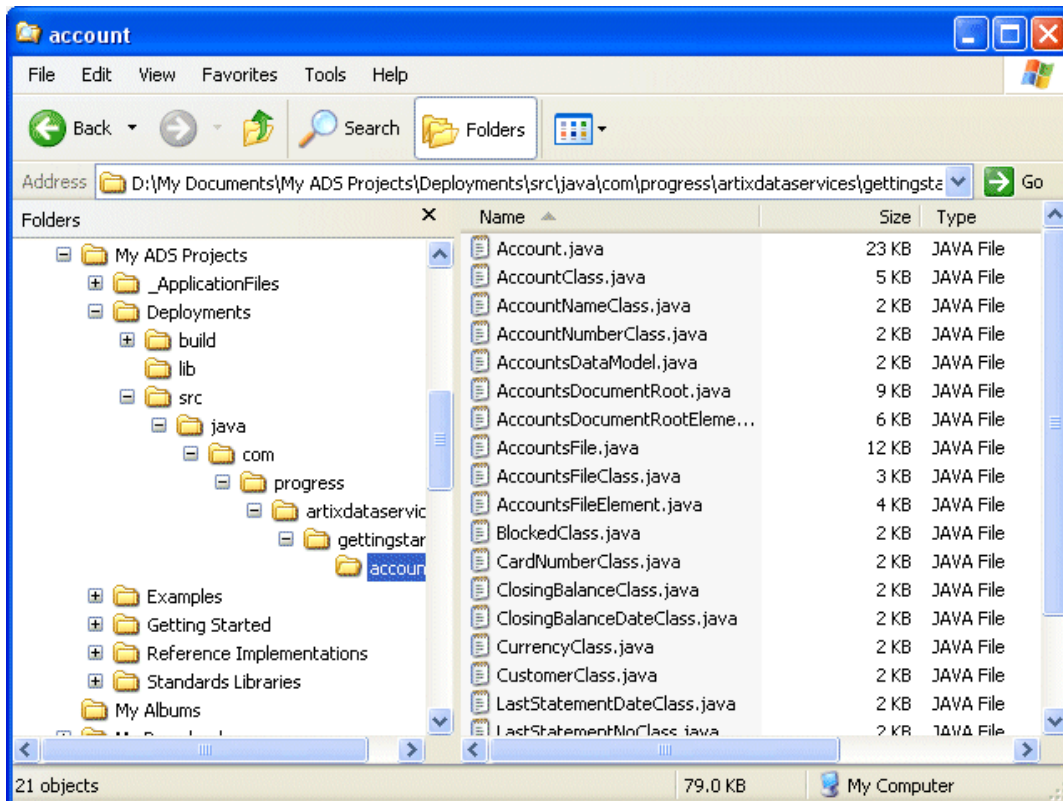
3. Then either double-click the **Default** descriptor or select it and then click **Open**.



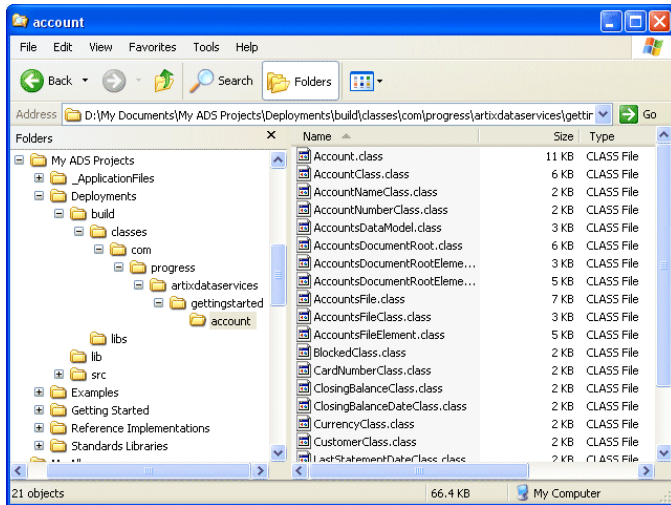
4. The first property in the Profile Settings dialog is the directory path defining the root of any deployment done from this project, using this profile. The default location is Deployments.



5. Look under this location you will find the generated Java source code in the src folder:



And the compiled code is written to the build/classes directory, while the JAR files are written to the build/libs directory.



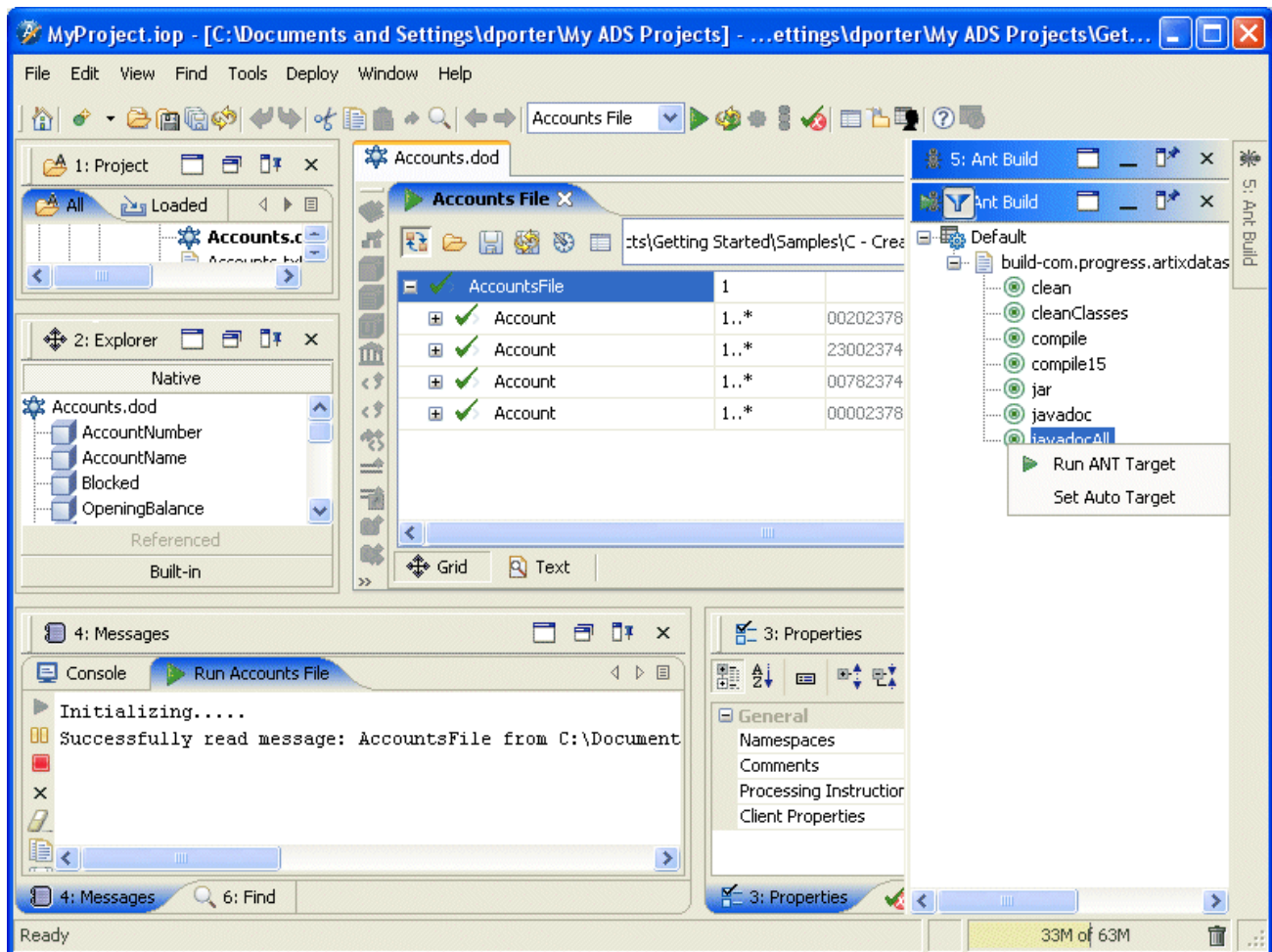
Using Ant

When you [build](#) a data model or component, a build.xml file for the Apache Ant build tool is created. The build file contains the following targets:

- clean
- cleanClasses
- compile
- compile15
- jar
- javadoc
- javadocAll

To run an Ant target:

1. In Designer, open the [Ant Build](#) window.
2. Expand the node for the getting.started.account.xml build file and run the javadocAll Ant target.



The Javadoc is generated into the docs/api directory off the deployment root:

The screenshot shows a web browser window with the following content:

Package [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV PACKAGE NEXT PACKAGE [FRAMES](#) [NO FRAMES](#)

Package
com.progress.artixdataservices.gettingstarted.account

Deployment of com.progress.artixdataservices.gettingstarted.account.

See: [Description](#)

Class Summary

Account	Account.
AccountClass	The Account complex data type.
AccountNameClass	The AccountName atomic simple data type.
AccountNumberClass	The AccountNumber atomic simple data type.
AccountsDataModel	The Accounts data

All Classes

- [Account](#)
- [AccountClass](#)
- [AccountNameClass](#)
- [AccountNumberClass](#)
- [AccountsDataModel](#)
- [AccountsDocumentRoot](#)
- [AccountsDocumentRootE](#)
- [AccountsDocumentRootE](#)
- [AccountsFile](#)
- [AccountsFileClass](#)
- [AccountsFileElement](#)
- [BlockedClass](#)
- [CardNumberClass](#)
- [ClosingBalanceClass](#)
- [ClosingBalanceDateClass](#)
- [CurrencyClass](#)
- [CustomerClass](#)
- [LastStatementDateClass](#)
- [LastStatementNoClass](#)
- [MyAtomicSimpleTypeClass](#)
- [OpeningBalanceClass](#)
- [OpeningBalanceDateClass](#)

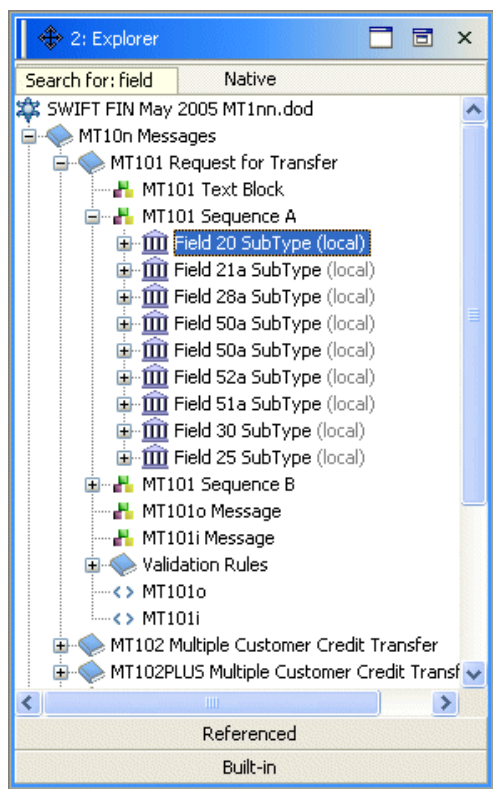
Finding Components In ADS Designer

ADS Designer includes functionality that makes it easier for you to find named components within lists, trees and tables. This is useful when you are working with complex data models. Below are some examples of how to use this functionality to quickly locate required data components.

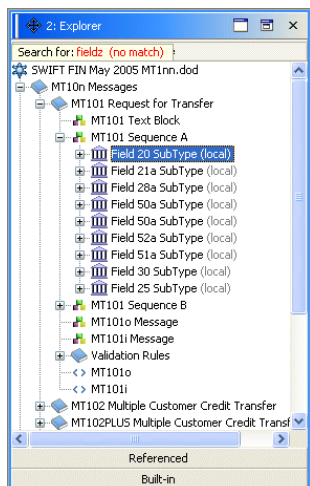
- [Searching for Specific Components](#)
- [Performing a Multiple Select](#)
- [Using Regular Expression Search Queries](#)

Searching for Specific Components

If you have a data model opened, as in the example below, you can navigate to a component by clicking the root node of the tree in the Explorer window and then typing the first few characters of the name of the component you are searching for.

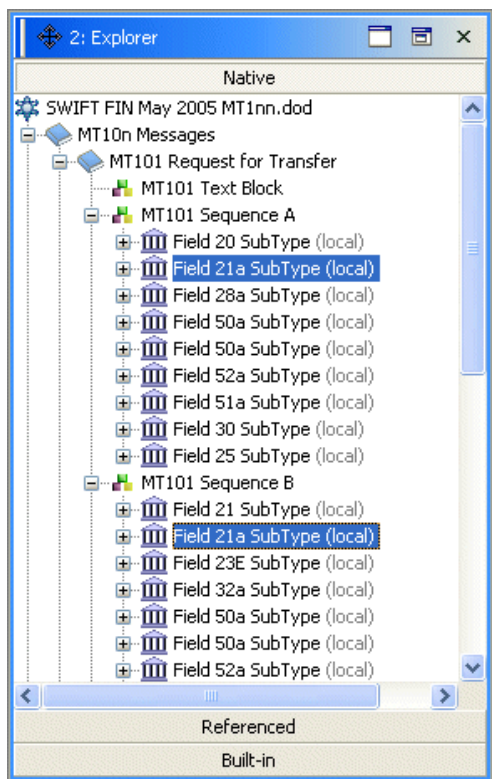


Doing this opens a text field showing the text you have entered so far. If components with names matching the entered text are present in the tree, the application's focus will move to the first matching occurrence. Pressing the down arrow key will bring you to the next matching occurrence and pressing the up arrow key will bring you to the previous matching occurrence. If no matching occurrence exists, the text field will highlight the input search terms in red as illustrated below.



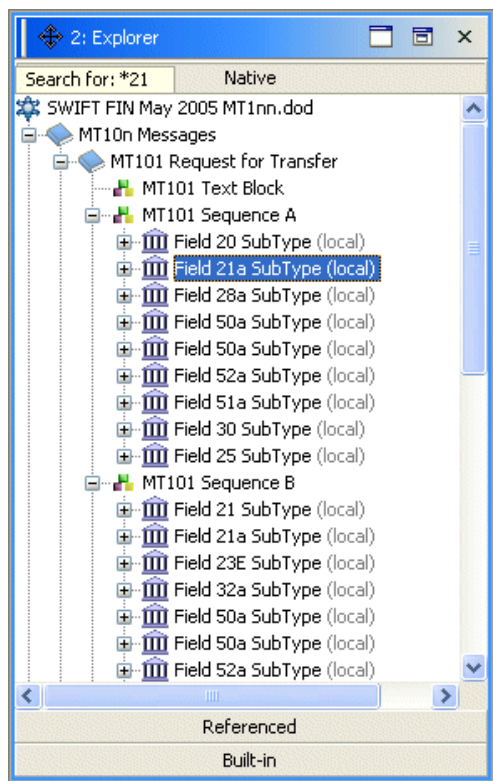
Performing a Multiple Select

After typing in a search parameter, it is possible to simultaneously select all matching components by pressing "A" or the up and down arrow keys while holding the **CTRL** key.



Using Regular Expression Search Queries

It is also possible to use regular expression search queries to look for components as illustrated below where entering *21 returns the first component whose name includes the number 21.



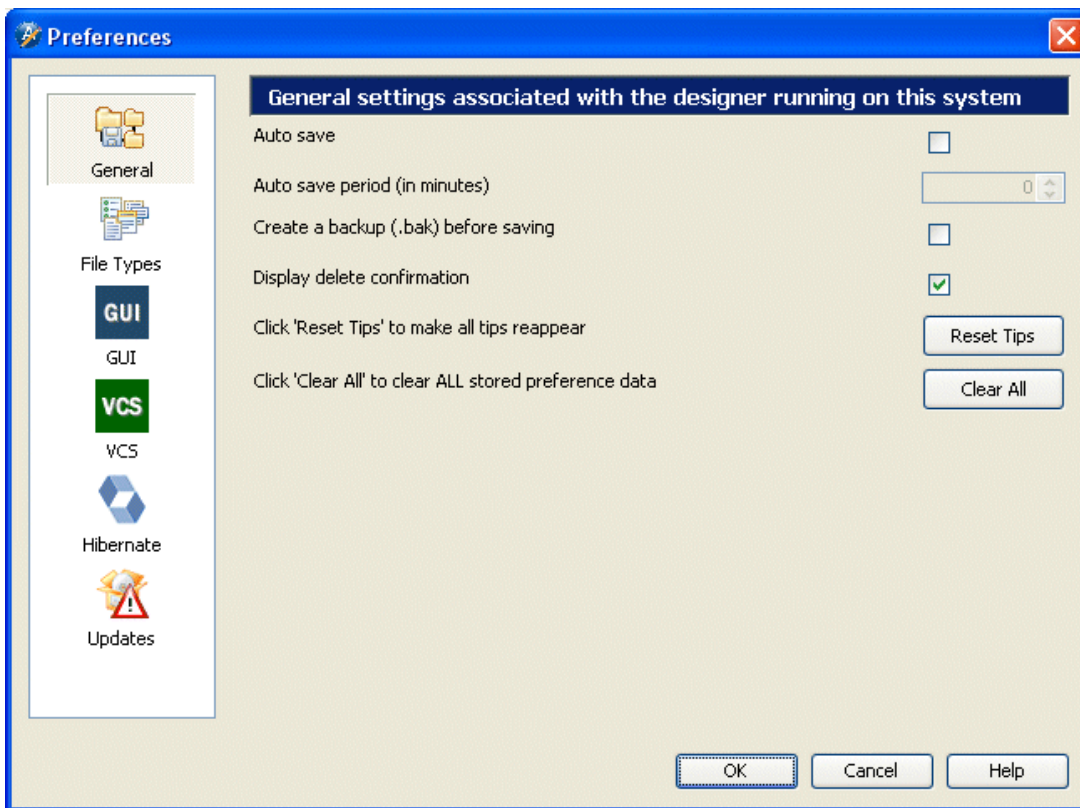
Designer Preferences

Before looking in detail at ADS, let's take a look at the various preferences that you can specify which will govern various aspects of the working and look-and-feel of the Designer.

Select **Edit > Preferences** from the menu bar to open the Preferences dialog, which is divided into six separate panels:

- [General](#)
- [File Types](#)
- [GUI](#)
- [VCS](#)
- [Hibernate](#)
- [Updates](#)

Preferences - General



The configurable preferences on the General panel are:

Auto save

Allows you to enable or disable automatic saves.

Auto save period

Allows you to set the period in minutes between saves, when auto save is enabled. Settings this value too low can degrade the performance of ADS, because a large amount of processing power is required to save some large data models.

Create a backup before saving

Allows you to enable or disable the creation of a .bak copy prior to saving.

Display delete confirmation

Allows you to enable or disable pop-up dialogs that prompt you for confirmation prior to deleting.

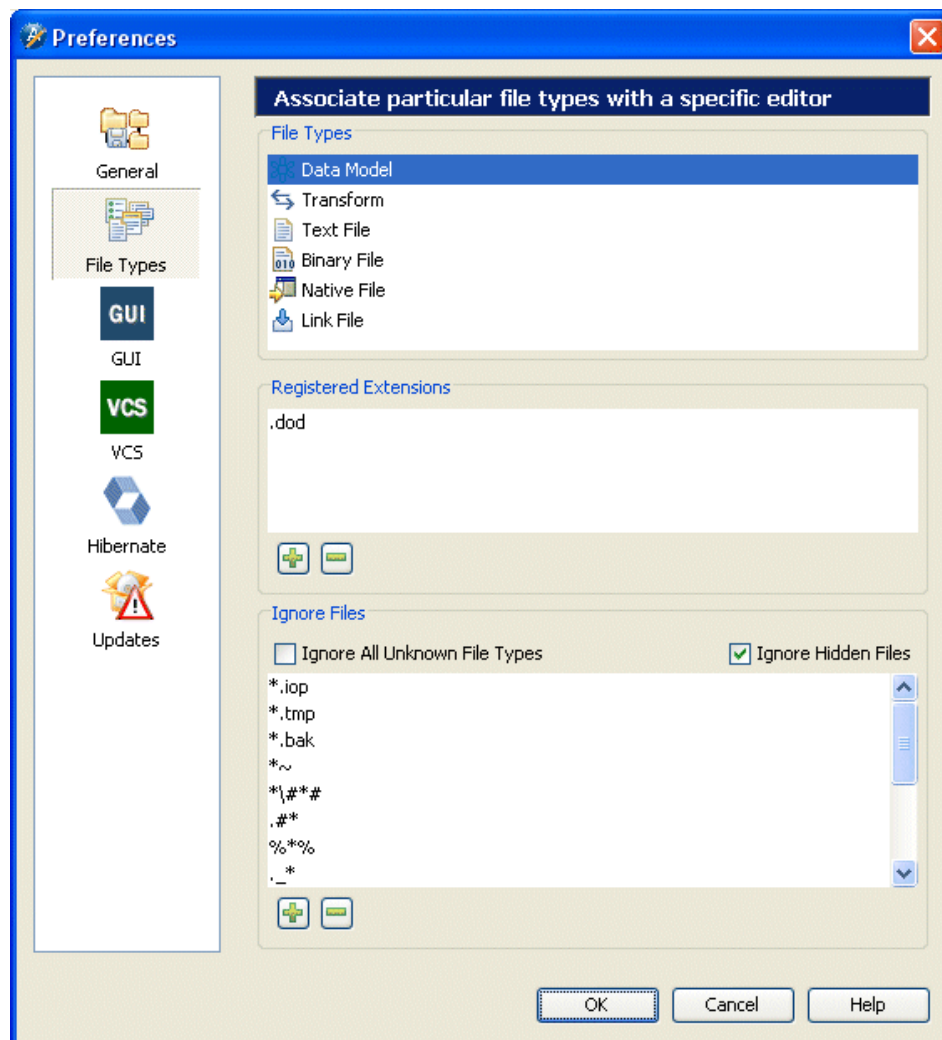
Reset Tips





Resets the display status of all pop-up tips so they all appear again.

Clear All

Clears all preference data, thereby restoring ADS to its factory settings.

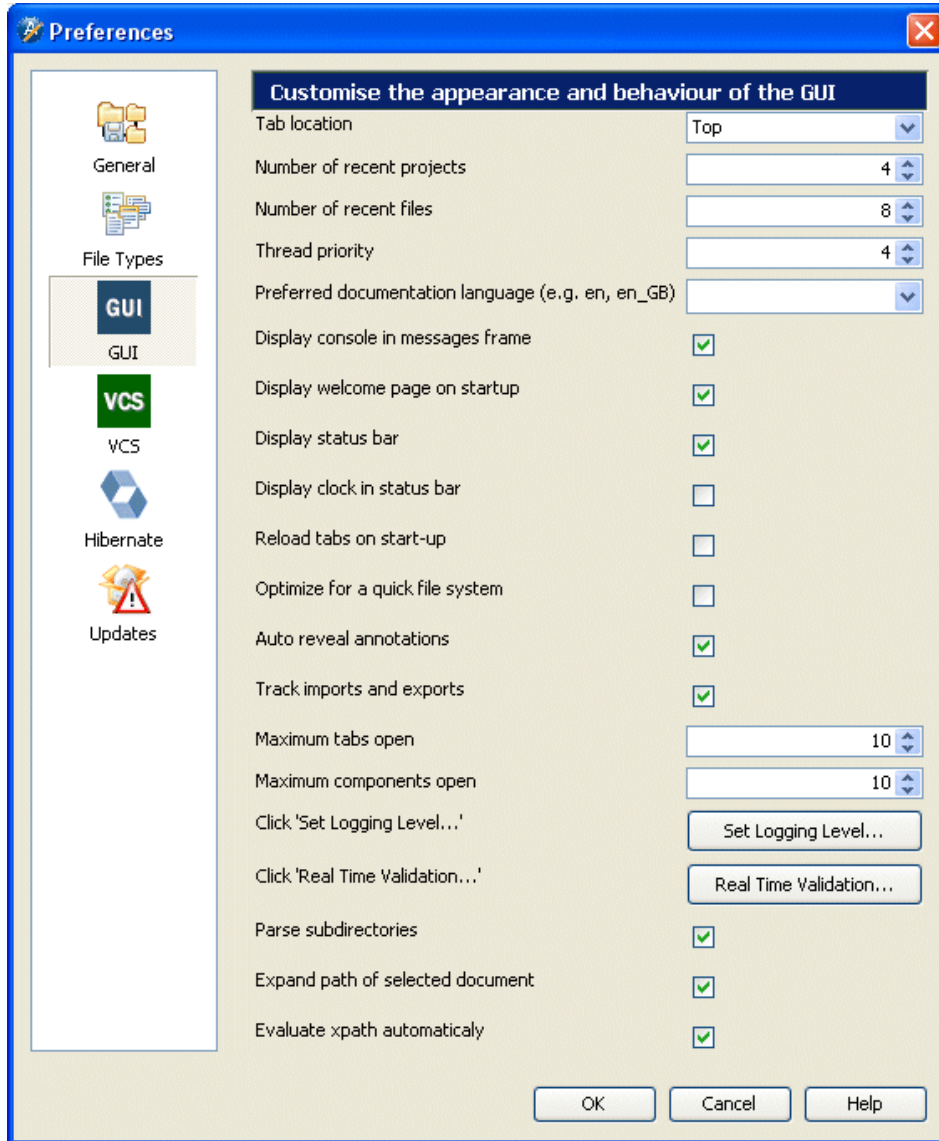
Preferences - File Types



The File Types panel controls which files are displayed in the Project window. Selecting the desired file type in the File Types list allows you to add or remove known file extensions on the Registered Extensions panel via the  and  icons. You can also configure your ignored file patterns via the lower set of  and  icons. By default,

this list contains many commonly found system and VCS files such as "CVS", "SCCS" and all files ending in ~. Finally, the Ignore Hidden Files check box can be used to show or hide files marked on the file system with the hidden flag. The Ignore Unknown File Types check box will hide files whose type is not known in ADS Designer.

Preferences - GUI



The configurable preferences on the GUI panel are:

Tab location

Allows you to specify on which side of a tab (top, bottom, right or left) the tab name should reside.

Number of recent projects

Allows you to set the number of most recently accessed projects that should be displayed when you select

File > Reopen.

Number of recent files

Allows you to set the number of most recently accessed data models and transformations that should be displayed when you select **File > Reopen**.

Thread priority

Allows you to set the relative priority for threads performing build, save, load functions, and so on. Setting this value too high can result in an unresponsive Designer. Setting this value too low means these tasks might take longer to execute.

Preferred documentation language

Allows you to set the preferred language in which to display documentation tool tips. If the preferred language is not available, the first documentation node will be used. The value of this property should contain a language code as defined in RFC 1766, Tags for the Identification of Languages and for wider applicability ISO 639 possibly followed by a country code. For non ISO 639 languages you can use one of the i-codes registered with IANA or prefix with "x-" or "X-" and invent your own.

Display console in messages frame

Allows you to choose whether to display the console output in the Messages window. (This can be useful for debugging errors.)

Display welcome page on startup

Allows you to choose whether the Home page is displayed each time you start Designer.

Display status bar

Allows you to choose whether the status bar will be displayed in the legend at the bottom of the Designer workbench.

Display clock in status bar

Allows you to choose whether the system clock will be displayed in the status bar.

Reload tabs on start-up

Allows you to choose whether to reload the last set of open tabs when the Designer is restarted.

Optimize for a quick file system

Allows you to choose whether calls to the file system should be minimized. This might be desirable if the file system is particularly slow or is a networked drive. Such is the case for some centrally managed version controlled drives such as ClearCase.

Auto reveal annotations

Allows you to choose whether annotations should be displayed.

Track imports and exports

Allows you to choose whether imports and exports are to be tracked.

Maximum tabs open

Allows you to set the maximum number of tabs that may be open at any one time.

Maximum components open

Allows you to set the maximum number of component tabs within a single data model tab that may be open at one time.

Set Logging Level

Allows you to set the logging level for parsing, validation and running transformations.

Real Time Validation

Allows you to choose whether validation of elements, attributes, enumerations and rules is to be performed in real time.

Parse sub directories

Allows you to choose whether files in subdirectories will be also parsed by default, when you load data from a file in a parent directory into a model.

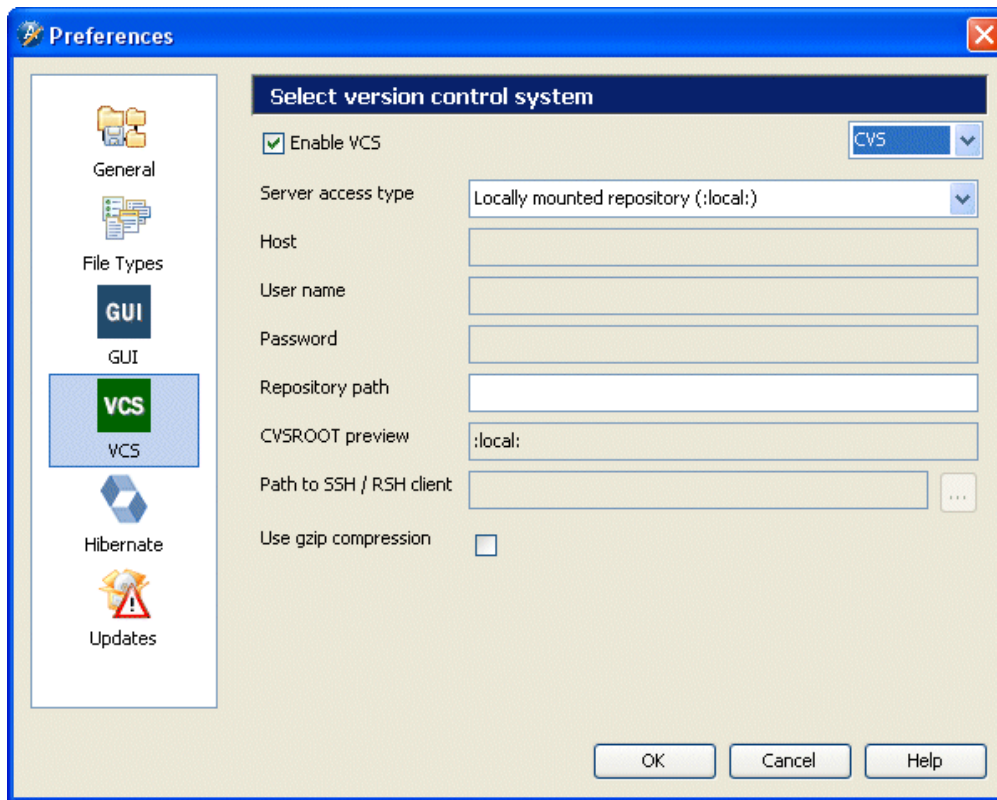
Expand path of selected document

Allows you to choose whether full document paths are to be displayed.

Evaluate XPath Automatically

Allows you to choose whether XPath expressions are to be evaluated automatically.

Preferences - VCS



The configurable preferences on the VCS panel will vary depending on the type of version control system you choose after selecting the Enable VCS check box.

If you select CVS as your version control system, the configurable preferences on the VCS panel are:

Server access type

Allows you to choose which type of server is involved in your CVS system.

Host

Allows you to set a valid hostname for the server, if it is not local.

User name

Allows you to set a valid user name for contacting the server, if it is not local.

Password

Allows you to set a valid password for contacting the server, if it is not local.

Repository Path

Allows you to set the full path for your CVS repository.

CVSROOT preview

Is updated with the value you type as the repository path.

Path to SSH / RSH client

Allows you to set the client path, if the server access type is external secure shell (:ext:).

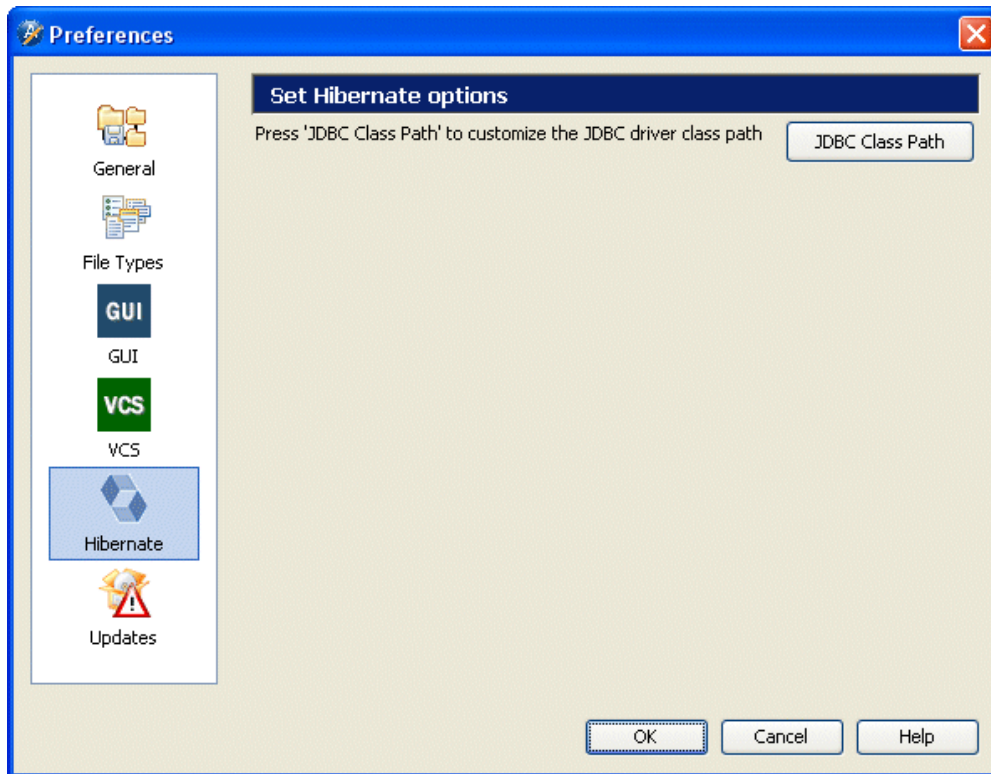
Use gzip compression

Allows you to choose whether CVS is to use gzip compression for your files.

If you select Subversion as your version control system, the VCS panel Allows you to choose whether to use the system default Subversion configuration directory. If you uncheck this check box, you can choose the alternative Subversion configuration repository that you want to use instead.

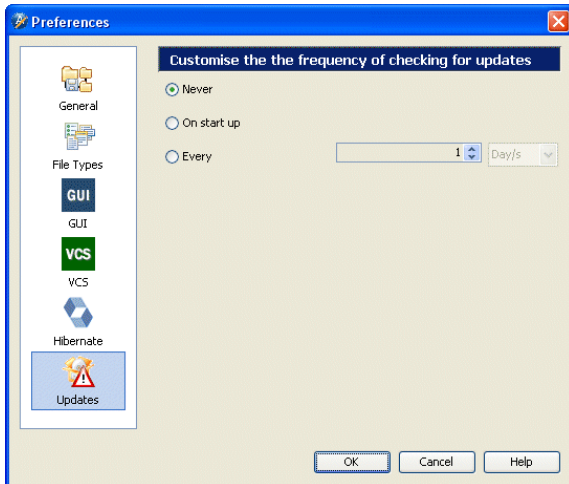
If you select ClearCase as your version control system, the VCS panel is blank.

Preferences - Hibernate



The Hibernate panel allows you to customize the JDBC driver classpath for your application.

Preferences - Updates

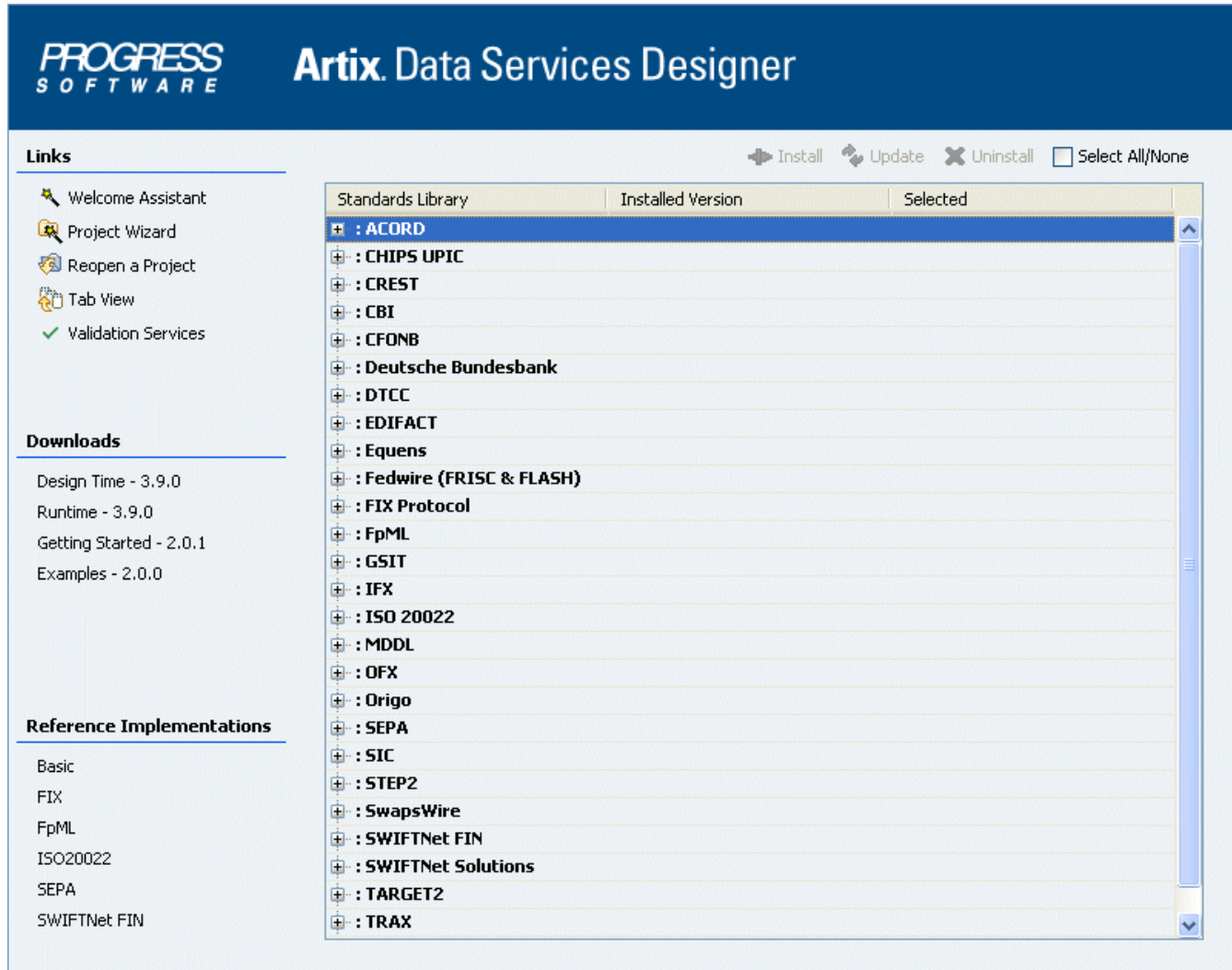


The Updates panel allows you to determine how often ADS should automatically check for updated versions of its various components and plug-ins. The available options are to never check, to check every time you start the Designer, or to check at regular intervals that can be measured in days, weeks or months.

Note: You may also check for updates manually, on an ad-hoc basis, by selecting **Help > Check for Updates** from the menu bar.

The Home Page

The ADS Designer Home page appears by default when you first launch the application.



If you close the home page, you can re-open it by selecting **Help > Home Page**.

The Home page contains the following sections:

Links

From here you can launch useful tools such as the Welcome assistant and the New Project wizard.

Downloads

You can download the latest version of the following ADS components from here:

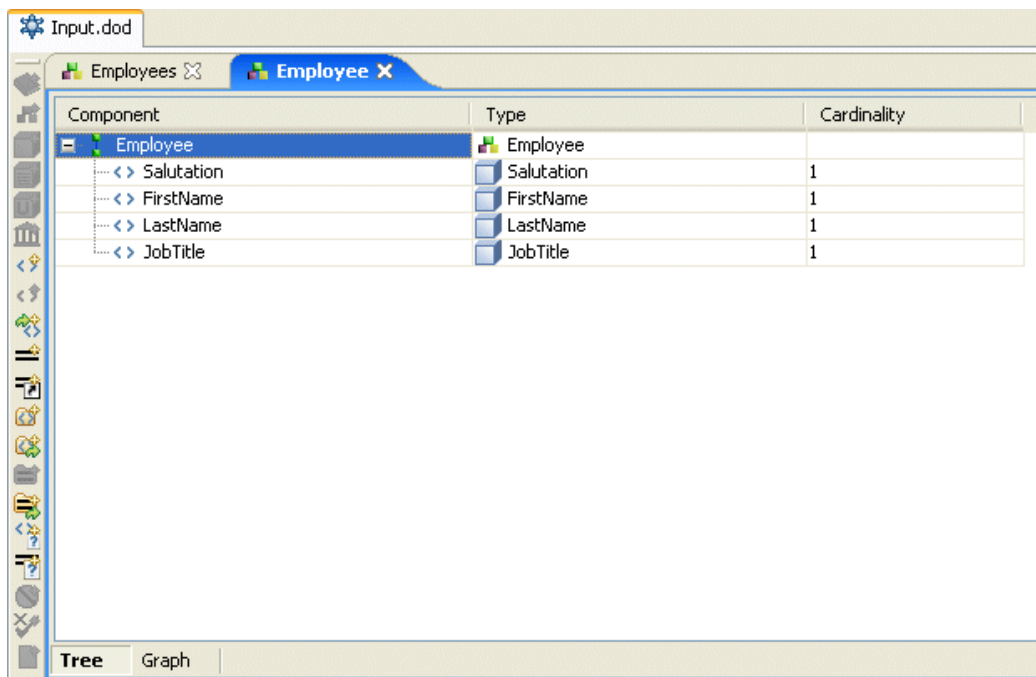
- Design Time - allows you to update ADS Designer to the latest version
- Run Time - allows you to update the ADS runtime to the latest version
- Getting Started - allows you to download the [Getting Started samples](#)
- Examples - allows you to download the [examples](#)
- Reference Implementations - allows you to download the [reference implementations](#)

The Main Window

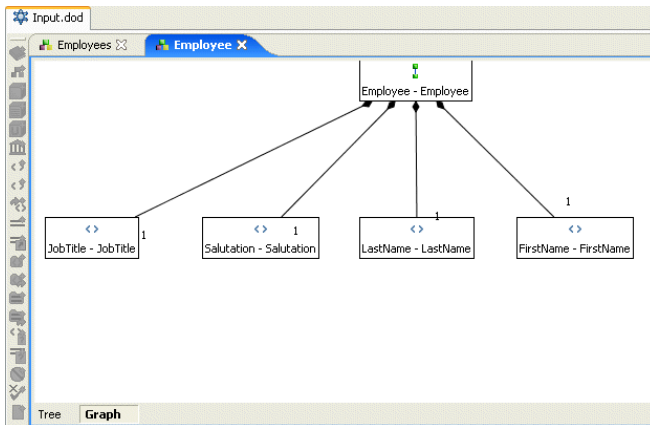
The main window serves a variety of different uses depending on which types of component are currently open. A separate tab is created in the main window for each component that is currently loaded in memory.

Data Model Tabs

A data model tab displays details of a data model and its components that you have opened to work on. Data model components are arranged within two levels of the tabbed pane. The top level of tabs displays each of the data models (.dod files) that are currently open. The second level of tabs displays all the components of the particular data model you have currently selected in the top level. To switch between data models, click the tabs in the top level. The components of that data model are then displayed in the second level of tabs. To switch between the components of a particular data model, click the tabs in the second level. The components, corresponding types and cardinality are easily distinguished in the grid format in the main window.

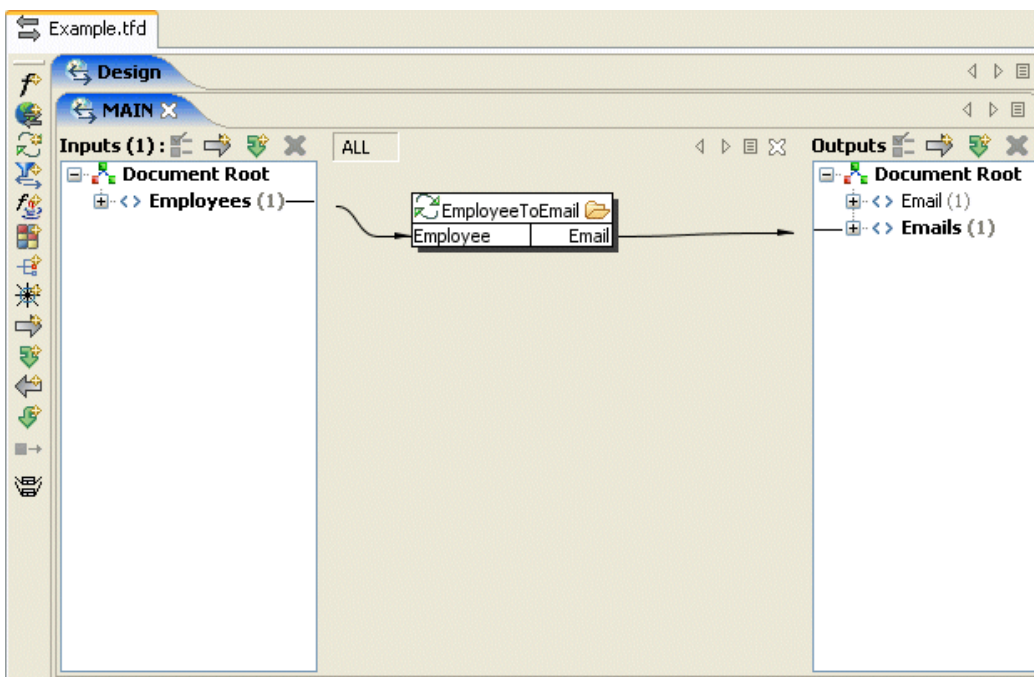


Within a particular tab, you can work with complex data types (as shown above), enumerations, validation rules, element groups and attribute groups. When working with complex data types, you can switch between tree and graph views of the complex type's structure:



Transformation Tabs

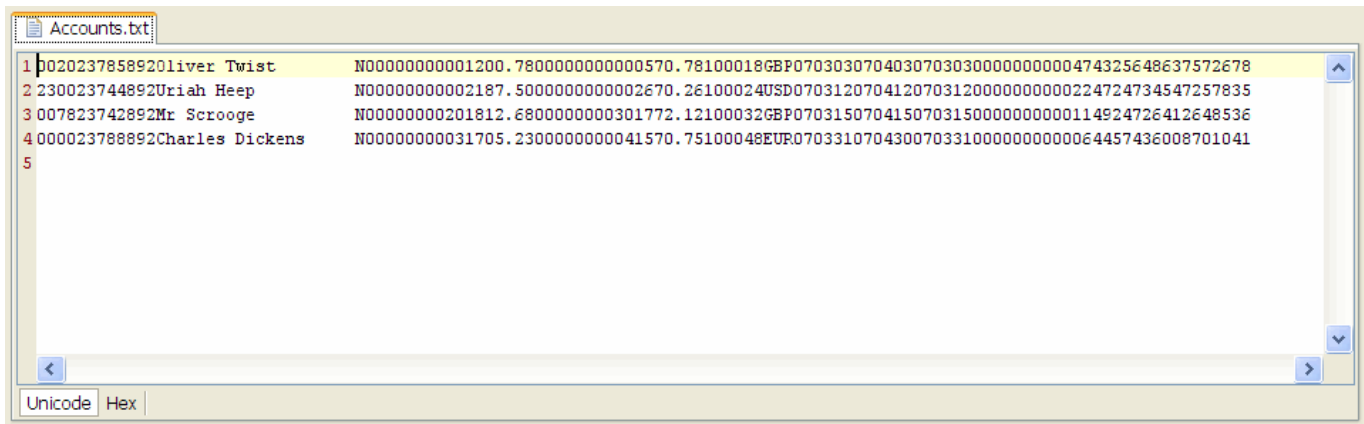
A transformation tab allows you to add inputs and outputs and edit the rules which comprise the transformation. Transformation components are arranged within three levels of the tabbed pane. The top level of tabs displays each of the transformations (.tfd files) that are currently open. The second level of tabs displays whether the transformation is in Design or Run mode. Provided the Design tab is open in the second level, the bottom level of tab displays all the currently open components (for example, local transformations, filters and so on) of the particular transformation you have currently selected in the top level. To switch between transformations, click the tabs in the top level. The components of that transformation can then be displayed in the lower level of tabs. To switch between the components of a particular transformation (when the Design tab is open in the second level), click the tabs in the bottom level. The MAIN tab always represents the global transformation.



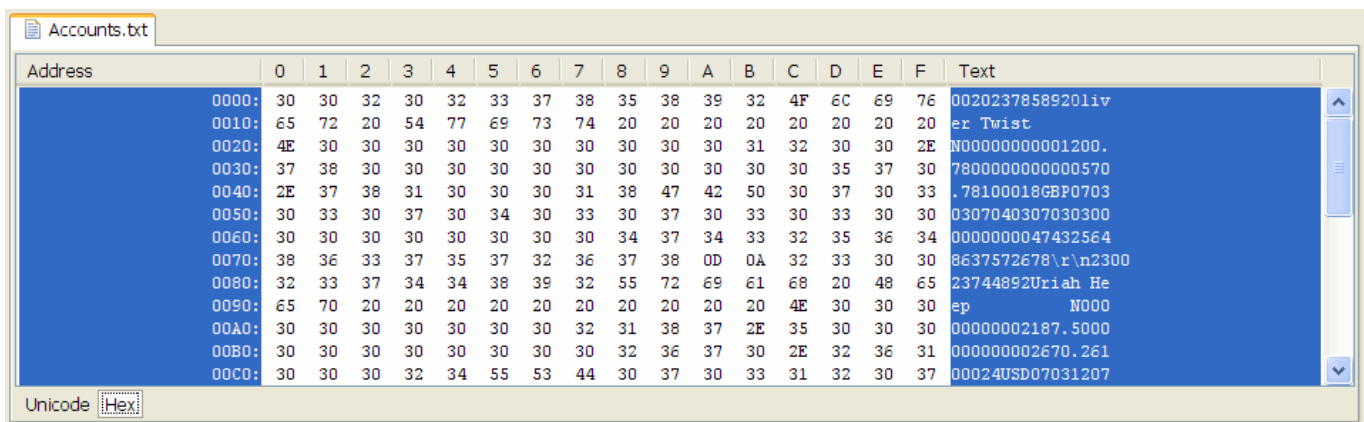
Tabs for Data Files

Data files such as text (.txt) files, XML (.xml) files, and XML (.xsd) schemas can be opened in their own tabs. In each case, the tab allows you to view the data in Unicode or Hexadecimal format.

For example, the following shows the sample Accounts.txt in Unicode format:

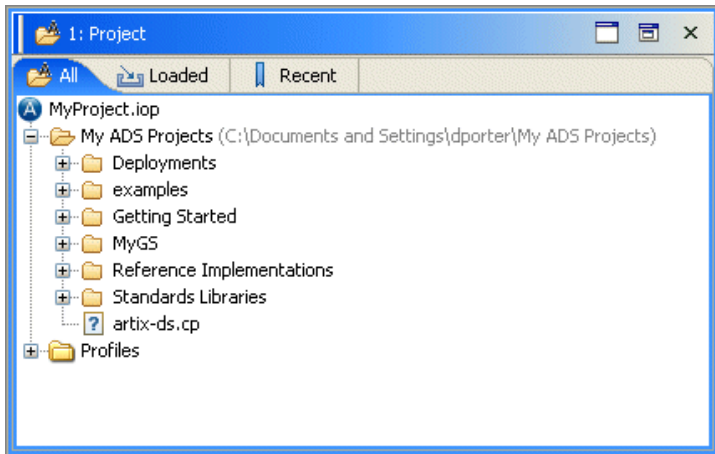


For example, the following shows the sample Accounts.txt file in Hexadecimal format:





The Project Window

The Project window displays a tree view of all directory location paths specified in the 'Paths' section of the project properties:



The example project shown knows about one path, My ADS Projects. This name is an alias to the true path name. The Examples, Getting Started and Standards Libraries directories are all examples of plug-ins that you can download as part of your product installation.

The built code for a project is stored under a directory called Deployments, which holds all Ant build files, Java source code, compiled Java classes and .jar files created at build time.

Data models are displayed with a  icon and transformations are displayed with a  icon.

If a particular file has been loaded into memory, its name is shown in bold in the Project window.

If a file contains [unresolved linkage errors](#), its name is shown in red.

The Project window allows you to:

- Open files by double-clicking on them, or by selecting them and pressing Enter, or by right-clicking them and selecting Open, or by dragging and dropping them from the file system.
- Build one or more files. (You can select multiple files simultaneously by clicking each file while pressing the CTRL key).
- Create new data models, transformations and directories.
- Copy and paste data models and transformations between directories.
- Rename data models, transformations and directories.
- Delete data models, transformations and directories.
- Save, load and unload files.
- Import metadata into a directory.
- Export metadata from one or more files.
- Open the [Run Wizard](#) and [Diff Tool](#) on a particular file.
- Interact with Version Control systems such as CVS or ClearCase.

These operations can be controlled either by using the toolbar or by using the context menus when you right-click.

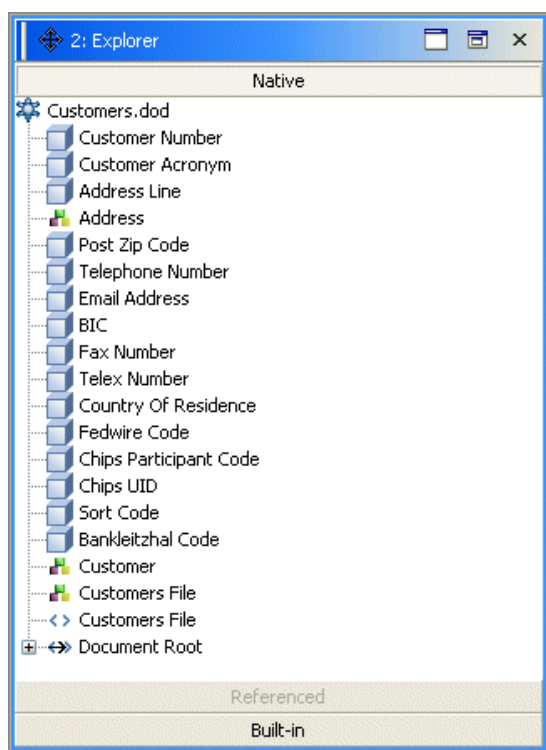
The Explorer Window

The Explorer window contains details of the data model or transformation tab currently displayed.

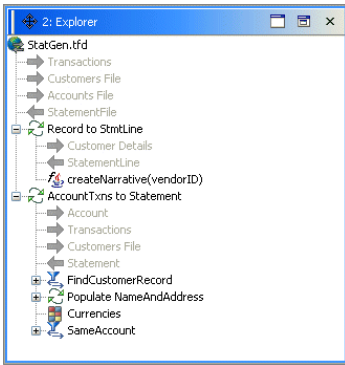
For a data model, it is split into three sections:

- **Native**, holding a classification hierarchy of data components belonging solely to this model.
- **Referenced**, holding a classification hierarchy for each data model imported, included or redefined by this data model.
- **Built-in**, holding a list of all built-in primitive data types supported by ADS.

You can access each section by clicking on its titled tab.



For a transformation, it shows you the inputs, outputs and reusable components and functions:



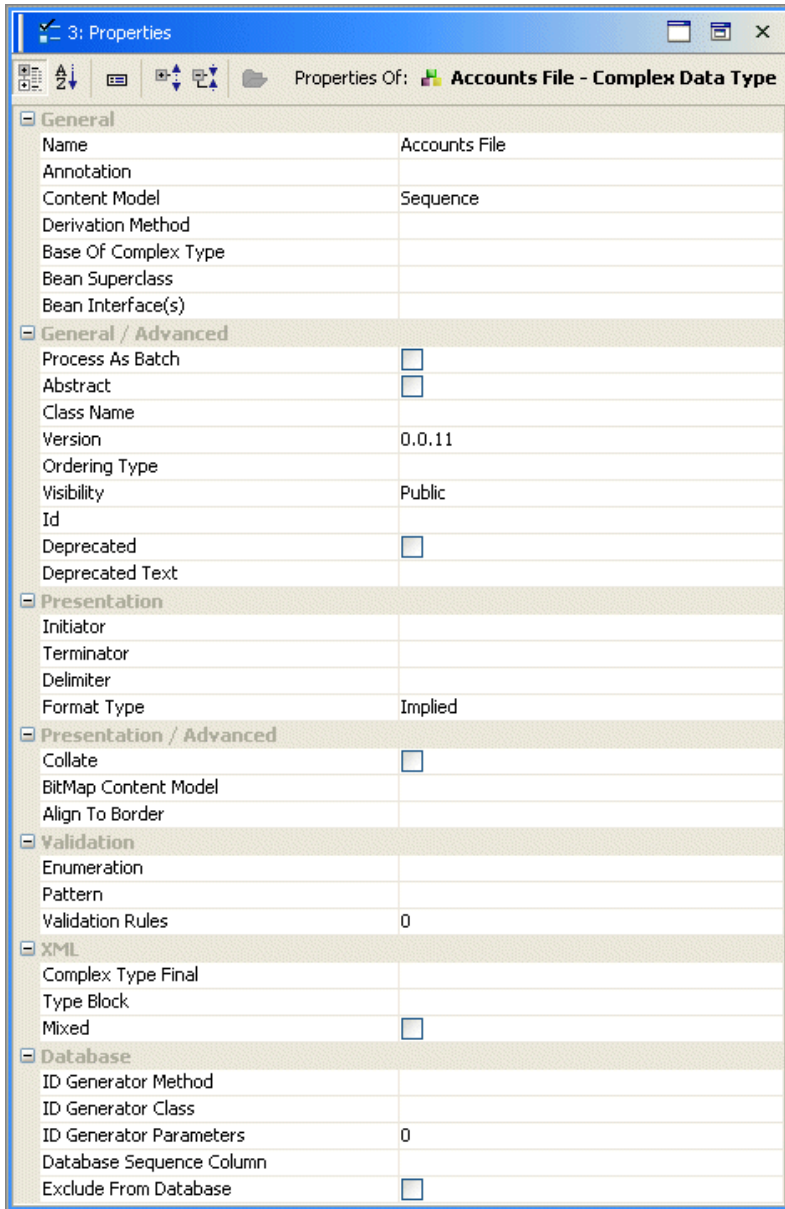
The Explorer window allows you to:

- Open certain components and referenced files by double-clicking on them, by selecting them and pressing **Enter**, or by right-clicking them and selecting **Open**
- Build one or more components. (You can select multiple files simultaneously by Ctrl-clicking each file)
- Verify one or more components
- Create new components
- Copy and paste components
- Delete and rename existing components
- Use the Diff Tool to display the differences between two components
- Find usages, derived types, and substitution elements of a selected component
- Open the [Run Wizard](#) on a particular component




These operations can be controlled either by using the toolbar or by using the context menus when you right-click.




The Properties Window

The Properties window is a context-sensitive tabular display of the properties associated with the currently selected ADS component. It displays different information depending on whether you have selected a data model, complex type, simple type, or transformation. The following example shows the properties for the complex data type Accounts File from the sample Accounts.dod data model:



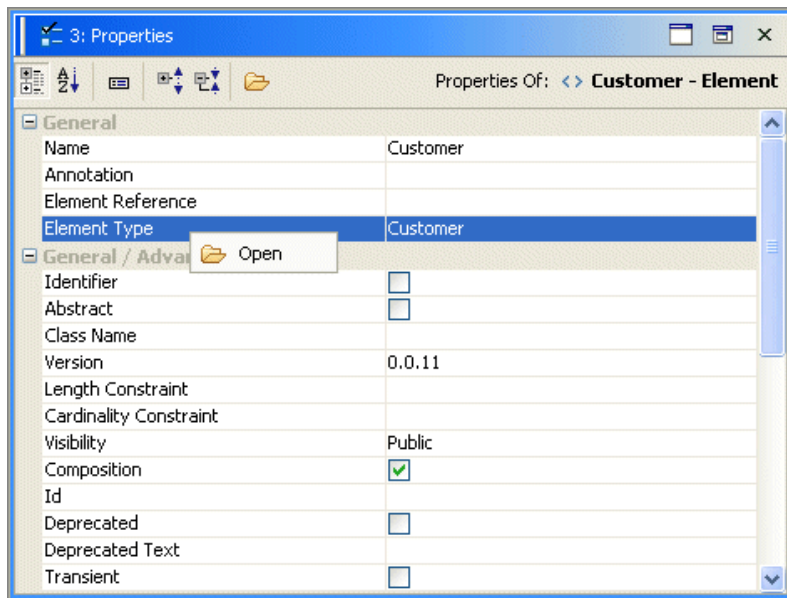
The icons at the top of the Properties window are:

-  allows you to arrange the properties in categorized sub-tables (as shown).
-  allows you to arrange the properties in alphabetical order.
-  allows you to display or hide the description area at the bottom of the window.


-  allows you to expand all sub-tables.
-  allows you to collapse all sub-tables.
-  allows you to edit properties that represent a reference to another ADS component (that is, components from pre-built data models or read-only models that are not open in their native tab).

Depending on the data component for which the properties are being displayed, some entries in the table can be edited. Clicking on the right-hand column causes editable components to change appearance or to display a dialog or some other input component, as appropriate.

When you right-click a property that represents a reference to another ADS component, a menu opens that allows you to navigate to that component, as shown below:



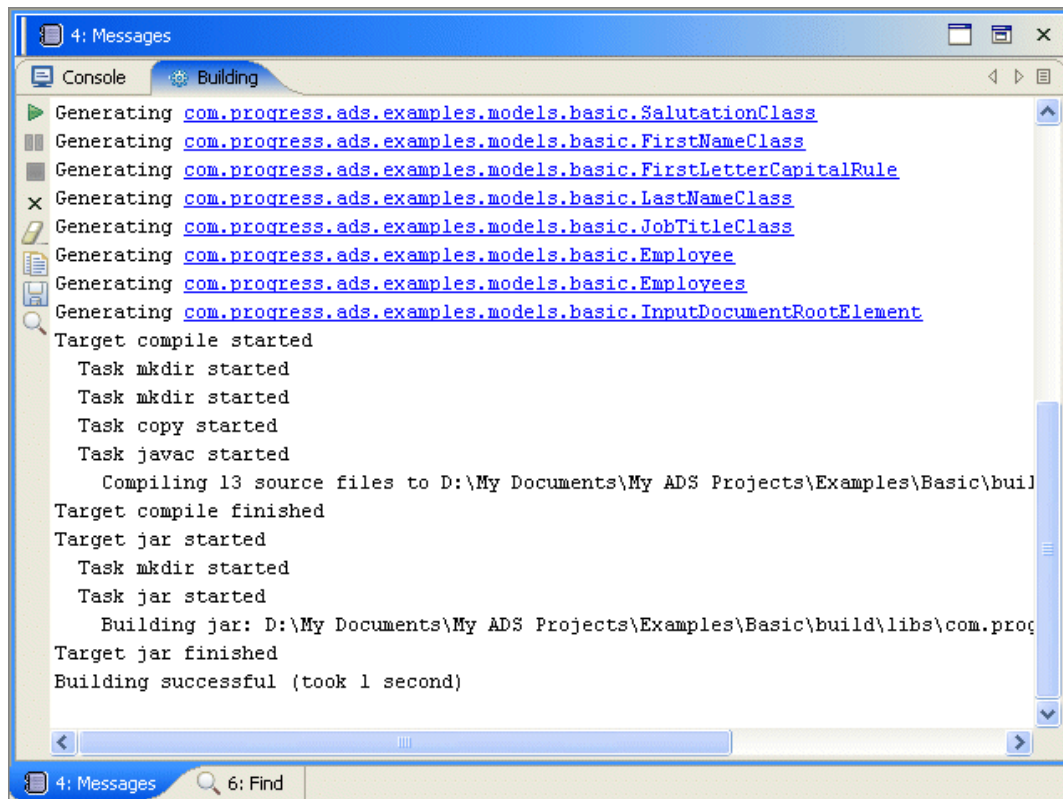
The **Open** option opens the component as if you had double-clicked it in the Explorer window.

Components from pre-built data models, read-only models, or those that are not open in their native tab are marked as read only and you cannot edit their properties directly. To edit these properties, right-click the component and select **Open**. Alternatively, click the component and then click the  icon.

The Messages Window

The Messages window is used to display messages:

- From the console. (This is enabled by default and is controlled through the **Edit > Preferences > GUI** menu option.)
- Relating to verification and builds (as shown).
- Relating to Ant build file task execution.



The icons on the left-hand side can be used to restart, pause, stop and kill the process being monitored.

Right-click in the Messages window to display the following options:

- **Clear** deletes all content in the Messages window.
- **Copy Content** copies all content in the Messages window to the system clipboard.
- **Copy Selected Content** copies the selected content in the Messages window to the system clipboard.
- **Save Content** saves the Messages window content to a text file.
- **Find** displays the Find panel where you can enter text you want to search for.

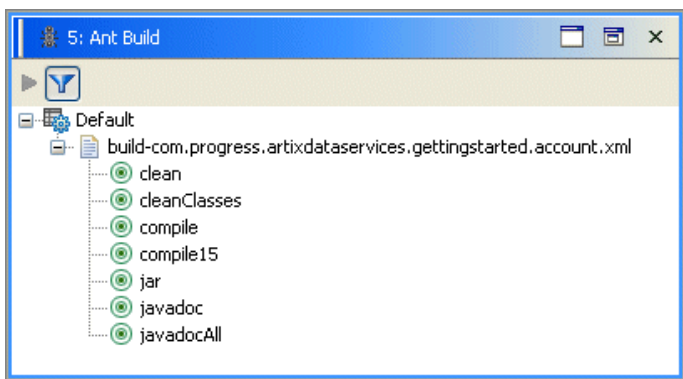
The Ant Build Window



The Ant Build window contains details of build files that have been generated as a result of models and transformations being built from ADS Designer.

A build file is generated for each namespace involved in a build.

ADS Designer can generate the file once it is aware of a namespace. Where a file has been generated but the corresponding build has not yet occurred, the filename is shown in red.

You run targets within a build file either by double-clicking, or by right-clicking and selecting **Run Ant Target**. Target execution results are displayed in the Messages window. Right-clicking and selecting **Set Ant Target** sets the selected target to be run when one or more components from the target namespace are built.



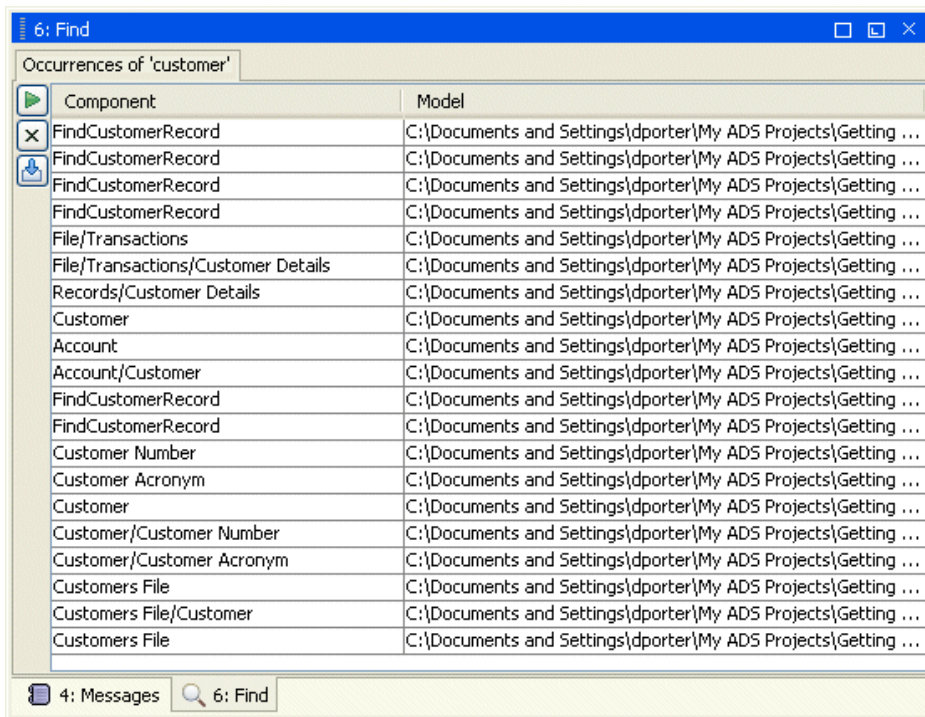
Click the  icon on the Ant Build window to run the selected ant target. Click the  icon to toggle the view of Ant files that have not been deployed. This enables you to hide all ant project files that have not been deployed (that is, those highlighted in red).

The Find Window




The Find window displays results of location, usage, derivation and substitution group searches. Some of these searches deliver hierarchical results and others deliver tabular results. In each case, you can navigate to components by double-clicking on them.

Find Components

The following example shows all components with the word 'Customer' in their names:

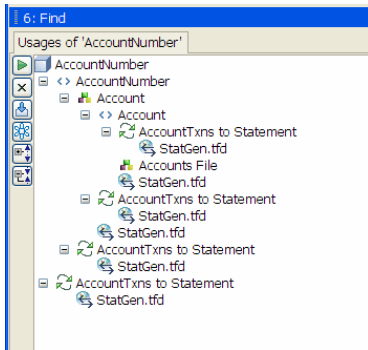


The icons on the left can be explained as follows:


- Click  to repeat a search.
- Click  to cancel a search.
- Click  to export the results of a search to XML.

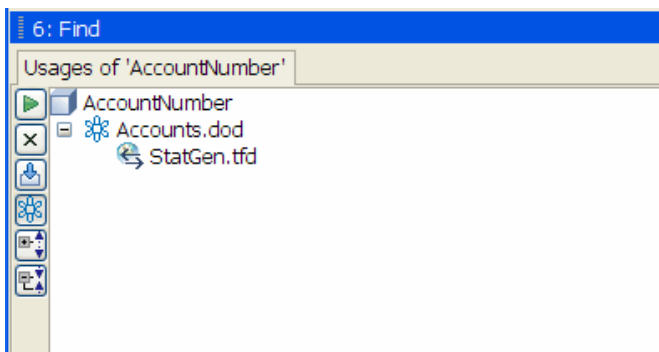
Find Usages

The following example shows the usages of the simple type "AccountNumber":






This shows that 'AccountNumber' is used by the 'Account' complex type which in turn is used by the 'Accounts File' complex type. It also shows that it is used by AccountTxns to Statement local transformation within the StatGen.tfd global transformation.

If you click the  icon, the search is limited to data models and global transformations only. For example, the following shows that 'AccountNumber' is used by the Accounts.dod data model and the StatGen.tfd global transformation.



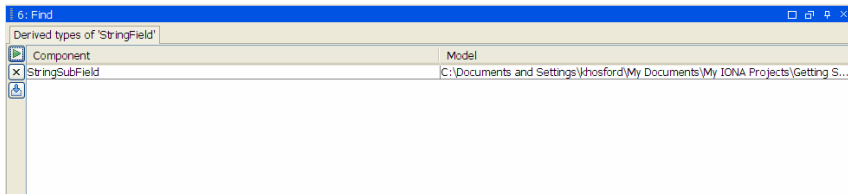
The additional icons on the left can be explained as follows:

- Click  to limit the search to data models and global transformations.
- Click  to expand all nodes.
- Click  to collapse all nodes.

Note: The tree is populated on demand and, with large data structures and many models loaded into ADS memory, this operation can be quite time consuming.

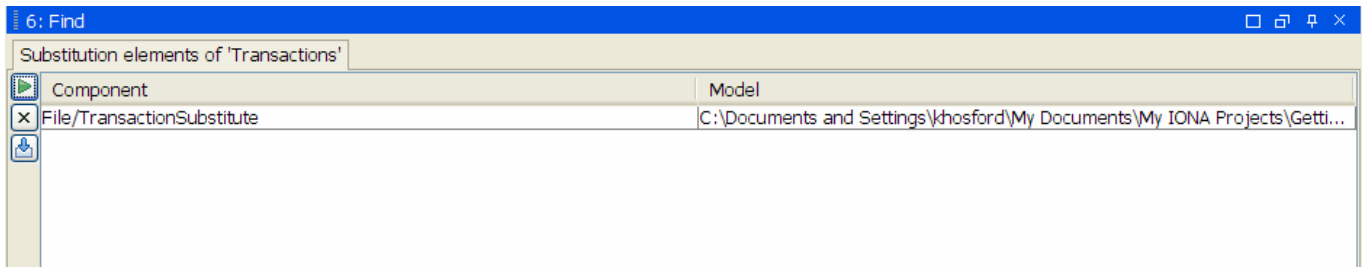
Find Derived Types

The following example shows a 'StringSubField' type that is derived from a 'StringField' type:



Find Substitution Elements

The following example shows a 'TransactionSubstitute' element in the [substitution group](#) of 'Transactions':

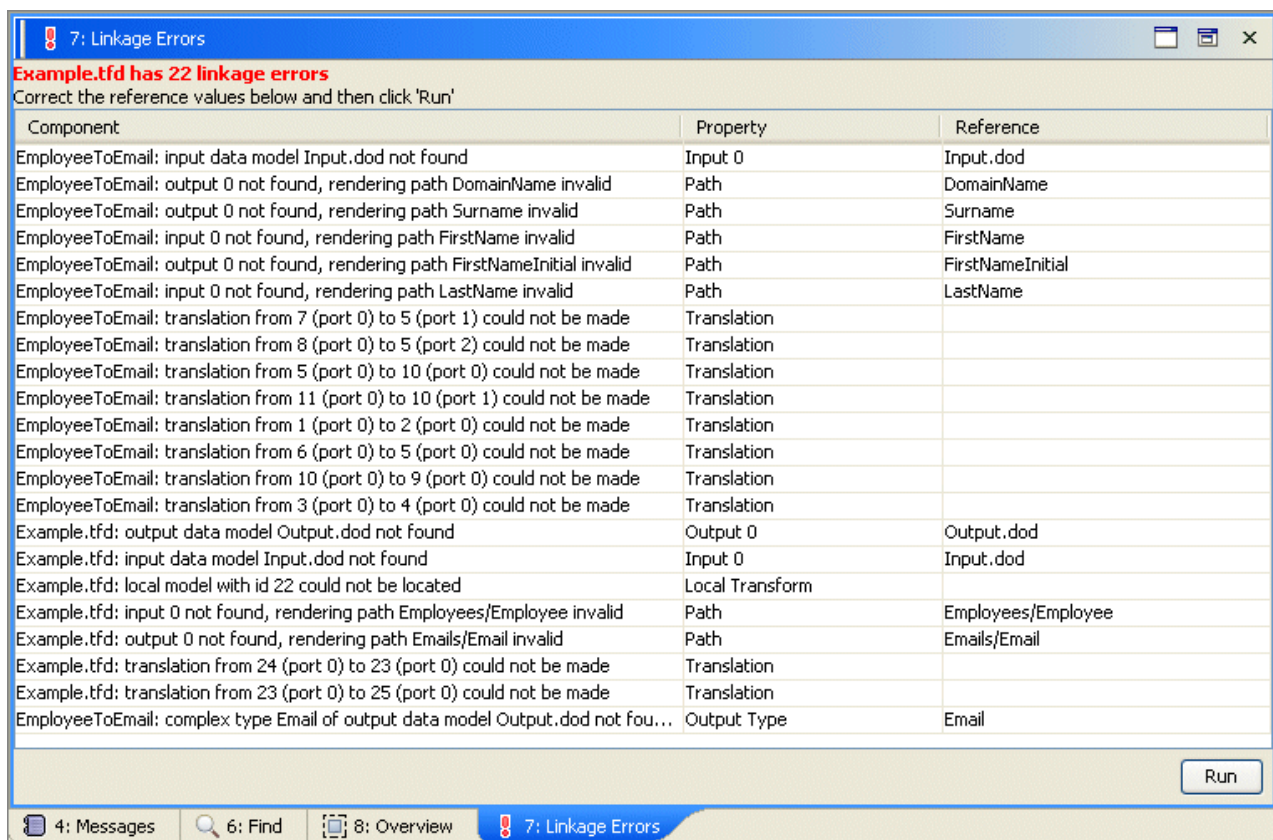


The Linkage Errors Window

The Linkage Errors window resolves any outstanding linkage errors in the selected file (whose name is highlighted in red). Linkage errors are caused when either the location, filename or contents of a file have been changed without any referencing file being loaded and resaved after the change.

For example, if model 'A' imported model 'B', and model 'B' is subsequently renamed to 'C' while model 'A' is not loaded, when model 'A' is next loaded it will be unable to resolve its path to model 'B'.

For this type of 'file linkage error', you are prompted to relocate the missing file. If you cannot find it, one or more outstanding linkage errors will result. Linkage errors can also be caused in the above example when a component in 'C' is deleted while it is being used by 'A'.

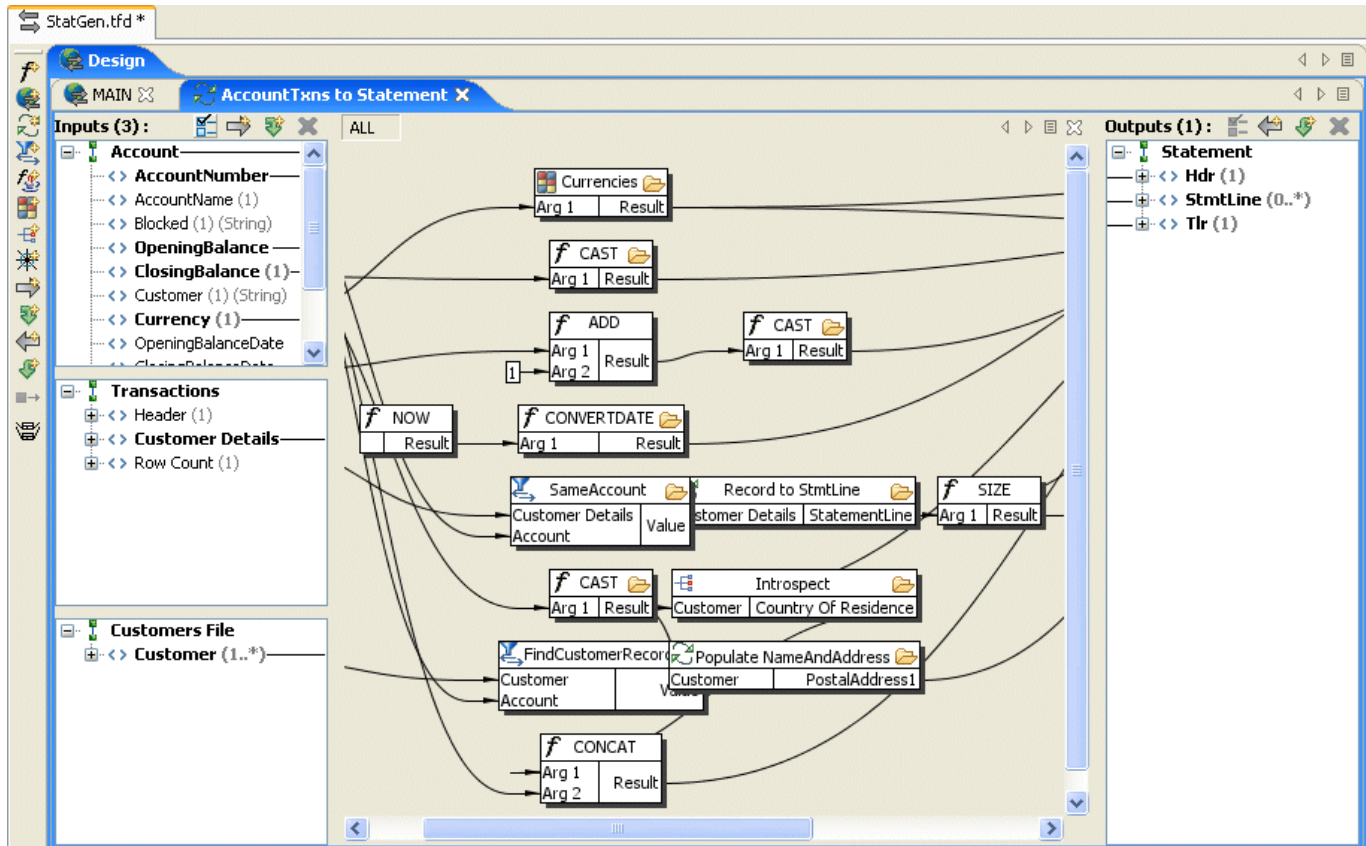


You should be aware of the following points in relation to linkage errors:

- It is often the case that resolving one linkage error will resolve several others. The error in the above example can be resolved by editing the reference value and clicking 'Run'. When this is done, in this case, all other errors are resolved.
- The reference of some errors cannot be edited, because they are purely dependent on the resolution of some other error in the list.
- It is possible to continue working on models with unresolved linkage errors but they will be lost upon saving.

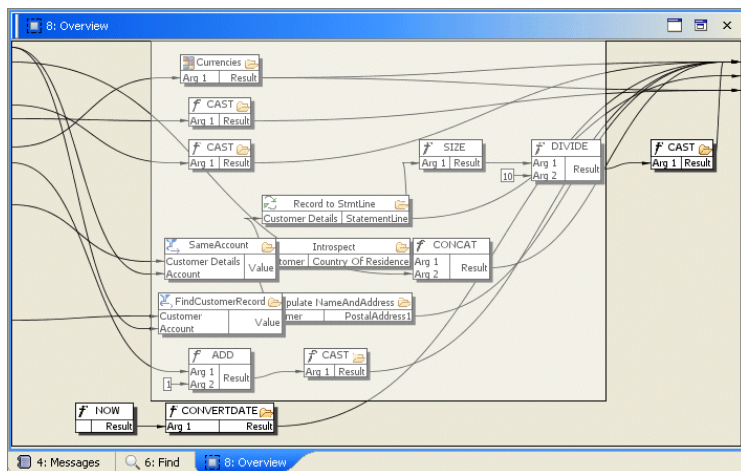
The Overview Window

Sometimes, transformations can become quite large and complex (as shown in the following example) and therefore difficult to work with. The Overview window lets you drag an outline of the visible area around on a larger canvas.



When you have zoomed in on a small area for a detailed operation, you might find it useful to see the entire image while you work. You can do this by opening the Overview window, which displays a small view of the image. When the image is too large to fit in its window, its visible area is enclosed by a rectangle. To move from one area of the image to another, move the cursor over the rectangle. When the cursor changes to the Hand icon, click the left mouse button, drag to the new area you want to view, and release the button. Note that the rectangle moves as you drag the mouse button.

Note: The Overview window displays transparent areas of an image as white.



Toolbar

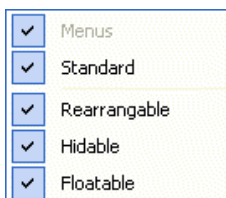
The ADS Designer toolbar contains the following options.



From left to right the icons are:

- [Home Page](#)
- [New](#)
- [Open File](#)
- [Save Tab](#)
- [Save All](#)
- [Synchronize Project](#)
- [Undo](#)
- [Redo](#)
- [Cut](#)
- [Copy](#)
- [Paste](#)
- [Rename](#)
- [Delete](#)
- [Find Components](#)
- [Back](#)
- [Forward](#)
- [Run](#)
- [Reload Active Run Configuration](#)
- [Build Components](#)
- [Verify Components](#)
- [Preferences](#)
- [Project Properties](#)
- [Profile Settings](#)
- [User Guide](#)
- [Quick Help](#)





















There are various options available to you on how you want the toolbar to look and feel. If you right click anywhere in the menu area, you can select which bars you want to be displayed.



Data Model Palette

When you open a data model, the Data Model palette is displayed along the left of the tab.

The icons in the Data Model palette are:

- [Classification Group](#) 
- [Complex Data Type](#) 
- [Atomic Simple Data Type](#) 
- [List Simple Type](#) 
- [Union Simple Type](#) 
- [SWIFT Field Type](#) 
- [Element](#) 
- [Element from Type](#) 
- [Element Reference](#) 
- [Attribute](#) 
- [Attribute Reference](#) 
- [Element Group](#) 
- [Element Group Reference](#) 
- [Attribute Group](#) 
- [Attribute Group Reference](#) 
- [Any Element](#) 
- [Any Attribute](#) 
- [Enumeration](#) 
- [Validation Rule](#) 
- [Annotation](#) 

Transformation Palette

When you open a transformation, the Transformation palette is displayed along the left of the tab.

The icons in the Transformation palette are:

- [Function](#) 
- [Transform Reference](#) 
- [Local Transform](#) 
- [Filter](#) 
- [Java Method](#) 
- [Hashtable](#) 
- [Introspector](#) 
- [Instantiator](#) 
- Global Input 
- Local Input 
- Global Output 
- Local Output 
- Perform Transformation 
- Align Transform 

File Menu

The content of the **File** menu changes depending on the currently selected tab. The **File** menu contains the following items:

- [Welcome Assistant](#)
- [New](#)
- [Open Project](#)
- [Reopen a Project](#)
- [Close Project](#)
- [Synchronize Project](#)
- [Reuse Component](#) (for transformations only)
- [Open File](#)
- [Reopen File](#)
- [Open Selected](#)
- [Load File\(s\)](#)
- [Save Tab](#)
- [Save All](#)
- [Save](#)
- [Close Active Tab](#)
- [Close Active Component](#) (for data models and transformations only)
- [Unload File\(s\)](#)
- [Close](#)
- [Packaging](#)
- [Exit](#)

Welcome Assistant

The Welcome Assistant is a self-explanatory introduction to ADS and a wizard to guide novice users through the most common tasks, such as importing and exporting metadata and creating transformations.

New

This opens a sub-menu with the following options:

- **Project** allows you to [create a new project](#)
- **Data Model** allows you to [create a data model](#).
- **Transform** allows you to [create a transformation](#).
- **Directory** creates a new directory under the currently selected folder in the Project window.
- **Text File** creates a new text file under the currently selected folder in the Project window.
- **Link File** creates a new link file (with an extension of .iol) under the currently selected folder in the Project window.
- **Binary File** creates a new binary file (with an extension of .bin) under the currently selected folder in the Project window.

Open Project

This opens a project (.iop file) from the file system.

Reopen a Project

This opens a sub-menu for quick access to recently opened projects. The number of items appearing in the list can be configured by the [number of recent projects](#) preference.

Close Project

This closes the project that is currently open.

Synchronize Project

This synchronizes the Project window with the file system, to pick up any new or removed files.

Reuse Component

This option is only available when a transformation tab is selected. It opens the Reuse Component dialog which allows you to select a transformation component that you want to reuse.

Open File

This opens an arbitrary file from the file system. If the file's extension is not recognized, you will be prompted to make a new [file association](#). If the selected file is not accessible from the current project, it will be displayed in the Project window after the project paths.

Note: The same behavior can be achieved by dragging a file from the file system and dropping it into the Project window.

Reopen File

This opens a sub-menu for quick access to recently opened files. The number of items appearing in the list can be configured by the [number of recent files](#) preference.

Open Selected

This opens the selected object. This option is only enabled when you click a particular file in the Project window.

Load File(s)

This loads the selected file(s). This option is only enabled when you click a particular file or folder (path) in the Project window. When you load a file it is displayed in bold in the Project window, and clicking on it again loads the properties for that file in the Properties window.

Note: If you click a particular folder, all files within that folder will be simultaneously loaded.

Save Tab

This saves the currently selected tab.

Save All

This saves all currently open files and tabs.

Save

This opens a sub-menu with the following options:

- **Save File(s)** saves the selected file(s) in the selected directory.
- **Save File As** saves the selected file with a new name.
- **Save Tab As** saves the currently selected tab with a new name.
- **Save All Tabs** saves all currently open tabs.

Close Active Tab

This closes the currently selected tab. (This does not unload the tab, hence you are not prompted to save a changed file before closing its tab.)

Close Active Component

This option is only available when a data model or transformation tab is selected. It closes the tab for the currently selected component.

Unload File(s)

This unloads the selected file(s). (You are prompted to save any changed file before unloading it.)

Close

This opens a sub-menu with the following options:

- **Close All Tabs** closes all currently open tabs. (This does not unload the tabs, hence you are not prompted to save changed files before closing the tabs.)
- **Close All But Active Tab** closes all currently open tabs apart from the one that is currently active. (This does not unload the tabs, hence you are not prompted to save changed files before closing the tabs.)
- **Close All Components** closes all currently open components.
- **Close All But Active Component** closes all currently open components apart from the one that is currently active.
- **Unload Tab** unloads the active tab. You are prompted to save the underlying file if it has changed.

Packaging

This opens a sub-menu with the following options:

- [Create Package](#)
- [Unpack](#)

Exit

This closes ADS Designer.

Edit Menu

The content of the **Edit** menu changes depending on the currently selected tab. It contains the following options:

- [Undo](#)
- [Redo](#)
- [Cardinality](#) (for data models only)
- [Content Model](#) (for data models only)
- [Global Input](#) (for transformations only)
- [Local Input](#) (for transformations only)
- [Replace Input](#) (for transformations only)
- [Global Output](#) (for transformations only)
- [Local Output](#) (for transformations only)
- [Replace Output](#) (for transformations only)
- [Set Constant Value](#) (for transformations only)
- [Remove Mappings](#) (for transformations only)
- [Move Mappings](#) (for transformations only)
- [Assign Elements](#) (for transformations only)
- [Cut](#)
- [Copy](#)
- [Copy Path](#)
- [Copy XPath](#) (for data models only)
- [Paste](#)
- [Rename](#)
- [Delete](#)
- [Preferences](#)
- [Project Properties](#)
- [Profile Settings](#)

Undo

This reverses the last operation performed in the Designer. Certain actions, such as deleting files, cannot be undone.

Redo

If you select **Undo**, this option is then enabled. It reverses the effects of **Undo**.

Cardinality

This option is only available when a data model tab is selected. It controls the mandatory, repetitive or optional nature of the constituent parts of a data structure.

Content Model

This option is only available when a data model tab is selected. It ensures that the contents within the model behave consistently.

Global Input

This option is only available when a transformation tab is selected. It adds a new global input model that can be used across more than one local transformation.

Local Input

This option is only available when a transformation tab is selected. It adds a new local input model to a specific local transformation.

Replace Input

This option is only available when a transformation tab is selected. It overwrites an existing input model with another model. This option is only enabled when you click the input model that you want to replace within the transformation.

Global Output

This option is only available when a transformation tab is selected. It adds a new global output model that can be used across more than one local transformation.

Local Output

This option is only available when a transformation tab is selected. It adds a new local output model to a specific local transformation.

Replace Output

This option is only available when a transformation tab is selected. It overwrites an existing output model with another model. This option is only enabled when you click the output model that you want to replace within the transformation.

Set Constant Value

This option is only available when a transformation tab is selected. It sets a constant value for a function parameter within your transformation. This option is only enabled when you click the function parameter for which you want to set the constant value.

Remove Mappings

This option is only available when a transformation tab is selected. It removes mappings associated with the selected node. This option is only enabled when you click a particular component within the transformation.

Move Mappings

This option is only available when a transformation tab is selected. It overwrites an existing local transformation with a new local transformation.

Assign Elements

This option is only available when a transformation tab is selected. It opens the Assign Elements dialog which allows you to assign an element of type 'Any Element' to a specific type.

Cut

This copies selected files to the system clipboard and removes them from their original location.

Copy

This copies selected files to the system clipboard and retains them in their original location.

Copy Path

This copies the directory path of the selected object.

Copy XPath

This option is only available when a data model tab is selected. It copies the xpath of the selected object to the system clipboard.

Paste

This pastes duplicates of previously copied files into the currently selected folder in the Project window.

Rename

This renames the selected file or folder.

Delete

This deletes the selected file or folder after prompting you for confirmation.

Preferences

This opens the Preferences dialog which allows you to set various [preferences](#) that will govern various aspects of the working and look-and-feel of the Designer.

Project Properties

This displays the [properties](#) of the project that is currently open.

Profile Settings

This displays the properties of a [profile](#) from the current project. If your project has a single profile, its properties will appear immediately. If your project has more than one profile, you will be prompted to choose the profile whose properties you wish to view or edit.

View Menu

The content of the **View** menu changes depending on the currently selected tab. It contains the following options:

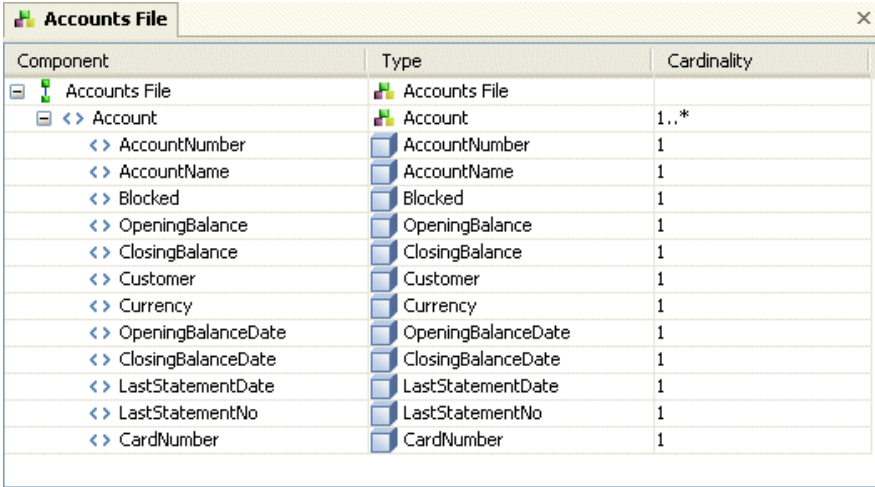
- [Use Curved Connectors](#) (for transformation tabs only)
- [View Qualified Names](#) (for data model and transformation tabs only)
- [View Substitution](#) (for data model and transformation tabs only)
- [View Derivation](#) (for data model and transformation tabs only)
- [View IDREF](#) (for data model and transformation tabs only)
- [Clear View](#)
- [Set Horizontal Spacing](#) (for data model tabs only)
- [Set Vertical Spacing](#) (for data model tabs only)
- [Zoom In](#) (for transformation tabs only)
- [Zoom Out](#) (for transformation tabs only)
- [Get Instance](#) (for transformation tabs only)
- [Add Instance](#) (for transformation tabs only)
- [Remove Get Instance](#) (for transformation tabs only)
- [Remove Add Instance](#) (for transformation tabs only)
- [Expand All](#)
- [Collapse All](#)

Use Curved Connectors

This allows you to choose whether mappings between your transformation components are displayed with curved or straight lines.

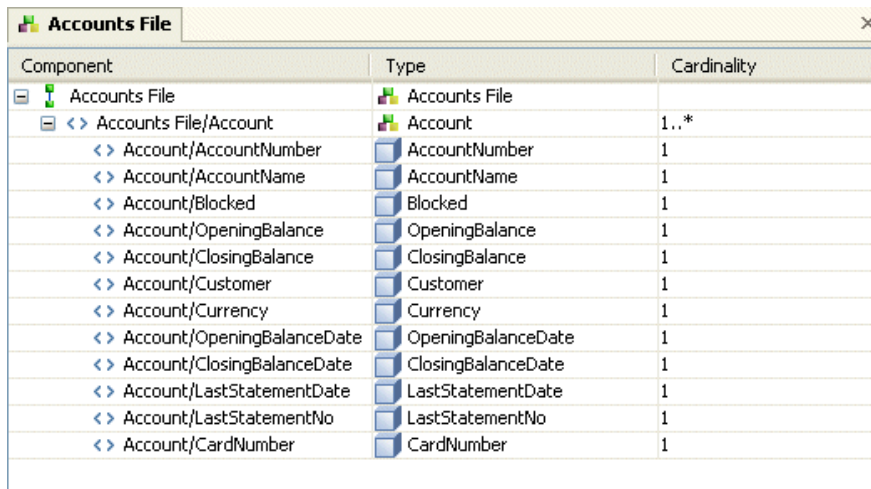
View Qualified Names

This displays the full path name of the nodes. If **View Qualified Names** is off, the tab looks like this:



Component	Type	Cardinality
Accounts File	Accounts File	
<> Account	Account	1..*
<> AccountNumber	AccountNumber	1
<> AccountName	AccountName	1
<> Blocked	Blocked	1
<> OpeningBalance	OpeningBalance	1
<> ClosingBalance	ClosingBalance	1
<> Customer	Customer	1
<> Currency	Currency	1
<> OpeningBalanceDate	OpeningBalanceDate	1
<> ClosingBalanceDate	ClosingBalanceDate	1
<> LastStatementDate	LastStatementDate	1
<> LastStatementNo	LastStatementNo	1
<> CardNumber	CardNumber	1

After you select **View Qualified Names**, the tab looks like this:



Component	Type	Cardinality
Accounts File	Accounts File	
<> Accounts File/Account	Account	1..*
<> Account/AccountNumber	AccountNumber	1
<> Account/AccountName	AccountName	1
<> Account/Blocked	Blocked	1
<> Account/OpeningBalance	OpeningBalance	1
<> Account/ClosingBalance	ClosingBalance	1
<> Account/Customer	Customer	1
<> Account/Currency	Currency	1
<> Account/OpeningBalanceDate	OpeningBalanceDate	1
<> Account/ClosingBalanceDate	ClosingBalanceDate	1
<> Account/LastStatementDate	LastStatementDate	1
<> Account/LastStatementNo	LastStatementNo	1
<> Account/CardNumber	CardNumber	1

View Substitution

This provides a list of possible elements that can be used as substitutes for a component selected in the main window. These elements have already been defined as having a substitution group.

View Derivation

This allows you to view the derivation of a particular type selected in the main window.

View IDREF

This allows you to reference an element with a unique identifier. The IDREF dialog displays a list of all ID attributes available in the document. You can choose the relevant attribute from this list.

Clear View

This clears views displayed.

Set Horizontal Spacing

This sets the horizontal spacing of the displayed elements in the Graph view. This is similar to zooming in and out across the page.

Set Vertical Spacing

This sets the vertical spacing of the displayed elements in the Graph view. This is similar to zooming in and out across the page.

Zoom In

This allows you to zoom in on the particular components of a transformation.

Zoom Out

This allows you to zoom out on the particular components of a transformation.

Get Instance

This creates a new instance of an element in an input model. This option is only enabled when you click an element in the Inputs section with a multiple cardinality of 0..* or 1..*. It creates an instance of the selected element and assigns it an identifier of [1], [2], or [3] and so on, depending on how many such instances of the element you have already created.

This is analogous to creating an INSTANCE function within your transformation.

Add Instance

This creates a new instance of an element in an output model. This option is only enabled when you click an element in the Outputs section with a multiple cardinality of 0..* or 1..*. It creates an instance of the selected element and assigns it an identifier of [1], [2], or [3] and so on, depending on how many such instances of the element you have already created.

A typical example of elements that use **Add Instance** is an address element with a cardinality of 0..*. Each address line (for example, house number and street name) will be added as an instance of the original address element.

Remove Get Instance

This removes a particular instance created with the **Get Instance** option.

Remove Add Instance

This removes a particular instance created with the **Add Instance** option.

Expand All

This expands all nodes.

Collapse All

This collapses all nodes.

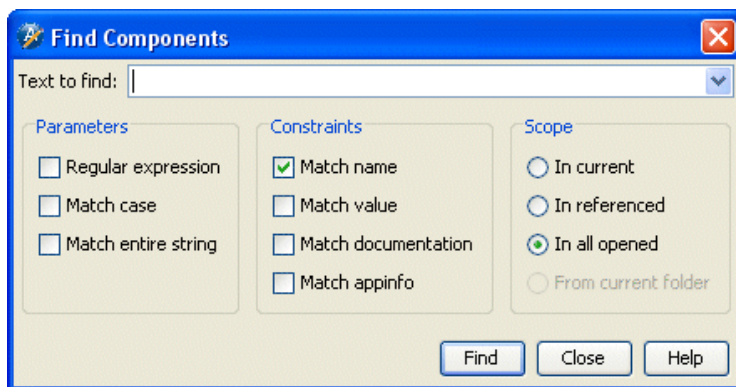
Find Menu

The content of the Find menu changes depending on the currently selected tab. The Find menu contains the following items:

- [Find Components](#)
- [Find Usages](#) (for data model and transformation tabs only)
- [Find Unmapped Components](#) (for transformation tabs only)
- [Find Derived Types](#) (for data model and transformation tabs only)
- [Find Substitution Elements](#) (for data model and transformation tabs only)
- [Go To Component](#) (for data model and transformation tabs only)
- [Go To File](#)
- [Back](#)
- [Forward](#)
- [Select Left Tab](#)
- [Select Right Tab](#)
- [Select Left Component](#) (for data model and transformation tabs only)
- [Select Right Component](#) (for data model and transformation tabs only)

Find Components

This opens the Find Components dialog to allow you to perform searches on loaded data models or transformations.



Type the text on which your search is to be based in the Text to find field.

The Find Components dialog allows you to select various search criteria. The Parameters search criteria includes:

- Regular expression - This means that the value specified in the Text to find field is to be interpreted as a Java regular expression. See <http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html> for more details.
- Match case - This means that a case-sensitive search is to be performed.
- Match entire string - This means that only partial matches are to be ignored.

The Constraints search criteria includes:

- Match name - Match the name of the component.
- Match value - Match the specified value.
- Match documentation - Match the content of any documentation entries in components' annotations.
- Match appinfo - Match the content of any appinfo entries in components' annotations.

The Scope search criteria includes:

- In current - Search in the currently selected model only.
- In referenced - Search in the currently selected model and all models that it references.
- In all opened - Search in all loaded models.
- From current folder - Search in the currently opened folder.

The results of the search are displayed in the [Find window](#).

Find Usages

This finds usages of the selected component. The results are displayed in the [Find window](#).

Find Unmapped Components

This searches for components that are not already mapped in the transformation.

Find Derived Types

This finds types derived from the selected type. The results are displayed in the [Find window](#).

Find Substitution Elements

This finds elements in the substitution group of the selected element. The results are displayed in the [Find window](#).

Go To Component

This allows you to enter the name of a component within a data model into a dialog and go directly to it. This only works when a data model is already open.

Go To File

This allows you to enter the name of a file into a dialog and go directly to it. This only works when searching for a file in the visible Project tree.

Back

This reverses the last tab action performed. (This option is only enabled when more than one tab is opened in the main window.)

Forward

If you select "Back", this option is then enabled. It reverses the effects of the Back option. (This option is only enabled when more than one tab is opened in the main window.)

Select Left Tab

This opens the tab to the left of the currently opened tab. (This option is only enabled when more than one tab is opened in the main window.)

Select Right Tab

This opens the tab to the right of the currently opened tab. (This option is only enabled when more than one tab is opened in the main window.)

Select Left Component

This opens the component to the left of the currently opened component. (This is especially useful when using the keyboard).

Select Right Component

This opens the component to the right of the currently opened component. (This is especially useful when using the keyboard).

Tools Menu

The content of the **Tools** menu changes depending on the currently selected tab. The **Tools** menu contains the following items:

- [Import](#)
- [Export](#)
- [Preview](#)
- [Ant](#)
- [Selection Tool](#)
- [Print Tool](#)
- [Capture Tool](#)
- [Query Class Names](#)
- [Decrypt Test Data](#)
- [Globalise](#) (for data models and transformations only)
- [Migrate](#) (for data models only)
- [Apply Mappings](#)
- [Sort](#)
- [Inherit Components](#) (for data models only)
- [Auto Mapping](#) (for transformations only)
- [Launch Diff Tool](#)
- [Open In Diff Tool](#)
- [VCS](#)

Import

Contains the following submenu items:

- [Import XML Schema](#)
- [Import DTD](#)
- [Import WSDL](#)
- [Import XML Instance\(s\)](#)
- [Import Text File](#)
- [Import RELAX NG \(XML\) Schema](#)
- [Import RELAX NG \(Compact\) Schema](#)
- [Import Java Class](#)
- [Import Spreadsheet](#)
- [Import Database](#)

The **Choose Importer** option opens the Choose Importer dialog which makes it easy for you to select one of the options using the keyboard (You can still use the mouse to make the selection if you prefer). The numbers displayed beside each selection on the Choose Importer dialog equates to a number on the keyboard.

Export

Contains the following submenu items:

- [Export XML Schema](#)
- [Export DTD](#)
- [Export RELAX NG \(XML\) Schema](#)
- [Export RELAX NG \(Compact\) Schema](#)
- [Export HTML](#)

The **Choose Exporter** option opens the Choose Exporter dialog which makes it easy for you to select one of the options using the keyboard (You can still use the mouse to make the selection if you prefer). The numbers displayed beside each selection on the Choose Exporter dialog equates to a number on the keyboard.

Preview

Contains the following submenu items:

- [Preview XML Schema](#)
- [Preview DTD](#)
- [Preview RELAX NG \(XML\) Schema](#)
- [Preview RELAX NG \(Compact\) Schema](#)
- [Preview HTML](#)

The **Choose Preview** option opens the Choose Preview dialog which makes it easy for you to select one of the options using the keyboard (You can still use the mouse to make the selection if you prefer). The numbers displayed beside each selection on the Choose Preview dialog equates to a number on the keyboard.

Ant

Contains the following submenu items:

- **Run Ant Target** runs the selected Ant target.
- **Set Auto Target** sets the selected Ant target to be automatically run after deployment.

These options correspond to Ant related actions available in the [Ant window](#).

Selection Tool

Enables the selection tool. The selection tool is the standard mouse 'pointer'. Essentially it means the [print tool](#) and [capture tool](#) are disabled.

Print Tool

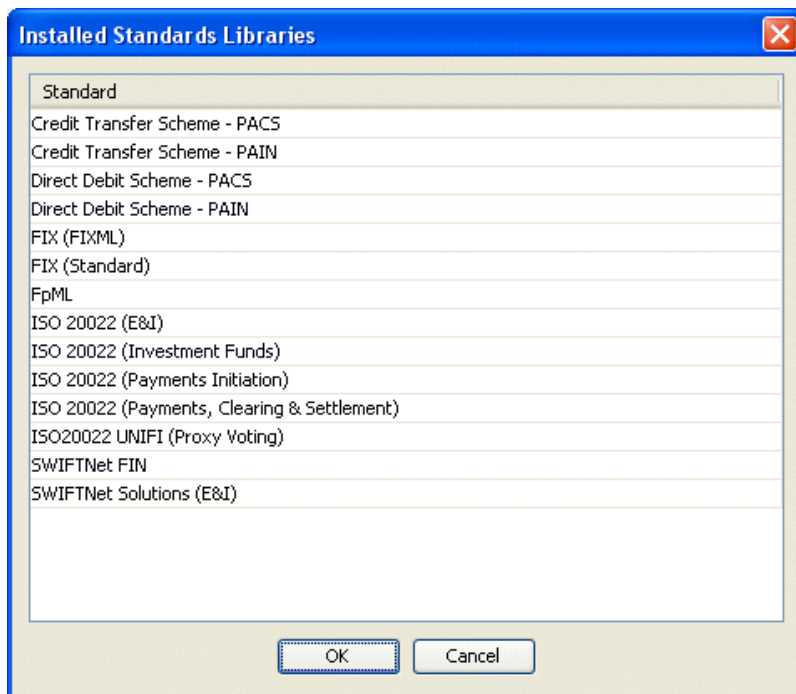
Enables the print tool, which attempts to print any GUI component clicked on (except for the menu bar). This is useful in situations where, for example, you require a quick printout of the structure of a data model, the graph view of a complex data type, or the composition of a transformation.

Capture Tool

Enables the capture tool. The capture tool is similar to the [print tool](#) except that instead of creating a print job, it will prompt for a graphical file format and a location on the file system for a snapshot of the selected component to be saved.

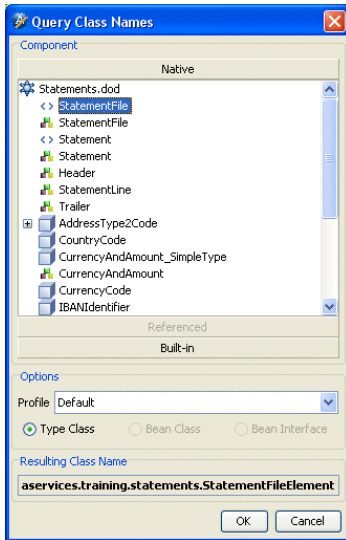
Decrypt Test Data

Allows you to decrypt standards library test data for use with [DataXtend SI](#). Select this option to open an Installed Standards Libraries dialog from where you can select the library for which you want to decrypt test data.

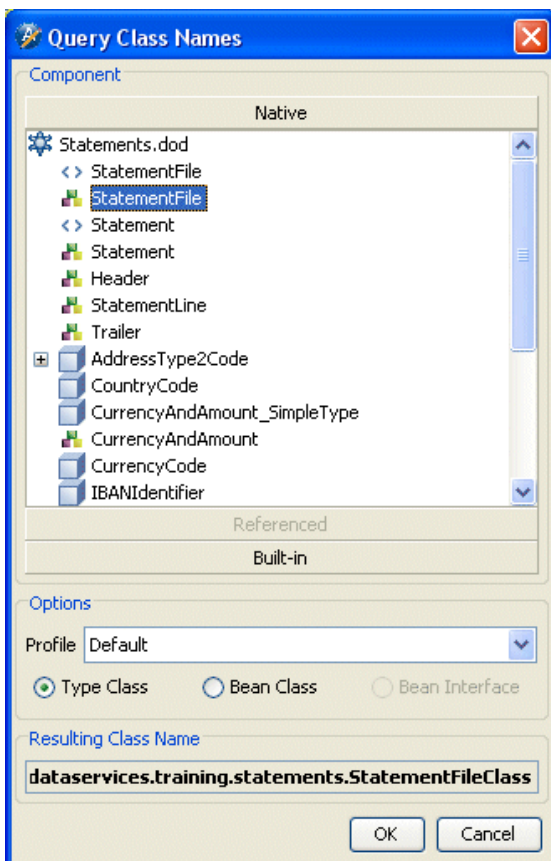


Query Class Names

This is only available when a data model is opened. It enables you to find out what will be the fully qualified deployed class name of a given data model component, based on the current profile. For example, if the component in question is a simple data type called StatementFile, you will see the following dialog:



If the type of the component is a complex data type, you will be prompted to select whether the component should be a bean or type interface before viewing the deployed class name:

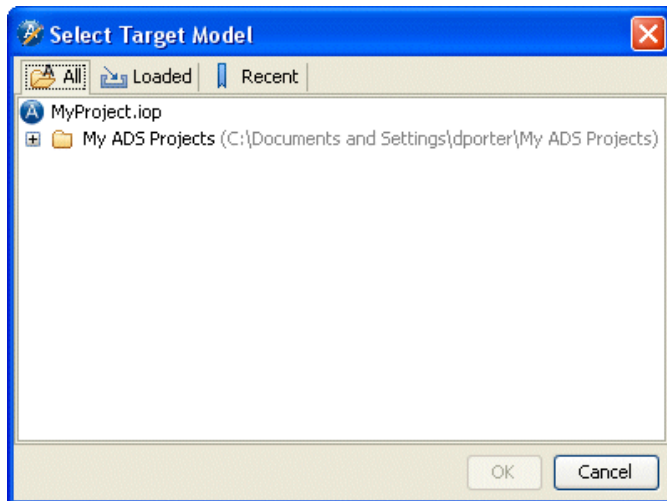


Globalise

This option is only available when a data model tab is selected. It globalizes a local node. If there is a classification group within the data model, you will be given a choice of where you want the node to globalize to.

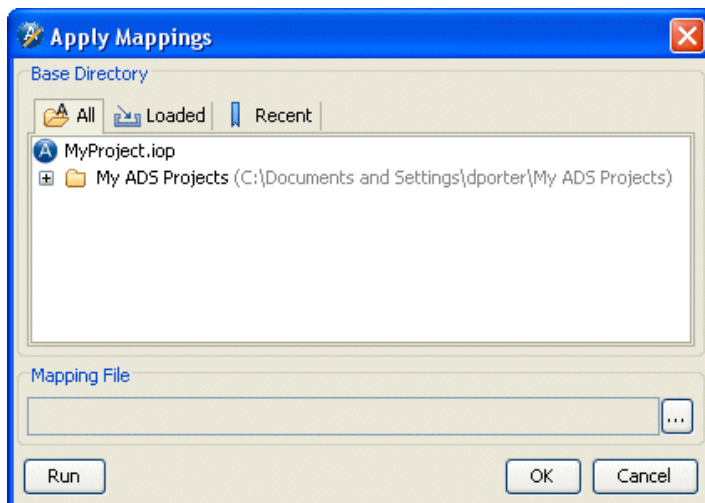
Migrate


This option is only available when a data model tab is selected. It migrates one or more components from one data model to another. A common use case is where a component has been developed in one data model, but it is later found to apply more generically to others and must therefore be moved to a 'common' model that can be imported by all interested parties. The migrate action works on the selected components and requires you to choose a target model.



Apply Mappings

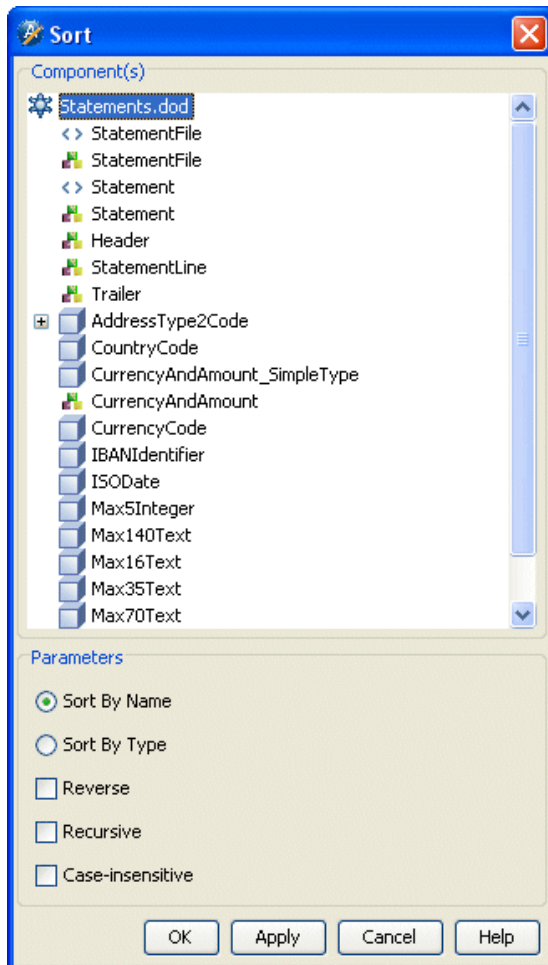
This opens the Apply Mappings dialog which allows you to alter a data model by applying mappings to it from an external file.



You can type the path to the required [mapping file](#) in the **Mapping File** text box or click the  button to browse for the required file.

Sort

This opens the Sort dialog which allows you to sort one or more classification groups according to a set of parameters.

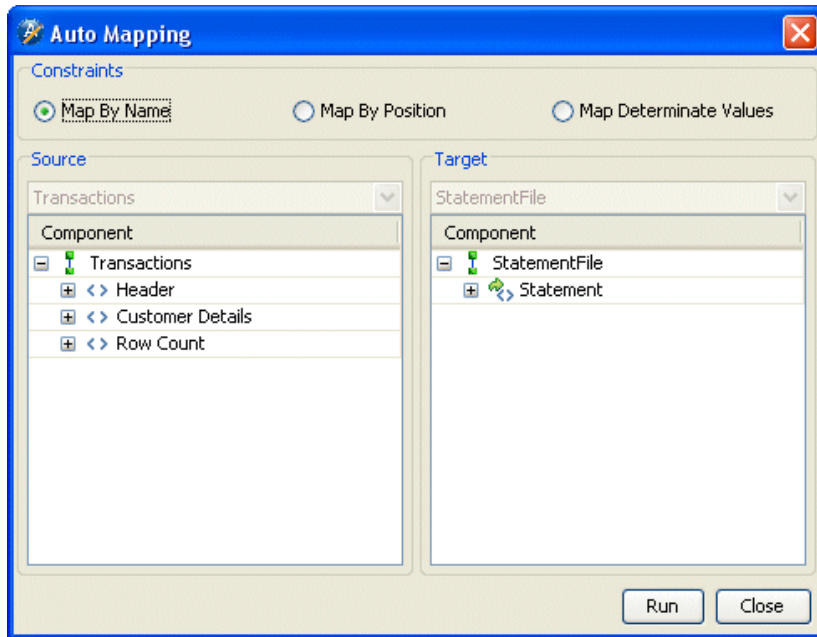


Inherit Components

This option is only available when a data model tab is selected. It adds inherited elements and attributes to the selected component.

Auto Mapping

This option is only available when a data model tab is selected. It opens the Auto Mapping dialog which allows you to map components using different constraints.



Launch Diff Tool

This launches the Diff Tool in a new window. If two data models are selected in the Project window, they will be loaded and the differences between them will be displayed in the Diff Tool window.

Open In Diff Tool

This opens the selected item(s) within the Diff Tool window.

VCS

You set the version control system ([VCS](#)) by selecting **Edit > Preferences**. If VCS is enabled, the name of the chosen VCS (for example, CVS, SVN, or ClearCase) is displayed as an option in the **Tools** menu.

If a chosen VCS name is displayed, the sub-menu for it contains the following items:

- **Add** allows you to add the selected file to the VCS currently in use.
- **Delete** allows you to remove the selected file from the VCS currently in use.
- **Check Out** (ClearCase) or **Update** (SVN) allows you to check out or update the selected file from the VCS currently in use. The exact operation is VCS dependant.
- **Check In** (ClearCase) or **Commit** (SVN) allows you to check in or commit the selected file from the VCS currently in use. The exact operation is VCS dependant.

Deploy Menu

The **Deploy** menu contains the following items:

- [Run](#)
- [Reload Active Run Configuration](#)
- [Run File](#)
- [Build File\(s\)](#)
- [Verify Files\(s\)](#)
- [Run Component](#)
- [Build Component\(s\)](#)
- [Verify Component\(s\)](#)
- [Build Project](#)
- [Build Namespace](#)

Run

Opens the [Run Wizard](#).

Reload Active Run Configuration

Reloads and rebuilds (if necessary) instances of the model in the currently active run tab. This is useful, for example, if you receive some parsing errors on a model. It means that you can modify the model and try reloading the data without having to close the run tab for that model.

Run File

Opens the [Run Wizard](#) for a specific data model (.dod) or transformation (.tfd) file that you select in the Project window. If selected for a data model, you are prompted to select a data model component.

Build File(s)

Builds Java class instances of the data model (.dod) or transformation (.tfd) files that you select in the Project window.

Verify Files(s)

[Verifies](#) the data model (.dod) or transformation (.tfd) files that you select in the Project window are valid and can be built.

Run Component

Opens the Run Wizard for a specific data model component that you select in the Explorer window.

Build Component(s)

Builds Java class instances of the data model components that you select in the Explorer window.

Verify Component(s)

[Verifies](#) specific data model components that you select in the Explorer window.

Build Project

Builds Java class instances for the currently selected project, prompting you to load all data models and transformations first.

Build Namespace

Opens a dialog which allows you to select a particular namespace for which you want to build Java class instances, prompting you to load relevant data models and transformations first.

Window Menu

The **Window** Menu allows you to switch between windows; to load, save and reset the screen layout; and to toggle the auto-hide behavior of the various windows. It contains the following items:

- [Window Shortcuts](#)
- [Save Layout](#)
- [Load layout](#)
- [Reset Layout](#)
- [Toggle Auto Hide](#)

Window Shortcuts

Select these various options (or type the corresponding keyboard shortcuts) to switch focus between the various windows.

Save Layout

This allows you to save the current window configuration as the default layout.

Load Layout

This allows you to reposition the windows according to the last-saved default layout.

Reset Layout

This allows you to reset the default layout to the original system default.

Toggle Auto Hide

This allows you to toggle the auto-hide behavior of the windows.

Help Menu

The **Help** menu contains the following items:

- [Install Plugin](#)
- [Home Page](#)
- [User Guide](#)
- [What's This?](#)
- [Quick Help](#)
- [Check for Updates](#)
- [Documentation](#)
- [View License Agreement](#)
- [View System Properties](#)
- [Standards Libraries](#)
- [Technical Support](#)
- [Contact Us](#)
- [About](#)

Install Plugin

Allows you to download various componentized plug-ins as part of your ADS installation. These are the same plug-ins that you can download from the Designer Home page. These plug-ins are persisted as .plg files and can be downloaded separately into ADS Designer.

Home Page

Displays the ADS Designer [Home page](#) in the main window.

User Guide

Opens this help system.

What's This?

Activates context-sensitive help for the application. This appears in the form of a cursor with a question mark attached to it. Wherever you click the cursor in the Designer, this Help System opens at the relevant page for that part of the Designer.

Quick Help

Relates to the off-the-shelf functions that you can use within transformations. This option is only enabled when you click a particular function in a transformation. Clicking this option displays context-sensitive help for the currently selected function.

Check for Updates

Allows you to check for updates of the installed components. If an update is available you will be given the option of downloading it.

Documentation

The Documentation sub-menu contains the following items:

API Javadoc

Displays the application program interface (API) for Artix Data Services.

Ant Tasks

Provides a short description of each Ant task and a link to the complete documentation.

Examples

Provides documentation on a broad but common range of ADS functionality.

Online Documentation

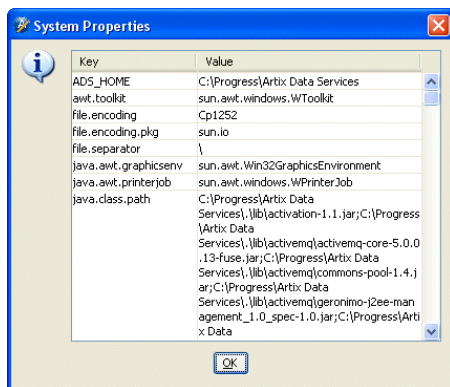
Opens the online documentation page for this ADS release.

View License Agreement

Opens the license agreement which contains detailed information of the license components and pre-built models and 3rd party credits.

View System Properties

Opens a dialog displaying the system properties of the JVM running ADS. For example:



Standards Libraries

Link to the description of the different standards libraries supported by ADS.

Technical Support

Opens the Progress Software Technical Support page.

Contact Us

Opens the ADS Contact page.

About

Displays ADS Designer version information.

Projects

In ADS, projects are used to hold the data models, transformations and other working files for the various tasks you wish to accomplish. [Creating a project](#) is therefore a prerequisite to performing any other task.


An ADS project is a collection of the following:

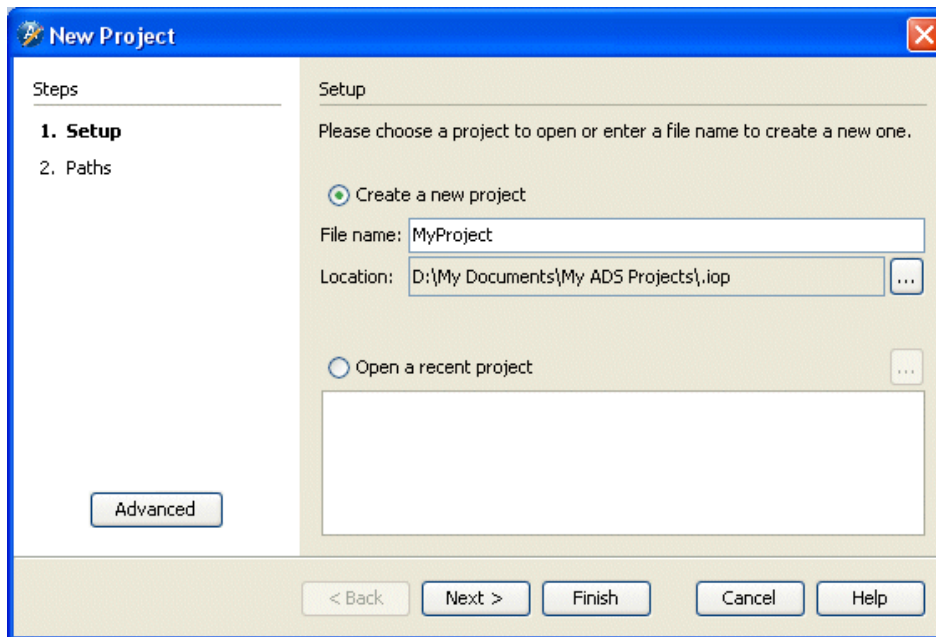
- [Paths](#) (working directories for the project).
- [Properties](#) (determining how the project is stored and accessed).
- [Profiles](#) (determining the behavior of code generated from the project).
- [Preferred Aliases](#) (allowing different naming conventions for the same components).

These various settings can be accessed by right-clicking the project (root) node in the Project window or by selecting **Edit > Project Properties** from the menu bar.

Creating a Project

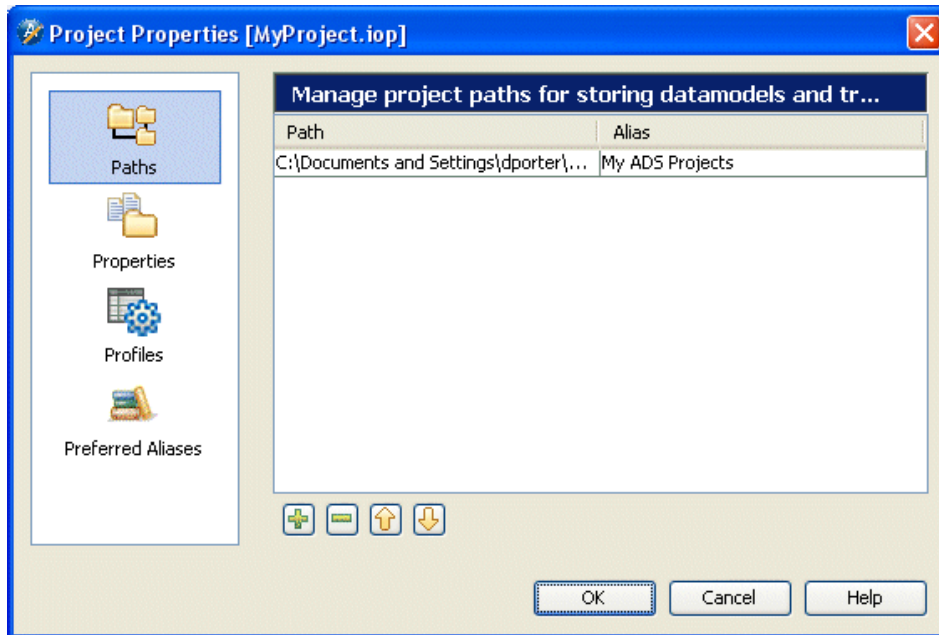
To create a project:


1. Do one of the following:
 - Click the **New** drop-down icon in the toolbar and select **New Project**.
 - Select **File > New > Project** from the menu bar.
2. In the Setup panel, type a unique project name in the **File name** field. The project name is added to the **Location** field and is assigned an .iop extension.
To select an alternative location for your project file, click the  button. Click **Next**.





Paths

Paths are the working directories for a project. They are the locations on your file system in which you store the data models and transformations relating to the project. A project can contain any number of paths and will commonly use aliases to shorten the filenames so that they are more identifiable.



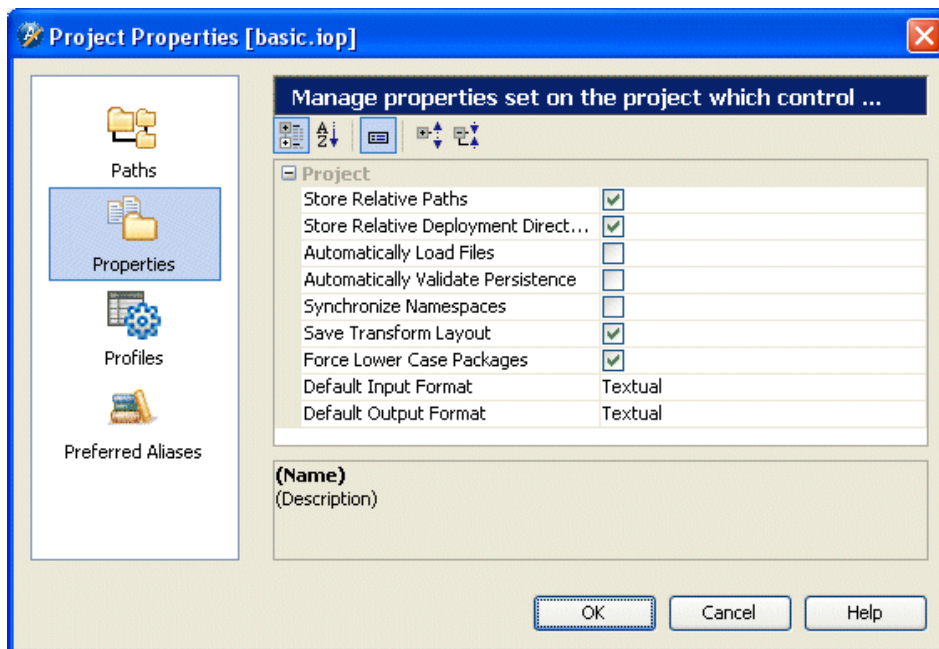
Click the  and  icons to add or remove paths respectively.

Click the  and  icons to move paths up or down the list.

Note: The directories specified need not be on your local file system. They could be on a remote or removable drive.

Properties

Each project contains a set of general properties that allow you to customize how the project file is stored and accessed.



These general properties include:

- [Store Relative Paths](#)
- [Store Relative Deployment Directories](#)
- [Automatically Load Files](#)
- [Automatically Validate the Persistence](#)
- [Default Input Mask](#)
- [Default Output Mask](#)
- [Force Lower Case Packages](#)
- [Automatically Load Files](#)
- [Synchronize Namespaces](#)

Store Relative Paths

This indicates whether the project is to store the paths relative to the project file, making it possible to move the project file and referenced paths to a different location or file system. If this is not enabled, absolute paths are stored, thereby allowing you to move the project file to a different location while keeping its references intact.

Store Relative Deployment Directories

This indicates whether the project is to store the deployment directories relative to the project file, making it possible to move the project file and still deploy to the same relative directory. If this is not enabled, absolute paths are stored, thereby allowing you to move the project file to a different location and still deploy to the same location.

Automatically Load Files

This indicates whether all referenced definitions should be loaded when the project is opened.

Warning: This will increase the start time of the Designer and will therefore not be suitable for big projects with many large data models.

Note: A change to this property only takes effect when the project is reloaded.

Automatically Validate the Persistence

This indicates whether all internal settings should be automatically validated when you load or save a component.

Synchronize Namespaces

This indicates whether all profile namespaces are to be synchronized, thereby reducing the amount of work needed to maintain them if all package names and default masks are the same.

Warning: Enabling this option will propagate the default profiles namespace mappings to all other descriptors.

Force Lower Case Packages

This indicates whether all package names should be converted to lower case, to follow standard Java coding rules. This might need to be disabled when writing package structure that include capitals.

Default Input Format

This allows you to select the default input mask that is to be used for new data models (that is, the format in which your input data will be displayed by default). The available options are:

- Textual
- XML
- Binary
- Java Class
- SWIFT
- Crest ISO

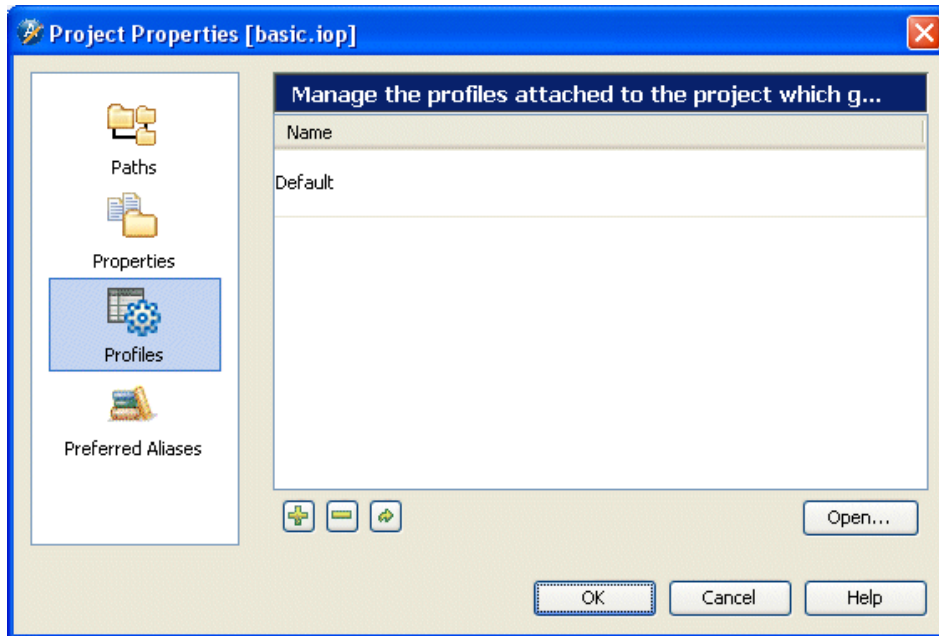
Default Output Format

This allows you to select the default output mask that is to be used for new data models (that is, the format in which your output data will be displayed by default). The available options are:

- Textual
- XML
- Binary
- Java Class
- Interchange
- Tag Value Pair


Profiles


Each project has a default profile that contains sets of parameters controlling various aspects of the code that ADS generates.



A project may be assigned multiple different profiles, with each profile containing a different range of settings.

To add a new profile, click the  icon and set the new profile name.

To remove a profile, click that profile in the list and then click the  icon.

To rename a profile, click that profile in the list and then click the  icon.

To open a particular profile for editing, double-click that profile in the list, or click that profile and then click **Open**. This opens the [Profile Settings](#) window.

Profile Settings

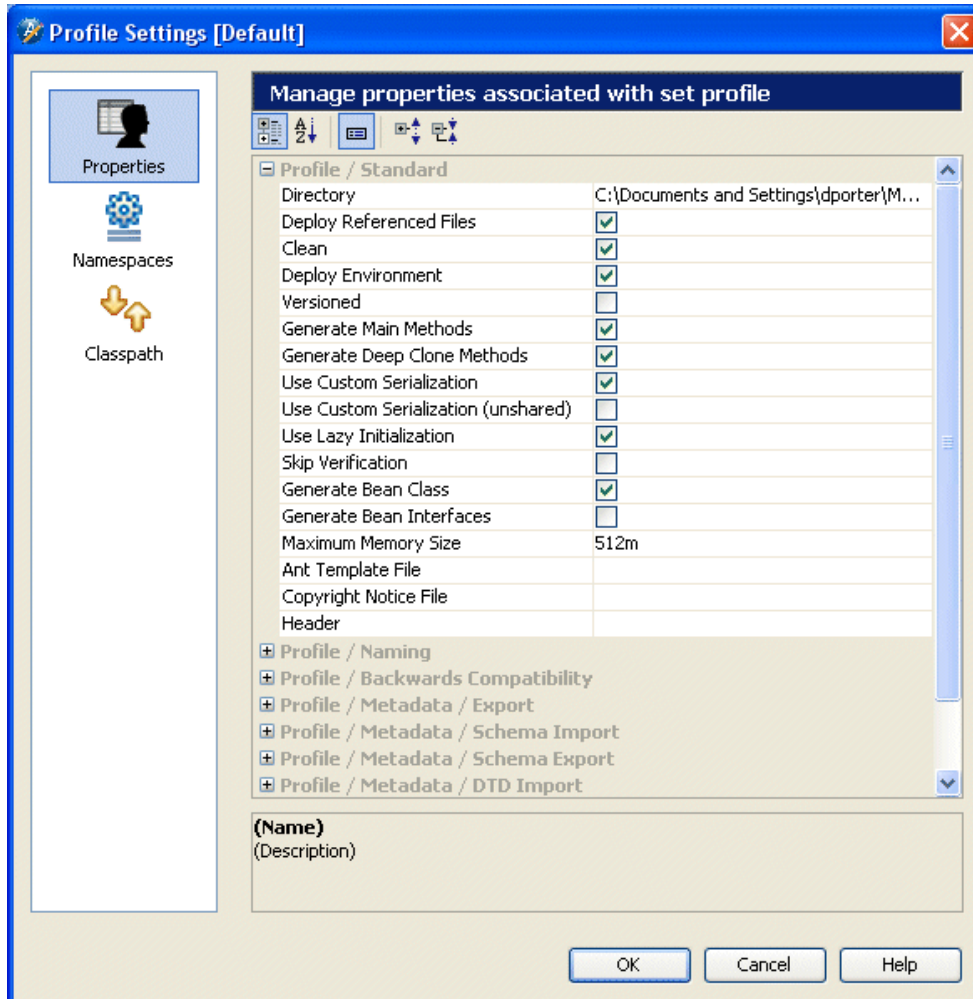
Profile settings are sets of parameters that control various aspects of the code that ADS generates. They are accessed by right-clicking on the project (root) node in the [Project window](#) or by selecting **Edit > Profile Settings** from the menu bar.

[Profile Settings](#) contain the following:

- [Properties](#)
- [Namespaces](#)
- [Classpath](#)

Profile Properties

The properties available in a project profile are described in detail below.



Available properties are grouped into 10 sections:

- [Standard](#)
- [Naming](#)
- [Backwards Compatibility](#)
- [Metadata Export](#)
- [Metadata Schema Import](#)
- [Metadata Schema Export](#)
- [Metadata DTD Import](#)
- [Metadata Java Class Import](#)
- [Metadata HTML Export](#)
- [Metadata Spreadsheet Import](#)

Standard Properties

Standard properties include:

- [Ant Template File](#)
- [Clean](#)
- [Copyright Notice File](#)
- [Deploy Environment](#)
- [Deploy Referenced Files](#)
- [Directory](#)
- [Generate Bean Class](#)
- [Generate Bean Interfaces](#)
- [Generate Deep Clone Methods](#)
- [Generate Main Methods](#)
- [Header](#)
- [Maximum Memory Size](#)
- [Skip Verification](#)
- [Use Custom Serialization](#)
- [Use Custom Serialization \(unshared\)](#)
- [Use Lazy Initialization](#)
- [Versioned](#)

Ant Template File

Specifies the template Ant build file used to construct individual build files for each deployment. By default, this property should point to the build-template.xml file delivered with the toolkit. At build time, namespace-specific build files are constructed by replacing various placeholders with the specific values for the deployment. The following replacements will occur at build 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
- @cvsheader@ is replaced by the default CVS header

Clean

Indicates whether to clean the <deployment directory>/src directory before generating code.

Copyright Notice File

This property allows you to specify the path to a text file containing any copyright notice that you want to include in the source code generated by the tool.

Deploy Environment

Indicates the extent of the environment to create when running a build. Disabling this property means that the build process just creates the Java source files for the deployed model. Enabling this property means that a complete deployment environment is created consisting of src, build and lib directories. An Ant build file is created per deployment for compilation of the contents of the src directory into the build directory.

Deploy Referenced Files

Indicates whether the data models associated with a transformation should automatically be built with it.

Directory

This property allows you to define a path to the root of the deployed code tree.

- Code is generated into <directory>/src/java.
- Code is compiled into <directory>/build/classes.
- Code is jar'd into <directory>/build/libs.
- Javadoc is built under <directory>/docs/api.
- Javadoc for all deployed code (linked together) is built under <directory>/docs/api-all.

Generate Bean Class

Indicates whether to generate specialized subclasses of the API class ComplexDataObject. These subclasses will provide type-safe get and set methods with return values and arguments respectively corresponding to the appropriate child element types. See the [FAQ](#) for a detailed description.

Generate Bean Interfaces

Indicates whether to additionally generate interfaces exposing the bean methods described under 'Generate Bean Class'. If set to 'true', the classes generated as a result of the 'Generate Bean Class' flag implement the corresponding interfaces. This property is only effective if the 'Generate Bean Class' property is set to 'true'. See the [FAQ](#) for a detailed description.

Generate Deep Clone Methods

Indicates whether to generate deep clone methods in bean classes.

Generate Main Methods

Indicates whether to generate main methods in deploy element classes for test and demonstration purposes.

Header

This property allows you to set the text that will appear in the class/interface header comment before any disclaimers.

Maximum Memory Size

Specifies the maximum size of the memory used for the underlying virtual machine (VM) in Ant build files. The default is 512 MB.

Skip Verification

Indicates whether to skip the automatic verification during deployments and metadata exports.

Use Custom Serialization

Indicates whether ADS will generate class specific `readObject()` and `writeObject()` methods, providing better serialization performance.

Use Custom Serialization (Unshared)

Indicates whether the generated `readObject()` and `writeObject()` methods should employ bit fields and unshared string serialization to provide better serialization performance when you want to instance data with very little repetition.

Use Lazy Initialization

Indicates whether ADS will build code that only initializes the singleton type hierarchy when required rather than at the point of instantiating the root.

Versioned

Indicates whether to create a versioned package structure. Enabling this property means that code relating to [bean classes and interfaces](#) is deployed into a package whose name includes the version number of the originating data model.

For example, suppose a data model has been assigned the [namespace](#) `com.progress.ads.deployed.examples.simple` and this data model is at version 1.0.5. Code would be deployed into the package `com.progress.ads.deployed.examples.simple.v1.v0.v5`.

Naming Properties

Naming properties include:

- [Attribute Class Suffix](#)

- [Attribute Group Class Suffix](#)
- [Bean Class Suffix](#)
- [Bean Interfaces Suffix](#)
- [Bean Local Group Suffix](#)
- [Bean Local Suffix](#)
- [Element Class Suffix](#)
- [Element Group Class Suffix](#)
- [Enumeration Class Suffix](#)
- [Replacement String](#)
- [Shorten Local Transform Class Names](#)
- [Type Class Suffix](#)
- [Validation Rule Class Suffix](#)

Attribute Class Suffix

This property allows you to set the suffix that will be appended to the attribute name, to formulate a class name for classes generated from attributes. For example, using the settings shown above, the attribute name will give rise to a deployed class NameAttribute.java.

Attribute Group Class Suffix

This property allows you to set the suffix that will be appended to the attribute group name, to formulate a class name for classes generated from attribute groups.

Bean Class Suffix

This property allows you to set the suffix that will be appended to the complex data type name, to formulate a class name for 'bean' classes generated from complex data types.

Note: This is only used when the Generate Bean Class property is enabled.

Bean Interfaces Suffix

This property allows you to set the suffix that will be appended to the complex data type name, to formulate a name for 'bean' interfaces generated from complex data types.

Note: This is only used when the Generate Bean Interfaces property is enabled.

Bean Local Group Suffix

The property allows you to set the additional suffix that will be appended to deployed bean classes and interfaces from locally defined types of element or attribute groups.

Bean Local Suffix

This property allows you to set the additional suffix that will be appended to deployed bean classes and interfaces that are derived from local types.

Element Class Suffix

This property allows you to set the suffix that will be appended to the element name, to formulate a class name for classes generated from elements. For example, using the settings shown above, the element 'SimpleFile' will give rise to a deployed class SimpleFileElement.java.

Element Group Class Suffix

This property allows you to set the suffix that will be appended to the element group name, to formulate a class name for classes generated from element groups.

Enumeration Class Suffix

This property allows you to set the suffix that will be appended to the enumeration name, to formulate a class name for classes generated from enumerations. For example, using the settings shown above, the enumeration 'Currencies' will give rise to a deployed class CurrenciesEnum.java.

Replacement String

This property allows you to specify a string detailing character replacements that should be performed at build time. The value specified must match the pattern `(.=.(.=.?)*)?` where the left hand side of each "=" expression is substituted by the right.

Shorten Local Transform Class Names

Indicates whether class names of local transformations should be shortened.

Type Class Suffix

This property allows you to set the suffix that will be appended to the type name, to formulate a class name for classes generated from data types. For example, using the settings shown above, the simple data type 'StringField' will give rise to a deployed class StringFieldClass.java.

Validation Rule Class Suffix

This property allows you to set the suffix that will be appended to the validation rule name, to formulate a class name for classes generated from validation rules. For example, using the settings shown above, the validation rule 'MT103C1' will give rise to a deployed class MT103C1Rule.java.

Backwards Compatibility Properties

Backwards Compatibility properties include:

- [Bean Named Value Methods](#)
- [Bean Subfield Lists](#)
- [Skip XPath Verification](#)
- [Versioned Type Hierarchy](#)
- [Skip Default & Fixed Value Initialization](#)
- [Ignore Target Package](#)

Bean Named Value Methods

Indicates whether bean classes for complex data types derived from simple data types should expose a particular type of get/set method.

For example, the complex data type 'Molecule' is derived from (extends) the simple data type 'Atom' (perhaps because we need to have a type that is like 'Atom', but can have an attribute, for example). The simple data type 'Atom' is an extension of the primitive type 'GenericString'.

If the Bean Named Value Methods property is enabled on the profile (and we are also generating bean classes, so the Generate Bean Class property is also enabled), the generated bean class for the complex type 'Molecule' will expose the methods getMoleculeValue() and setMoleculeValue(), returning and taking an argument of type 'String'.

This property is primarily available for backwards compatibility with ADS version 2.

Bean Subfield Lists

Indicates how bean methods dealing with repeating elements are implemented. If this property is enabled, all such methods accept and return java.util.List objects. If this property is disabled, all such methods use type specific arrays.

This property is primarily available for backwards compatibility with ADS version 2.

Ignore Target Package

Indicates whether to ignore the 'target package' property and have the deployment package derived solely from the 'target namespace' property.

Skip Default & Fixed Value Initialization

Indicates whether to skip automatic initialization of default and fixed values. If this property is disabled, default and fixed values will be automatically initialized.

Skip XPath Verification

Indicates whether to skip verification of XPath statements defined in validation rules. This property should be enabled for data models created prior to version 3.5, because prior to this, invalid XPath was permitted.

Versioned Type Hierarchy

Indicates whether to include data types, elements, attributes, enumerations and validation rules in a versioned package structure, as well as [bean classes and interfaces](#).

This property is primarily available for backwards compatibility with ADS version 2.

Metadata Export Properties

Metadata Export properties include:

- [Append Version to Filename](#)
- [Replace Spaces In Filename](#)

Append Version to Filename

When exporting an ADS data model to an XML schema, indicates whether the version number of the data model should also be appended to the filename.

Replace Spaces in Filename

Indicates whether to replace spaces with underscores in the name of the exported file.

Metadata Schema Import Properties

Metadata Schema Import properties include:

- [Annotation Whitespace](#)
- [Import Annotations](#)
- [Import Ecore Annotations](#)
- [Use 'Subgroup'](#)
- [Use 'Subgroup' Local Indices](#)

Annotation Whitespace

This property allows you to select how the whitespace of imported annotations is to be handled. There are three options:

- Preserve - keeps all whitespace as it was in the source document.
- Replace - replaces tabs (0x09), line feeds(0x0a) and carriage returns (0x0d) with spaces (0x20).

- Collapse - performs a replace first, then collapses multiple space characters into a single space.

Import Annotations

When importing an XML schema into ADS, indicates whether xs:annotation components in an XML schema should be imported as [annotation](#) objects in the resulting ADS data model.

Import ECore Annotations

Indicates whether ECore annotations should be imported.

Use 'Subgroup'

Indicates whether local groups in schemas should be imported with the name "SubGroupX" (where X is a unique number) or whether group names should be based on their contents.

For example, in a schema a local sequence group contains an element 'A', a local choice group contains 'B' and 'C', and a local all group contains 'D' and 'E'. If this property is disabled, the resulting element group in ADS will be called "A then B or C then D and E", because sequence groups use the compositor "then", choice groups use "or" and all groups use "and".

Use 'Subgroup' Local Indices

Indicates whether the uniquely identifying index applied to imported local groups named 'Subgroup' should be relative to the containing global data type (when set to true) or the containing schema (when set to false).

Metadata Schema Export Properties

Metadata Schema Export properties are:

- [Export Annotations](#)
- [Export ECore Annotations](#)
- [Export Default Occurrences](#)
- [Flatten Directories](#)
- [Inline](#)
- [Inline Minimal](#)
- [XML Namespace Schema Location](#)

Export Annotations

When exporting an ADS data model to an XML schema, indicates whether annotations are written as xs:annotation elements in the resulting schema.

Export ECore Annotations

Indicates whether ECore annotations should be exported.

Export Default Occurrences

When exporting an ADS data model to XML schema, this property determines whether components with cardinality 1 are exported with the default occurrence constraint attributes minOccurs=1, maxOccurs=1.

Flatten Directories

Indicates whether directories should be flattened during exports.

Inline

When exporting an ADS data model to an XML schema, indicates whether [included](#) and [redefined](#) models are written to the exported schema ("inlined") in their entirety or not. See also the [Inline Minimal](#) option for a less 'drastic' version of the same principle that inlines only those imported or included types that are actually used in the model being exported to the schema.

Inline Minimal

When exporting an ADS data model to an XML schema, indicates whether components defined in [included](#), [imported](#) and [redefined](#) models that are used in the local model being exported are written to the exported schema as if they were local as well (that is, "inlined").

XML Namespace Schema Location

Specifies the schema location to be used for the XML namespace import (if present). Leave this blank if you want to omit the schema location attribute.

Metadata DTD Import Properties

Metadata DTD Import properties are:

- [Import Via Castor](#)
- [Import Mixed Content As String](#)

Import Via Castor

Indicates whether the Castor XML libraries are used to import DTD's. This was the default prior to version 3.4.2.

Import Mixed Content As String

Indicates whether mixed content types are to be derived from String types instead. This is enabled by default for new projects. This should be disabled for existing projects, to preserve backwards compatibility.

Metadata Java Class Import Properties

Metadata Java Class Import properties are:

- [Exclude Primitive Flags](#)
- [Access Modifier Level](#)
- [Include Transient Fields](#)
- [Import Simple Data Types as Attributes](#)

Exclude Primitive Flags

When importing Java bean classes previously built in ADS (or classes which use a similar bean style), indicates whether isXSet methods should be ignored.

Access Modifier Level

This property allows you to set the level of access modifiers to be imported. By default, it is set to "Public".

Include Transient Fields

Indicates whether the flags which store the state of transient fields should be included during import.

Import Simple Data Types as Attributes

Indicates whether to import Simple DataTypes as attributes.

Metadata HTML Export Properties

Metadata HTML Export properties are:

- [Sort By Component](#)
- [Print Namespace Prefixes](#)
- [Print Referenced Models](#)
- [Print Glossary](#)

Sort By Component

Indicates whether to sort the components in the generated HTML by their classification or whether to leave them in the order in which they appear in the model.

Print Namespace Prefixes

Indicates whether to append namespace prefixes to component names in the generated HTML.

Print Referenced Models

Specifies whether to export HTML for any models that are referenced (imported or included) in the model that you are exporting.

Print Glossary

Indicates whether to print a glossary of terms used in the export.

Metadata Spreadsheet Import Properties

You can set the following property when importing a spreadsheet:

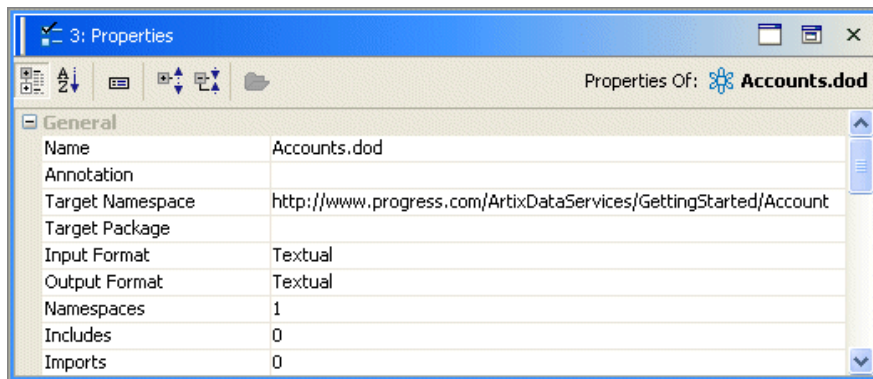
Process Spreadsheet Formula

Specifies whether to process any formulas contained in the spreadsheet. By default, this option is selected and the result of the formula is analyzed by the importer. If this option is disabled, the formula is imported as a string starting with the equals (=) character.

Namespaces

Profile settings hold tables of namespace/package name mappings. A namespace/package mapping must be made for every namespace that a project knows about.

Every data model and transformation can have an associated target namespace which is specified in the Properties window.

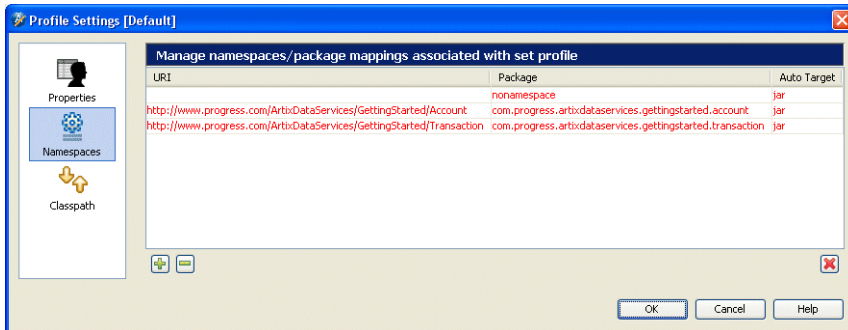


By specifying a namespace you are asserting that nothing else exists in that namespace with an identical name and classification. This implies, once the verification algorithm has passed, that the build process is free to regard any pre-existing file to be an earlier version of the component that to be deployed. If a data model or transformation does not define a target namespace (that is, the properties value is empty), by default it is deployed into the "nonamespace" package.


Data models can also have import or include relationships with other data models, as follows:



- A data model includes another if it uses definitions from it and shares the same namespace.
- A data model imports another if it uses definitions from it but does not share the same namespace.
- A data model redefines another if it uses definitions from it, shares the same namespace but wishes to change the definitions of one or more of its components.

The list of namespaces held by a profile primarily governs the mapping between namespaces (and hence data models and transformations) and generated Java packages:



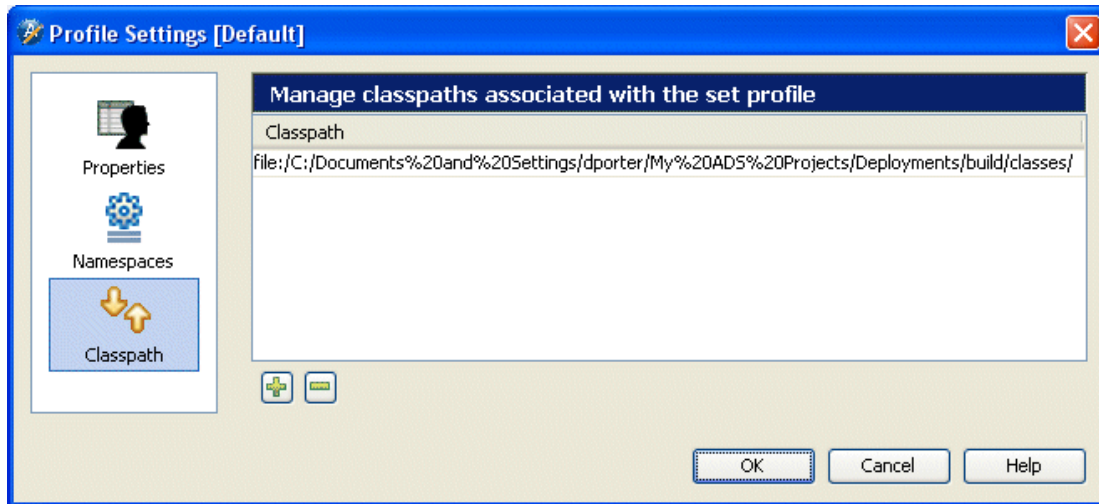
In the above example, classes from components in the namespace with the URI `http://www.progress.com/ArtixDataServices/GettingStarted/Account` are deployed to a package `com.progress.artixdataservices.gettingstarted.account`.

Namespace mappings are added to profiles when a model or transformation is opened or a target namespace property is changed. They can also be added from the above screen by clicking on the  icon.

The rows highlighted in red indicate namespace descriptors unused by currently loaded models. They are therefore eligible for removal which can be done either by highlighting the desired row and clicking  icon, or by clicking  which will remove ALL unused entries. The Auto Target column allows users to set which ant target, in the namespace build file, to execute at build time. By default, this is set to "compile".

Note: Two components (for example, a complex data type and an element) can exist in the same namespace (and in fact the same data model or classification group) even if they have the same name. This is because they are of different classifications. Complex and simple data types are classified the same, so it is NOT valid to have a simple type and a complex type with the same name.

Classpath



The classpath specified in the profile settings has a number of uses:

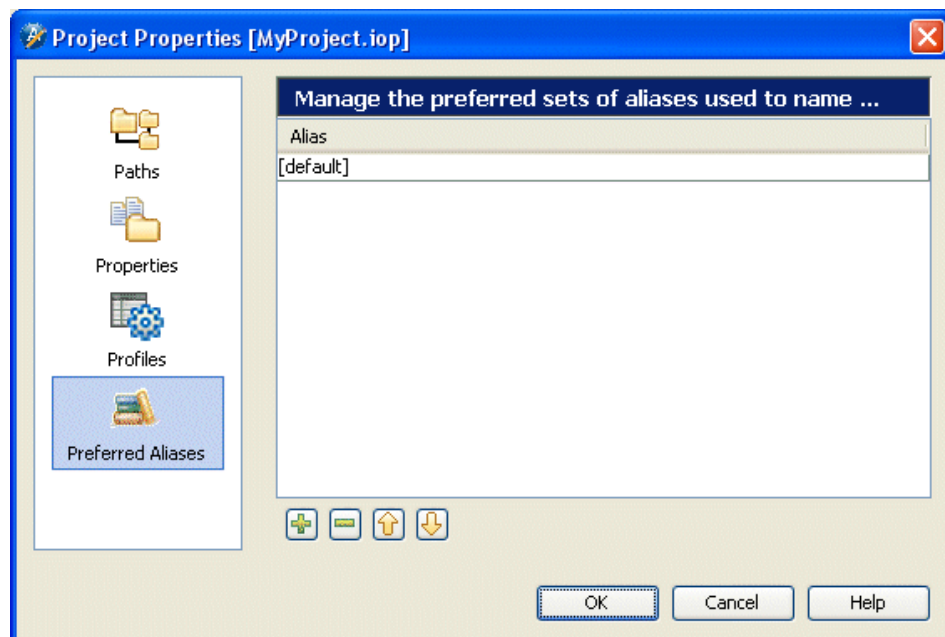
- If a data type has either the Bean Interface(s) or Bean Superclass property specified, Designer uses the classpath to look for the specified interfaces/superclasses at build time and validate that there are no conflicts in method signatures.
- The Run Wizard uses any directories specified here as possible locations of classes and JAR files to use.
- When the profile setting is marked as versioned (see [properties](#)), the classpath is used to try to find earlier compiled versions of your classes and interfaces, so that it can establish whether it can extend or implement them.
- When using the Java class import optional feature, the classpath is used to look up the classname via reflection.



The path to the /build/classes directory, under the specified deployment directory for the profile setting, is included by default.



Preferred Aliases

Preferred aliases allow you to use different sets of names for the same components in a data model. This feature is useful, for example, if you want to create 'internationalized' models where different sets of names in different languages are defined for the same components. For example, you might wish to choose between seeing English and French names within the same data model. Using preferred aliases allows you to easily switch from one set of names to another.

Another example for the use of preferred aliases would be if you wanted to choose between seeing different sets of names for standards libraries such as SWIFT. In this case, you might wish to choose between seeing a set of technical names and a set of business names within the same data model.



Click the  and  icons to add or remove aliases.

Click the  and  icons to move aliases up or down the list.

Data Models

A data model is an abstract description of a real-world data structure. It represents the top-most level of containment in ADS and encapsulates a number of different [model components](#).

It also stores a number of properties that are used by the contents of the model, such as several default values which are inherited by components if not declared locally. [Includes](#), [imports](#), [redefines](#) and [namespaces](#) are all held as [properties](#) of data models.

Data models are persisted in data object definition (.dod) files.

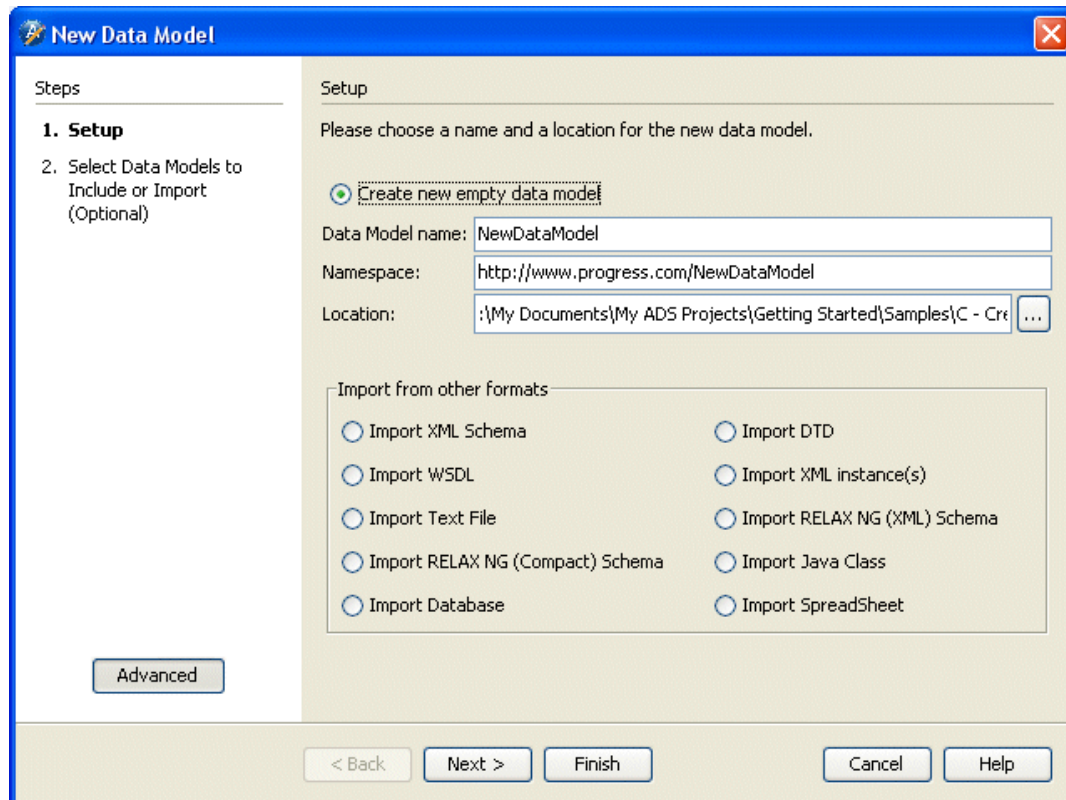
You can create a data model [manually](#) or by [importing data](#) from some other source.


Creating a Data Model Manually

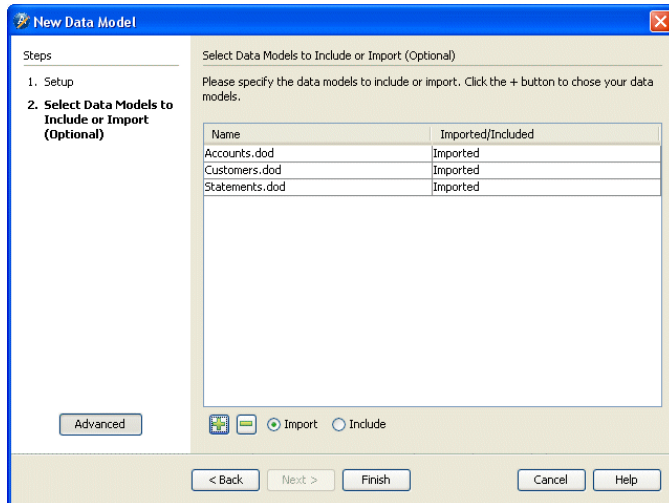
ADS allows you to create data models either manually or by importing data from other formats.




To create a data model manually::

1. Ensure that your project is open in the Project window.
2. In the Project window, navigate to the folder where you wish to store the data model, right-click and select **New > Data Model**. Alternatively, select **File > New > Data Model** from the menu bar. This opens the Setup panel of the New Data Model wizard.



3. Ensure that the Create new empty data model radio button is selected.
4. Type the name you want to assign to your data model in the File name field.
5. If you do not wish to accept the default namespace, type an alternative in the Namespace field.
6. If you do not wish to accept the default location, click the  button beside the Location field to select an alternative location for your data model.
7. If you do not wish to include or import another data model in your new data model, skip to step 9. If you wish to include or import another data model in your new data model, click the **Advanced** button and then click **Next**. This opens the Select Data Models to Include or Import panel.



8. Click the Import or Include radio button, depending on which action you wish to take.
9. Click the  icon. This opens either a New Import or New Include dialog, depending on which radio button is selected.
10. Navigate to the data model you wish to include or import, and then click **OK**. The selected data model is then added to the list. Note: You can import or include as many data models as you wish. Click the  and  icons to respectively add or remove data models.
11. Click **Finish**. At this point, the new data model is loaded and displayed in both the Project window and Explorer window. See [Adding Components to a Data Model](#) for details of what to do next.

Adding Components to a Data Model

A *component* is any object that can be defined in a [data model](#). To add a component to a data model, ensure that your data model is open in the Explorer window.

You can add the following components to a data model:

- [Classification Group](#)
- [Complex Type](#)
- [Atomic Simple Type](#)
- [List Simple Type](#)
- [Union Simple Type](#)
- [SWIFT Field Type](#)
- [Element](#)
- [Element from Type](#)
- [Attribute](#)
- [Element Group](#)
- [Attribute Group](#)
- [Any Element](#)
- [Any Attribute](#)
- [References](#)
- [Enumeration](#)
- [Validation Rule](#)

Model Components

Model component is the generic term used for the different objects that can be defined in a [data model](#). These are:

- [Simple Data Types](#)
- [Complex Data Types](#)
- [Elements](#)
- [Element Groups](#)
- [Any Elements](#)
- [Attributes](#)
- [Attribute Groups](#)
- [Any Attributes](#)
- [Enumerations](#)
- [Validation Rules](#)
- [Classification Groups](#)
- [Annotations](#)
- [SWIFT Field Data Types](#)

Simple Data Types

Simple types are the building blocks of ADS. They define the format and validity of the individual data items that comprise the data structure.

The three main simple data type are as follows:

- [Atomic Simple Types](#)
- [List Simple Types](#)
- [Union Simple Types](#)

Atomic Simple Types

Atomic simple types are derived from other atomic simple types and built in types.


All atomic simple types have a base type that they restrict. You can set the base type via the Base Type panel when creating the atomic type. Alternatively, you can set the base type by clicking on the atomic type in the Explorer window and updating the Base of Atomic Simple Type property in the Properties window.

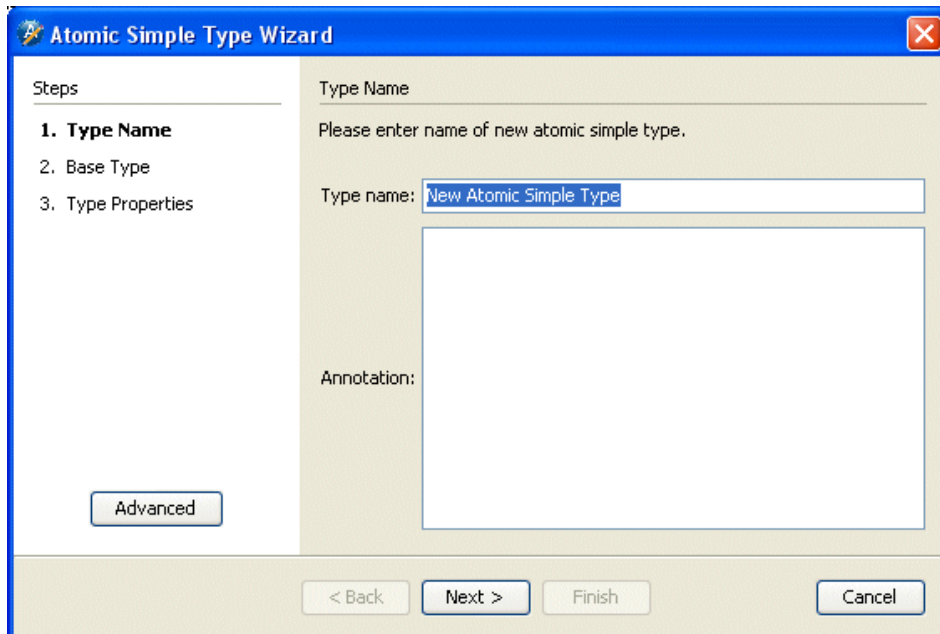
The way in which an atomic simple type restricts its base type varies, depending on what the base type is. For example, when deriving from a string type, you can restrict the string's length; when deriving from a double type, you can restrict its minimum and maximum values.

Note: You cannot create atomic simple types that extend their base type.

Adding an Atomic Simple Type

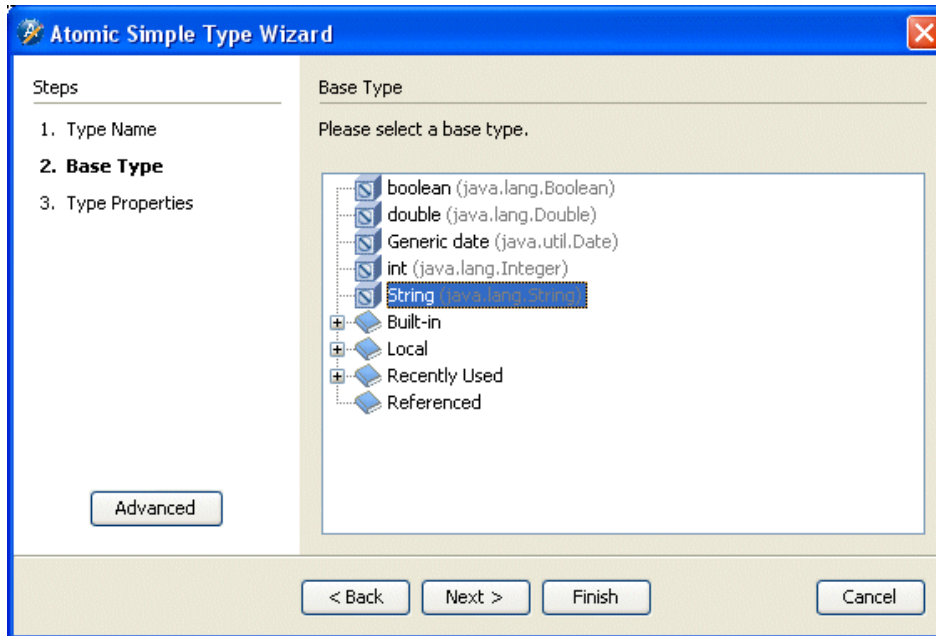
To add an atomic simple type:

1. Do one of the following:
 - Right-click the data model (.dod) file in the Explorer window and select **New > Atomic Simple Type**.
 - Click the data model (.dod) file in the Explorer window and click  on the data model palette.
2. In the Atomic Simple Type wizard, enter the name of the new atomic simple type in the **Type name** field.



3. If you wish to assign an annotation to this type, enter it in the relevant text box.

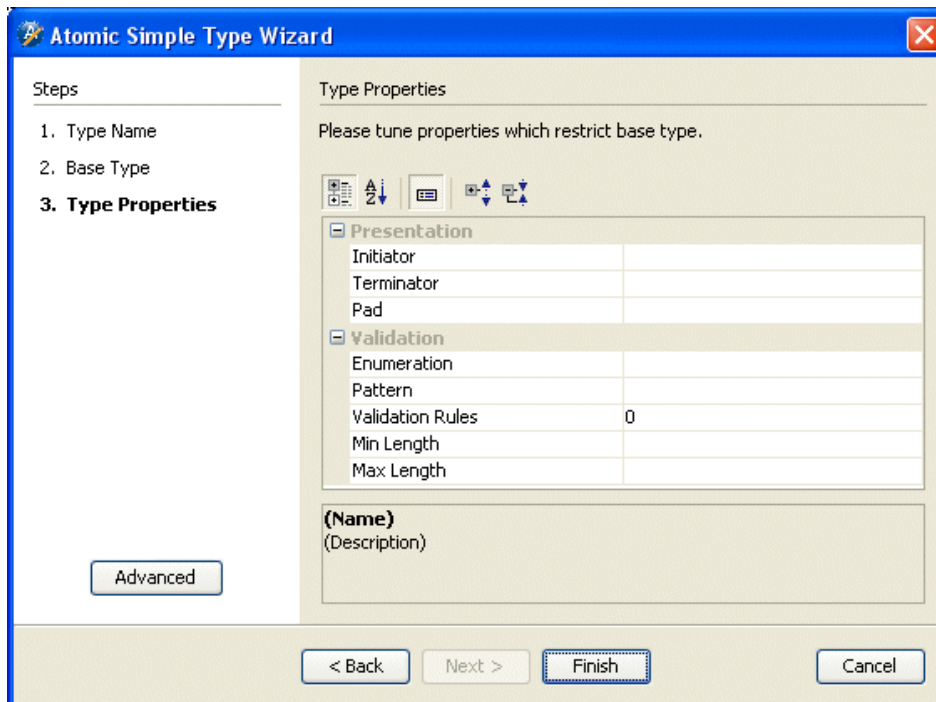
4. Click **Next**. This opens the Base Type panel.



5. Select the type on which your new atomic type is to be based. The types you can select are categorized as "**Built-in**" (for example, boolean, string, or int).

Note: The most common built-in types are listed at the top of the list.

6. Click **Next**. This opens the Type Properties panel.

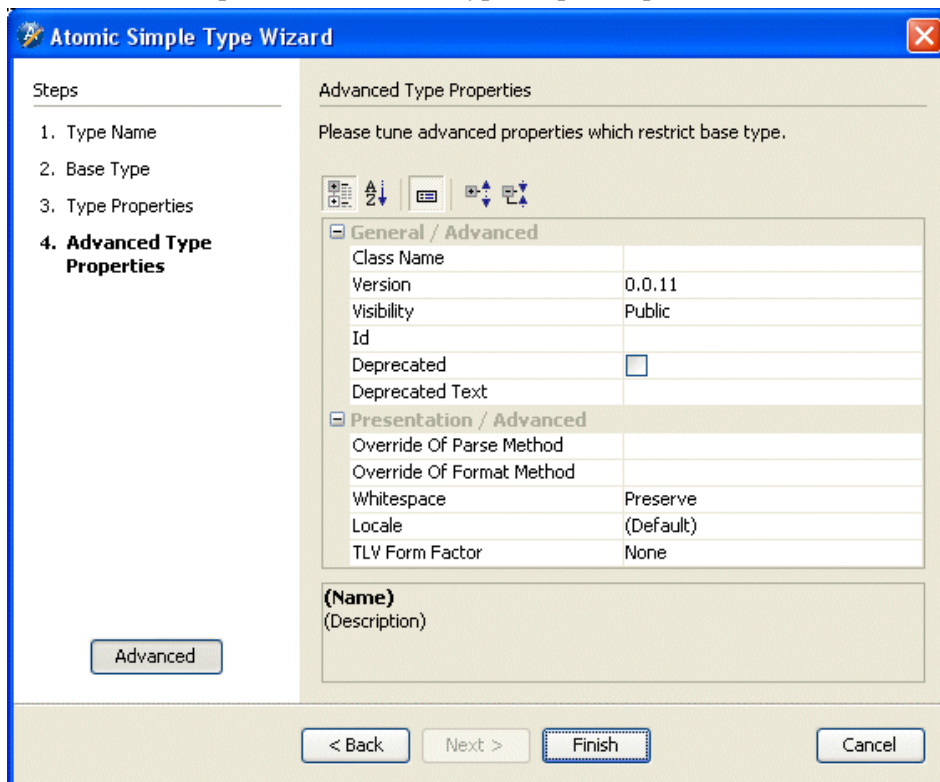


7. Click on a particular property to enter or select a value for that property in its corresponding text field.

Note: The properties displayed on this panel will vary depending on which base type you selected on the previous panel. See [Model and Transformation Properties](#) for more details of the various properties available.

Note: You do not have to set any property values before you finish creating a type. You can always update the properties for that type later, by clicking on the type in the Explorer window and updating the relevant properties in the Properties window.

- If you wish to set more advanced properties for the type you are creating, click the **Advanced** button and then click **Next**. This opens the Advanced Type Properties panel.



- Click on a particular property to enter or select a value for that property in its corresponding text field.
Note: The properties displayed on this panel will vary depending on which base type you selected on the previous panel. See [Model and Transformation Properties](#) for more details of the various properties available.

Note: You do not have to set any property values before you finish creating a type. You can always update the properties for that type later, by clicking on the type in the Explorer window and updating the relevant properties in the Properties window.


- Click **Finish**. The newly created type is added to the Explorer window and its properties are displayed in the Properties window.

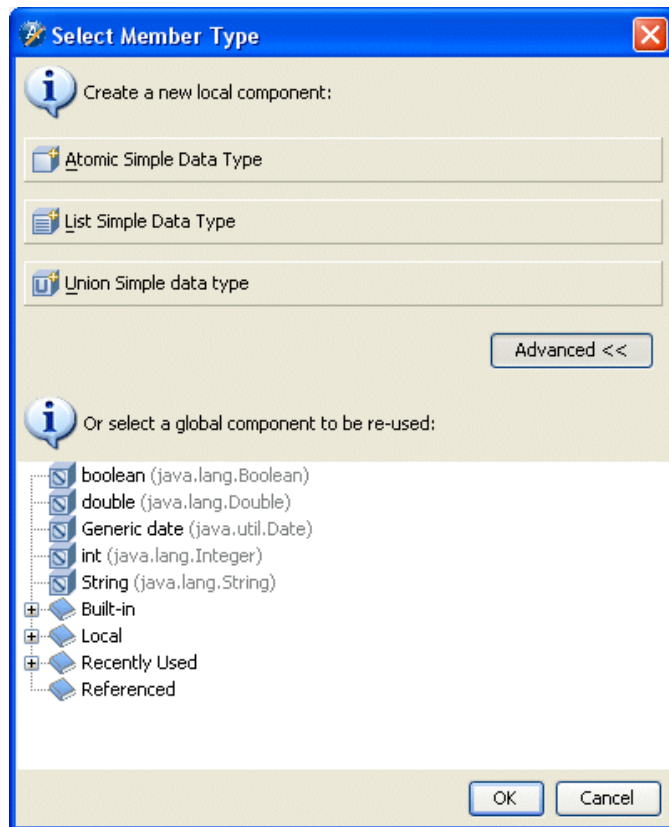
List Simple Types

List simple types have Atomic or Built-in Types which are known as the member type. They store their values as arrays of their underlying base types and then present this as a space-delimited string.

Adding a List Simple Type

To add a list simple type:

1. Open the New List Simple Type dialog in either of the following ways:
 - Right-click the data model (.dod) file in the Explorer window and select **New > List Simple Type**.
 - Click the data model (.dod) file in the Explorer window and click  on the data model palette.
2. Type the name you want to assign to the list simple type and click **OK**. This opens the Select Member Type dialog.



Note: Click the **Advanced** button to display List and Union options in the **Create a new local component** section of the dialog.

3. If you want to create a new member type specifically for this list type, click the relevant option in the **Create a new local component** section and follow the necessary steps to create that simple data type. Alternatively, if you want to select an existing type as the member type, click the relevant option in the 'Or select a global component to be re-used' section.


4. Click **OK**. The newly created list type is added to the Explorer window and its properties are displayed in the Properties window.

Union Simple Types

Union simple types are built up from atomic and built-in types, which are known as the set of member types. The union of the member types' value space specifies the allowable values of the resultant types. This means that the value may be a valid value of any of the member types. For this reason, union types store their values as a `java.lang.Object`.

Adding a Union Simple Type

To add a union simple type:

1. Open the New Union Simple Type dialog in either of the following ways:
 - Right-click the data model (.dod) file in the Explorer window and select **New > Union Simple Type**.
 - Click the data model (.dod) file in the Explorer window and click  on the toolbar.
2. Type the name you want to assign to the union simple type and click **OK**. The new component is then added to the Explorer window.

Built-in Data Types

Every data model has access to the built-in types defined below:

Type Name	Derived From	Description	Examples
Text			
String		Text containing any sequence of characters. The most general simple type. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#string)	ADS
Normalized String	String	String with carriage returns and tabs replaced by spaces. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#normalizedString)	ADS
Token	Normalized String	Normalized String with line feeds, tabs, leading and trailing whitespace, and sequences of two or more spaces removed. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#token)	ADS
Language	Token	A valid RFC1766: Tags for the Identification of Languages code. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#language)	en-GB, en-US, fr
NMTOKEN	Token	Legal XML 1.0 name tokens. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#NMTOKEN)	US, Brésil
Name	Token	Legal XML 1.0 names. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#token)	shipTo
NCName	Name	Non-colonised name, i.e. a legal XML 1.0 name without a colon. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#Name)	UKAddress
ID	NCName	Unique NCName among all other values of type ID. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#NCName)	
IDREF	NCName	NCName whose value is used elsewhere with a type of ID. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#IDREF)	
ENTITY	NCName	NCName whose value is declared as an unparsed entity in the documents DTD. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#ENTITY)	

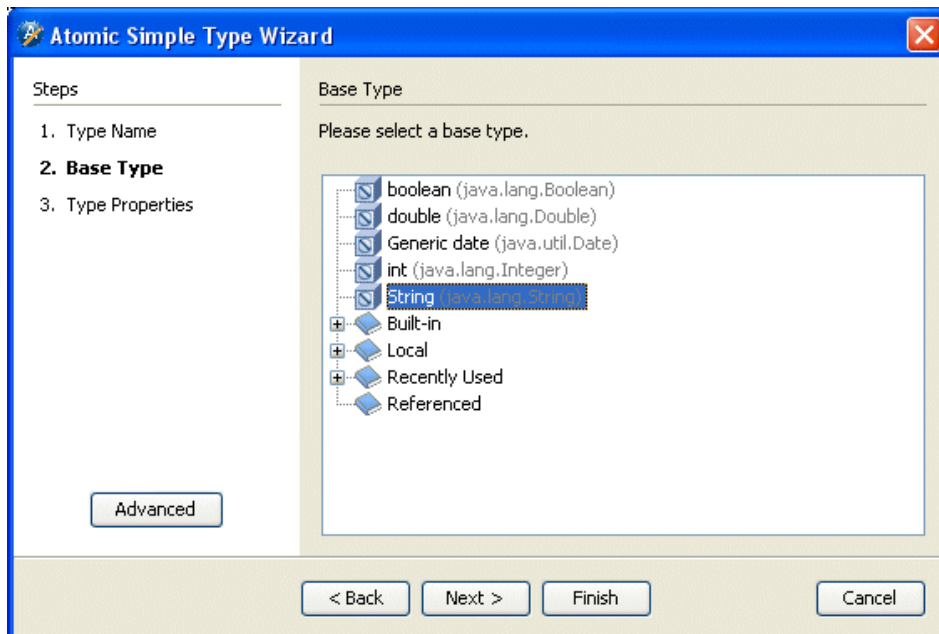
Any URI	String	Any valid URI (and therefore URL or URN as well). anyURI)	http://www.progres
QName	String	Namespace qualified name. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#QName)	po:UKAddress
NOTATION	String	Qualified name declared as a notation. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#NOTATION)	
SQL CLOB		A SQL Character Large Object	
Numeric			
double		Eight-byte binary floating point numbers in IEEE754 format. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#double)	-INF, -1E4, -0, 0, 1, INF, NaN
float	double	Four-byte binary floating point numbers in IEEE754 format. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#float)	-INF, -1E4, -0, 0, 1, INF, NaN
decimal		Base 10 number with any finite number of the digits 0-9 before and after the decimal point. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#decimal)	-1.23, 0, 123.4, 100
integer	decimal	Base 10 number with any finite number of the digits 0-9 before the decimal point. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#integer)	-126789, -1, 0, 1, 12
long	integer	Signed integer represented as an eight-byte two's compliment number. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#long)	-1, 1267896754323
int	long	Signed integer represented as a four-byte two's compliment number. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#int)	-1, 126789675
short	int	Signed integer represented as a two-byte two's compliment number. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#short)	-1, 12678
byte	short	Signed integer represented as a one-byte two's compliment number. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#byte)	-1, 126
char	int	16 bit Unicode character.	a, Z, &, @
non positive integer	integer	Integer less than or equal to 0. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#nonPositiveInteger)	-126789, -1, 0

non negative integer	integer	Integer greater than or equal to 0. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#nonNegativeInteger)	0, 1, 126789
positive integer	integer	Integer greater than 0. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#positiveInteger)	1, 126789
negative integer	integer	Integer less than 0. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#negativeInteger)	-126789, -1
unsigned long	non negative integer	Non negative integer that can be stored in 8-bytes. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#unsignedLong)	0, 12678967543233
unsigned int	unsigned long	Non negative integer that can be stored in 4-bytes. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#unsignedInt)	0, 1267896754
unsigned short	unsigned int	Non negative integer that can be stored in 2-bytes. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#unsignedShort)	0, 12678
unsigned byte	unsigned short	Non negative integer that can be stored in 1-byte. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#unsignedByte)	0, 126
Date & Time			
Generic Date		Generic date / time with user defined format.	23 July 1979, 15:50
ISO8601 Duration		A length of time measured in years, months, days, hours, minutes, seconds and fractions of a second. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#duration)	P1Y2M3DT10H30
ISO8601 Date Time		A specific moment in history measured in years, months, days, hours, minutes, seconds and fractions of a second. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#dateTime)	1999-05-31T13:20:
ISO8601 Date		A specific day in history measured in years, months and days. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#date)	1999-05-31

ISO8601 Time		A specific time on no particular day measured in minutes, seconds and fractions of a second. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#time)	13:20:00.000, 13:20:00.000-05:00
ISO8601 gDay		A specific day of no particular month in the Gregorian calendar. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#gDay)	---31
ISO8601 gMonth		A specific month of no particular year in the Gregorian calendar. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#gMonth)	--05-31
ISO8601 gYear		A specific year in the Gregorian calendar. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#gYear)	1999
ISO8601 gMonthDay		A specific day of a specific month in the Gregorian calendar. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#gMonthDay)	--05--
ISO8601 gYearMonth		A specific month of a specific year in the Gregorian calendar. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#gYearMonth)	1999-02
SQL date		A SQL formatted date.	
SQL time		A SQL formatted time.	
SQL timestamp		A SQL formatted timestamp.	
Boolean			
boolean		A logical boolean value with configurable values for true and false. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#boolean)	true, false, yes, no,
Binary			
Hex Encoded Binary		An arbitrary sequence of bytes that has been encoded by replacing each byte with two hexadecimal digits from 0 through 9 and A through F. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#hexBinary)	0FB7

Base 64 Encoded Binary		An arbitrary sequence of bytes that has been encoded in ASCII character using the algorithm defined in RFC2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#base64Binary)	GpM7
SQL BLOB		A SQL Binary Large Object	
List			
IDREFS	IDREF	LIST of type IDREF. IDREFS)	
ENTITIES	ENTITY	LIST of type ENTITY. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#ENTITIES)	
NMTOKENS	NMTOKEN	LIST of type NMTOKEN. (http://www.w3.org/tr/2001/rec-xmlschema-2-20010502#NMTOKENS)	US UK, Brésil, Can

Every time you create a new [atomic simple type](#), you need to specify a base type, which can either be another atomic simple type or one of the built-in data types listed above. The following screen shows the dialog where you choose between using another atomic data type or a primitive data type.



The potential base types are categorized as follows:

Built-in

Any of the types listed in the table above.

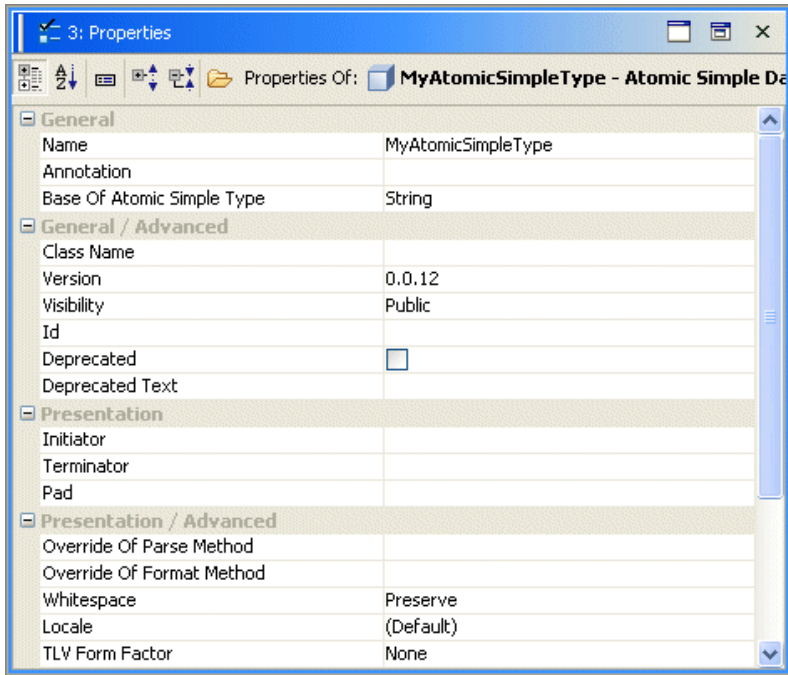
Local

Any atomic simple type already defined in the current data model.

Referenced

Any atomic simple type defined in a data model that is referenced or imported in the current data model.

Note: After you have defined the base type for a new atomic data type, you can change the base type through the Properties window for the selected atomic simple type.



Click the value of the Base of Atomic Simple Type to open a Select Component dialog where you can select a new base type.

Complex Data Types

Complex data types collect together a set of [elements](#) and [attributes](#) to form a reusable object. Each element and attribute has a type associated with it and this therefore leads to a hierarchical (and possibly recursive) data structure.

Whether you define an object as a simple or complex type depends on the granularity with which you need to access the data. An address, for example, could be represented as a simple string type if it will only ever be treated as an atomic object. If, however, you need access to the postal (zip) code or you need to know the building number or name, you should consider making the address type complex, thereby using the power of ADS to parse the parts of the address needed.


See <http://www.w3.org/TR/xmlschema-0/#DefnDeclar> for details.

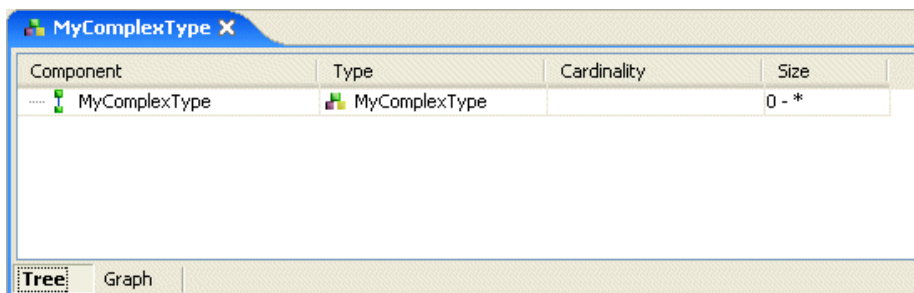
To use a complex type, you must decide what content model your data adheres to:

- Sequence, where the order of child elements contained within the type matters. All new complex types are created by default as sequences.
- Choice, where one and only one of the specified possible child elements can exist in any one instance of the complex type.
- All, where the order of the list of child types does not matter.

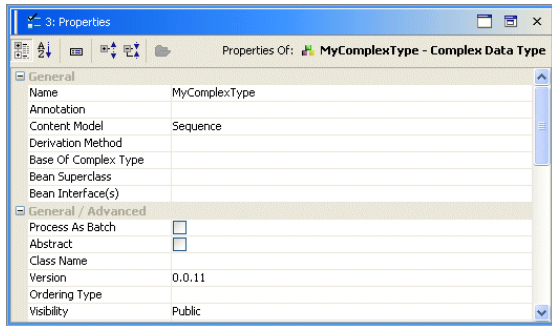
Adding a Complex Type

To add a complex type to a data model:

1. Do one of the following:
 - Right-click the data model (.dod) file in the Explorer window and select **New > Complex Type**.
 - Click the data model (.dod) file in the Explorer window and click  on the data model palette.
2. In the New Complex Type dialog, type the name you want to assign to the complex type and click **OK**. The new component is added to the Explorer window. A new tab is also opened for the complex type in the main window.



The properties of the type (of which the content model is one of the more important) are displayed in the Properties window.



3. Add other components to the complex type by using the toolbar, by dragging components from the Explorer window, or by right-clicking on a component in the main window and using the resultant context menu.

Data Components

Data Components is the generic term used for:

- [Elements](#)
- [Element Groups](#)
- [Any Elements](#)
- [Attributes](#)
- [Attribute Groups](#)
- [Any Attributes](#)

Elements


Elements represent the application of a type in order to build up a hierarchical data structure. This most commonly occurs in the structure of a [complex type](#) where a local element is declared with either a type or a reference to a global element. As well as their use as references, global elements signify a possible root of a data structure.

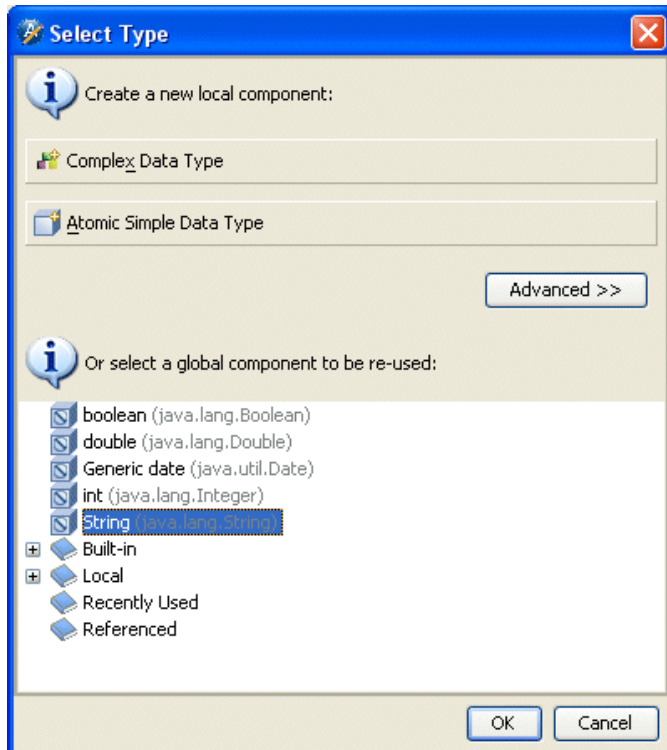
In strict XML terms, an instance document adheres to a specific global element (which then defines its type). In the ADS API, deployed objects can be created directly from complex types, in which case an element declaration will be created at runtime.

See <http://www.w3.org/tr/xmlschema-0#globals> for details.

Adding an Element

To add an element:

1. Open the New Element dialog in either of the following ways:
 - Right-click the data model (.dod) file in the Explorer window and select **New > Element**.
 - Click the data model (.dod) file in the Explorer window and click  on the toolbar.
2. Type the name you want to assign to the element and click **OK**. This opens the Select Type dialog.



Note: Click the **Advanced** button to display Swift, List and Union options in the 'Create a new local component' section of the dialog.

3. If you want to create a new type specifically for this element, click the relevant option in the 'Create a new local component' section and follow the necessary steps to create that type. Alternatively, if you want to select an existing type, click the relevant option in the 'Or select a global component to be re-used' section.
4. Click **OK**. The newly created element is added to the Explorer window and its properties are displayed in the Properties window.

Creating Elements from Types

ADS Designer allows you to quickly create a global element from an existing type. The resulting element has the same name and is of the same type as the source type.

To create an element from a type:

1. In the Explorer window, right-click a complex type or simple type and select **New > Element from Type**.
2. In the New Element from Type dialog, the name of the source type appears with a number appended. You can name the new element the same as the originating type, provided that an element of that name does not already exist.

Element Groups


Element groups encapsulate a set of [elements](#) that can be used by many different complex types.

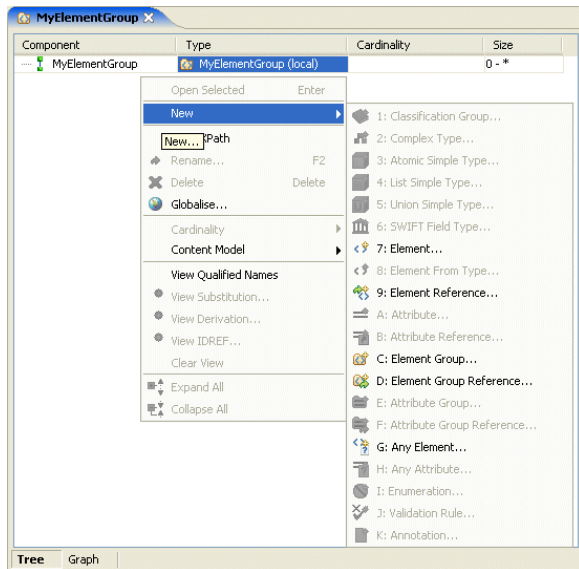
If you find you are constantly reusing the same collection of elements over and over again, you should consider embedding them in an element group and using this instead. This has the added advantage that when you decide to add or remove an element from the group, the change will affect all complex types that use the group.

See <http://www.w3.org/tr/xmlschema-0#groups> for details.

Adding an Element Group

To add an element group:

1. Open the New Element Group dialog in either of the following ways:
 - Right-click the data model (.dod) file in the Explorer window and select **New > Element Group**.
 - Click the data model (.dod) file in the Explorer window and click  on the data model palette.
2. Type the name you want to assign to the element group and click **OK**. The new component is added to the Explorer window and a new tab is opened for the element group in the main window.
3. Add elements to the element group by using the toolbar, by dragging elements from the Explorer window, or by right-clicking in the main window and selecting New Element.
When you double-click an element group in the Explorer window, it is then opened in its own tab for editing:



New elements, element references, element groups, and element group references can be added in any of the following ways (make sure you have clicked in the Type column in the element group tab first):

- Select **File > New Component**.
- Click the relevant data model palette icon.
- Right-click in the Type column and select **New Component**.

Any Elements


An any element allows any element from a particular namespace to appear.

See <http://www.w3.org/TR/xmlschema-1/#element-any> for details.

Adding an Any Element

You can add an 'any' element to a complex type or element group that is opened in the main window.

To add an 'any' element, do either of the following:

- Right-click the component in the main window and select **New > Any Element**.
- Click the component in the main window and select  on the data model palette.

The any element is added to the main window.

Attributes

Attributes allow instance documents to provide additional data about an object. They are commonly used for 'supporting' information, rather than the core data itself which should be embedded in [elements](#). Because attributes are essentially tag value pairs, they are limited to [simple types](#).


Attributes can be defined globally or locally.

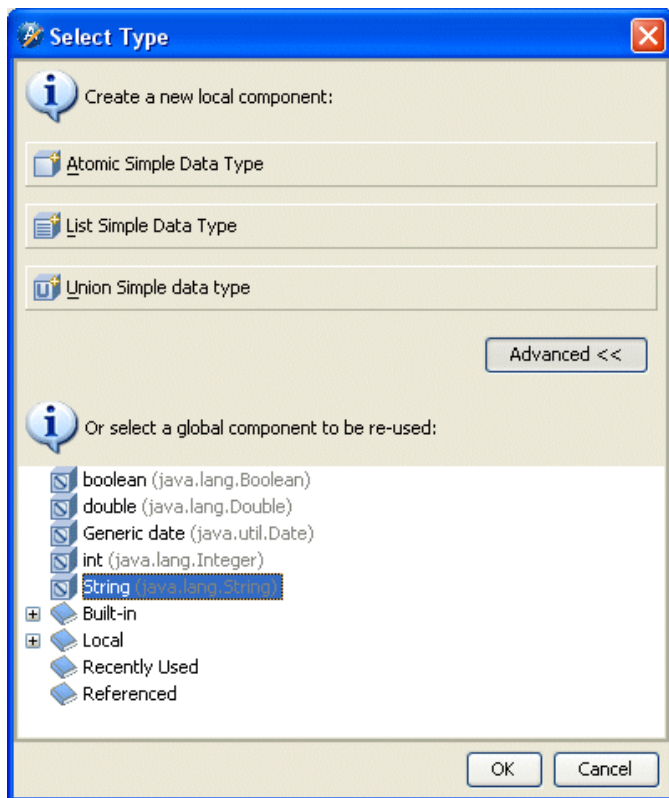
A globally defined attribute must specify the type of its data and cannot contain a reference, whereas a locally defined attribute can reference a global attribute or define its type directly.

See <http://www.w3.org/TR/xmlschema-0/#DefnDeclars> for details

Adding an Attribute

To add an attribute:

1. Open the New Attribute dialog in either of the following ways:
 - Right-click the data model (.dod) file in the Explorer window and select **New > Attribute**.
 - Click the data model (.dod) file in the Explorer window and click  on the data model palette.
2. Type the name you want to assign to the attribute and click **OK**. This opens the Select Type dialog.



Note: Click the **Advanced** button to display List and Union options in the 'Create a new local component' section of the dialog.

3. To create a new type specifically for this attribute, click the relevant option in the 'Create a new local component' section and follow the necessary steps to create that type.
Alternatively, to select an existing type, click the relevant option in the 'Or select a global component to be re-used' section.
4. Click **OK**. The newly created attribute is added to the Explorer window and its properties are displayed in the Properties window.

Attribute Groups

Attribute groups encapsulate a set of [attributes](#) that can be used by many different complex types.


If you find you are constantly reusing the same collection of attributes, you should consider embedding them in an attribute group and using this instead. This means that when you decide to add or remove an attribute from the group, the change will affect all complex types that use the group.

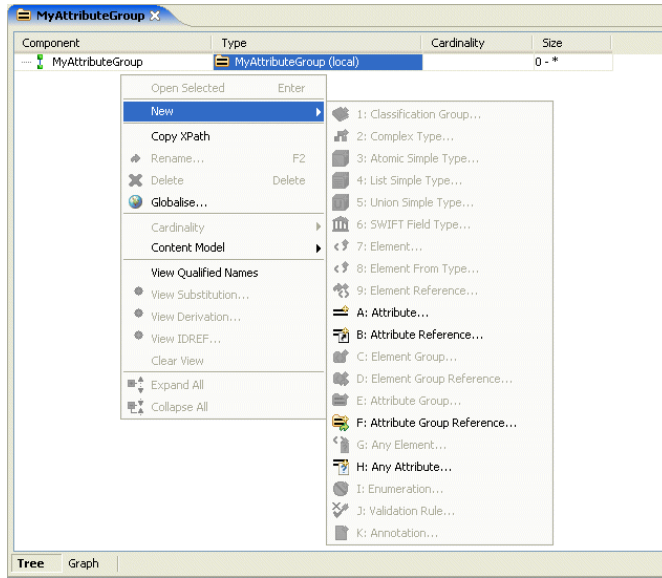
Note: When an attribute group is generated into code it is flattened out, so that the complex type that uses it contains the attributes in the group directly.

See <http://www.w3.org/tr/xmlschema-0#attrgroups> for details.

Adding an Attribute Group

To add an attribute group:

1. Open the New Attribute Group dialog in either of the following ways:
 - Right-click the data model (.dod) file in the Explorer window and select **New > Attribute Group**.
 - Click the data model (.dod) file in the Explorer window and click  on the data model palette.
2. Type the name you want to assign to the attribute group and click **OK**. The new component is added to the Explorer window and a new tab opens for the attribute group in the main window.
3. Right-click or click the relevant data model palette icon to add attributes, attribute references, attribute group references, or any attributes to the attribute group.




Any Attributes

An any attribute allows any attribute from a particular namespace to appear. You can add an any attribute to a complex type or attribute group that is opened in the main window.

See <http://www.w3.org/tr/xmlschema-0#ref32> for details.

Adding an Any Attribute

To add an any attribute, do either of the following:

- Right-click the component in the main window and select **New > Any Attribute**.
- Click the component in the main window and select  on the data model palette.

The any attribute is added to the main window.


References

You can declare the following components globally and then reference them in a data model:

- [Element](#)
- [Attribute](#)
- [Element Group](#)
- [Attribute Group](#)


Adding an Element Reference

You can add an element reference to a complex type that is opened in the main window. To add an element reference:

1. Open the Select Referenced Element dialog in either of the following ways:
 - Right-click the data model (.dod) file in the Explorer window and select **New > Element Reference**.
 - Click the data model (.dod) file in the Explorer window and click  on the toolbar.
2. Select the element you wish to reference and click **OK**. The element reference is then added to the main window.


Adding an Attribute Reference

You can add an attribute reference to a complex type that is opened in the main window. To add an attribute reference:

1. Open the Select Referenced Attribute dialog in either of the following ways:
 - Right-click the data model (.dod) file in the Explorer window and select **New > Attribute Reference**.
 - Click the data model (.dod) file in the Explorer window and click  on the data model palette.
2. Select the attribute you wish to reference and click **OK**. The attribute reference is then added to the main window.

Adding an Element Group Reference


You can add an element group reference to a complex type that is opened in the main window. To add an element group reference:

1. Open the Select Referenced Element Group dialog in either of the following ways:
 - Right-click the data model (.dod) file in the Explorer window and select **New > Element Group Reference**.
 - Click the data model (.dod) file in the Explorer window and click  on the data model palette.

2. Select the element group you wish to reference and click **OK**. The element group reference is then added to the main window.

Adding an Attribute Group Reference

You can add an attribute group reference to a complex type that is opened in the main window. To add an attribute group reference:

1. Open the Select Referenced Attribute Group dialog in either of the following ways:
 - Right-click the data model (.dod) file in the Explorer window and select **New > Attribute Group Reference**.
 - Click the data model (.dod) file in the Explorer window and click  on the data model palette.
2. Select the attribute group you wish to reference and click **OK**. The attribute group reference is then added to the main window.

Validation Components

Validation Components is the generic term used for:

- [Enumerations](#)
- [Validation Rules](#)

Enumerations

Enumerations allow you to restrict the allowable values of a type.

They can be defined either globally or locally. A globally defined enumeration can be reused in several places. An enumeration is essentially a list of allowable text values for the object. If none of these are found to match the object's value, the object is considered invalid.

See <http://www.w3.org/tr/xmlschema-0#ref10> for details.

Note: When an enumeration is applied to a complex type without content, the ADS API tries to match the output derived from formatting the type with a DefaultSink, but without any initiator or terminator applied to the complex type. (The complex type's delimiter, and the full formatting including initiators and terminators of any contained simple or complex types, will be used). When an enumeration is applied to a complex type with content, the output from formatting the object's content will be matched.




Enumerations can be one of two types:

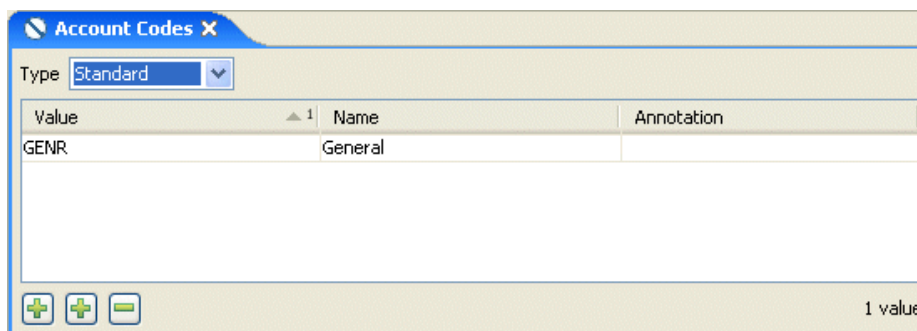
- Standard
- [Factory Lookup](#)





Changing the type of an enumeration in Designer clears all previous settings.

Adding a Global Enumeration

To add a global enumeration to a data model:

1. Open the New Enumeration dialog in either of the following ways:
 - Right-click the data model (.dod) file in the Explorer window and select **New > Enumeration**.
 - Click the data model (.dod) file in the Explorer window and click  on the data model palette.
2. Type the name you want to assign to the enumeration and click **OK**. The new enumeration is added to the root of the data model in the Explorer window. A new tab is also opened for the enumeration in the main window.
3. In the new enumeration tab, select an enumeration type of either **standard** or **factory lookup**.
4. If you selected a standard enumeration type, click the  and  icons to add or remove literal values for the enumeration. You can click on a particular literal value in the list to add a name and descriptive annotation for it.



If you selected a [factory lookup](#) enumeration, click the  and  icons to add or remove classes for the enumeration. Click the  and  icons to move classes up or down the list.

Adding a Local enumeration

You can add a local enumeration to a particular type, by setting the Enumeration property in the [Validation](#) section of the Properties window.

Factory Lookups

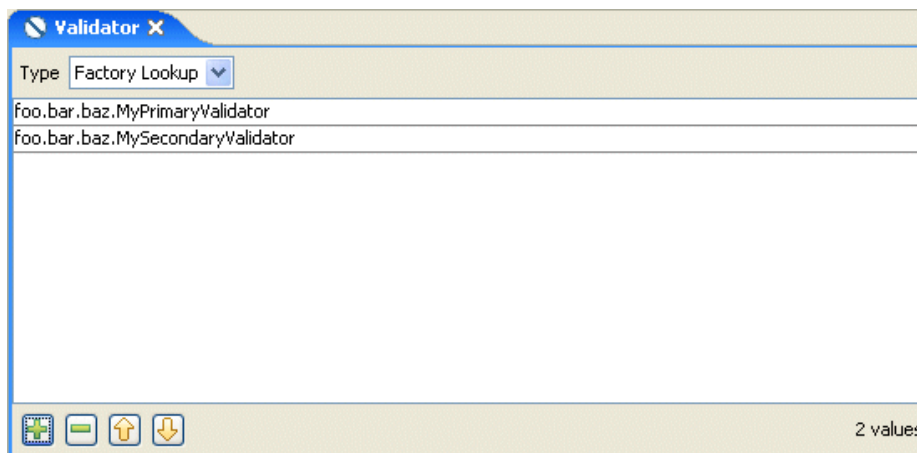
Factory lookups allow you to externalize types validation, by specifying a series of class names that ADS will in turn attempt to instantiate





The first valid class it finds that implements `biz.c24.io.api.data.Validator` will be used for validation. If none are found, ADS uses the `biz.c24.io.api.data.SimpleValidator` which will pass or fail all instances depending on the value of a system property.

Factory lookups can be applied to [enumerations](#) and validation rules.

Factory lookups provide a powerful validation mechanism if used correctly. For example, they can query a database, Web service or even another deployed data object. They can also be used to enforce locale or date-specific validation rules.

In the example below, `foo.bar.baz.MyPrimaryValidator` will be queried first and if it can be found the validation will fall back to `foo.bar.baz.MySecondaryValidator`.



Click the  and  icons to add or remove classes for the enumeration. Click the  and  icons to move classes up or down the list.

Validation Rules

Validation rules allow you to add validation to a data type. Rules can be expressed using one of the following set of mechanisms:


- [XPath](#)
- [XQuery](#)
- [Java Code](#)
- [Database Lookup](#)
- [Contextual](#)
- [Domain Constraint](#)
- [Factory Lookup](#)

You can use only one of these formats at any one time. Changing the type of validation rule causes a dialog to appear warning you that all current settings will be cleared.

Rules can be defined either globally or locally, A global rule can be reused in several places.

Adding a Global Validation Rule

To add a global validation rule to a data model:

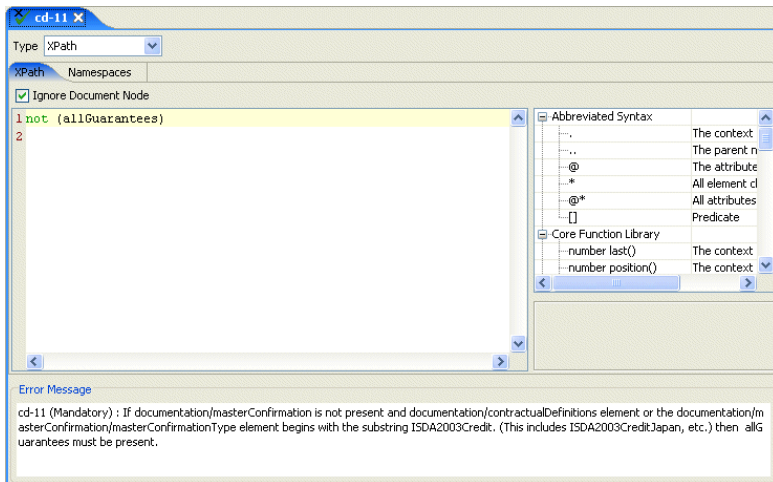
1. Open the New Validation Rule dialog in either of the following ways:
 - Right-click the data model (.dod) file in the Explorer window and select **New > Validation Rule**.
 - Click the data model (.dod) file in the Explorer window and click  on the data model palette.
2. Type the name you want to assign to the validation rule and click **OK**. The new validation rule is added to the Explorer window. A new tab also opens for the validation rule in the main window.
3. In the new validation rule tab, select the type of validation rule you want to set up ([XPath](#), [XQuery](#), [Java Code](#), [Database Lookup](#), [Contextual](#), [Domain Constraint](#), or [Factory Lookup](#))

Adding a Local Validation Rule

You can add a local validation rule to a particular type, by setting the Validation Rules property in the [Validation](#) section of the Properties window.

XPath

An XPath validation rule requires you to enter an XPath expression that must evaluate to true for validation to fail and an error message to be used in failure cases.




The XPath editor includes the following tabs:

XPath

Enter your XPath expression here. You can quickly add core XPath functions and operators by double-clicking them in the tree on the right. Select the **Ignore Document Node** checkbox if you want to ignore the document node in the XPath expression. If this property is selected, the XPath expression should not include the name of the component whose type has this rule applied to it. This is the preferred approach since the type to which this rule is applied might be used by multiple components.

Namespaces

If you select the **Namespace Aware** checkbox, the XPath expression must use the prefixes defined in the namespaces table below. To add a namespace, click the  icon.

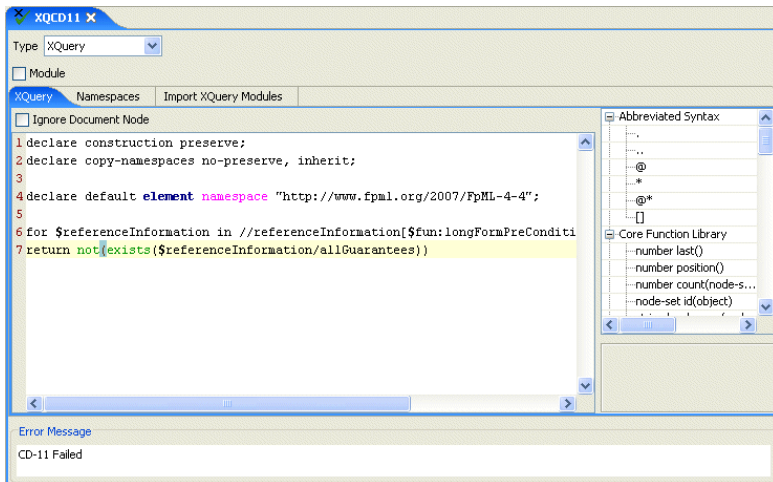
XQuery

An XQuery validation rule requires you to enter an XQuery expression that must evaluate to true for validation to fail and an error message to be used in failure cases.

XQuery validation rules support:

- FOR, LET, WHERE, ORDER BY, RETURN (FLWOR) expressions
- Value comparison operators
- Built-in functions
- Imported XQuery modules

Note: ADS validation rules do not support schema-aware XQuery.




The XQuery editor includes the following tabs:


XQuery

Enter your XQuery here. You can quickly add core XQuery functions and operators by double-clicking them in the tree on the right. Select the **Ignore Document Node** checkbox if you want to ignore the document node in the XQuery. If this property is selected, the XQuery should not include the name of the component whose type has this rule applied to it. This is the preferred approach since the type to which this rule is applied might be used by multiple components.

Namespaces

If you select the **Namespace Aware** checkbox, the XQuery must use the prefixes defined in the namespaces table below. To add a namespace, click the  icon.

Import XQuery Modules

Select the **Modules** checkbox if the query you are creating is a module—a fragment of XQuery code that conforms to the Module grammar. Click the  to specify a prefix and URI for the module namespace.

Java Code

The Java Code type of validation rule requires you to use the method signature laid out below to return a boolean value, where true implies the object is valid:

```

cd-44 X
Type Java Code
public boolean validate(java.lang.Object object,
biz.c24.io.api.data.DataComponent component,
biz.c24.io.api.data.ComplexDataObject context,
com.iona.artix.ds.api.data.ValidationManager manager)
{
// START OF METHOD
1 biz.c24.io.api.data.ComplexDataObject referencePool = (biz.c24.io.api.data
2
3     String childName = null;
4     for (int i = 0; i < referencePool.getElementCount("referenceP
5     {
6         biz.c24.io.api.data.ComplexDataObject item = (biz.c24.io.e
7         String newChildName = null;
8         if (item.getElementCount("constituentWeight") > 0)
9         {
10            biz.c24.io.api.data.ComplexDataObject constituentWeigh
11
12            for (int j = 0; j < constituentWeight.getElementDeclC
13            {
14                biz.c24.io.api.data.Element decl = constituentWeig
15                if (constituentWeight.getChildCount(decl) > 0)
// END OF METHOD
}

```

Result

PASSED	IF
FAILED [msg]	R
FAILED [msg]	R
FAILED [msg]	R
FAILED [msg]	R

Cast

biz.c24.io.api.data.I...	C
java.lang.Boolean	C
java.lang.Long	C
java.sql.Timestamp	C
java.lang.Float	C
biz.c24.io.api.data.C...	C
biz.c24.io.api.data.I...	C
java.math.BigInteger	C
java.lang.Double	C
java.sql.Clob	C
biz.c24.io.api.data.I...	C
java.lang.Character	C

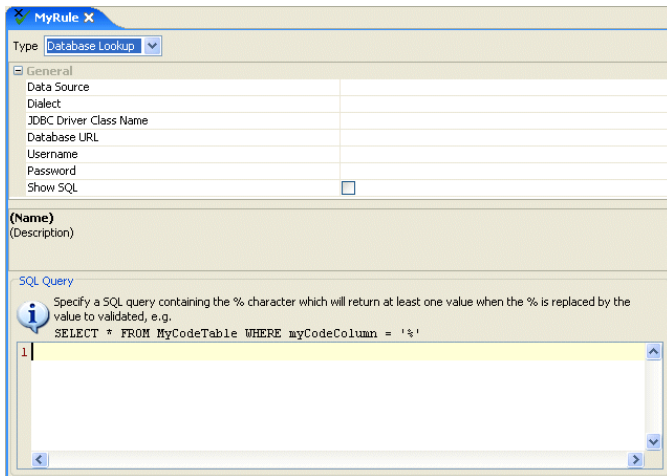
In addition to returning the validity of the object, you are also required to call:

```
context.validationEventOccurred(biz.c24.io.api.ValidationResultEnum.FAILED_RULE_ERROR, "Some error message");
```

Where the first argument can be any type of validation result (although FAILED_RULE_ERROR is the most common) and the second argument is the reason.

Database Lookup

The Database Lookup type of validation rule requires you to specify an SQL query which will return at least one value after validating the relevant input parameter.



You can set the following database properties:

Data Source

Specifies a name that is used to identify the connection being used (for example, a JNDI name).

Dialect

Specifies the Hibernate dialect that is to be used to communicate with the database.

JDBC Driver Class Name

Specifies the class name of the JDBC database driver.

Database URL

Specifies the URL of the target database.

Username

Specifies a valid username that is to be used to connect to the database.

Password

Specifies a valid password that is to be used to connect to the database.

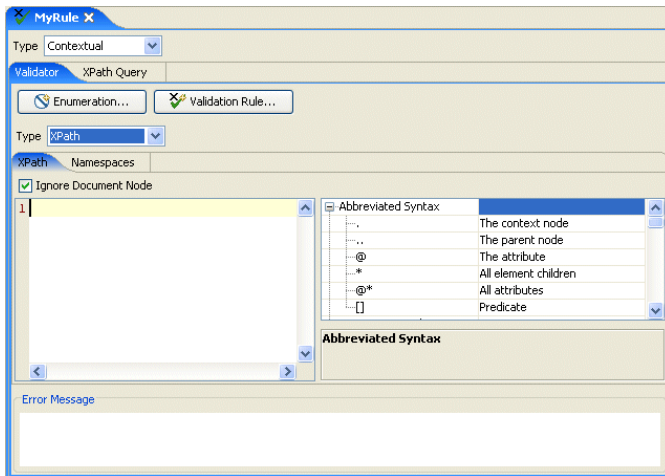
Show SQL

Indicates whether SQL statements are to be printed to the console.

Type the relevant SQL query in the available text box

Contextual

The Contextual type of validation rule should be used when you only want to validate depending on the value of a particular field. For example, in the case of `<person gender="male">` you could have a validation rule that will only be executed if the gender attribute is set to "male".



Click the **Enumeration** button to set up the allowable values. See [Enumerations](#) for more details.

Click the **Validation Rule** button to set up the appropriate validation rule (for example, an XPath statement).

Domain Constraint

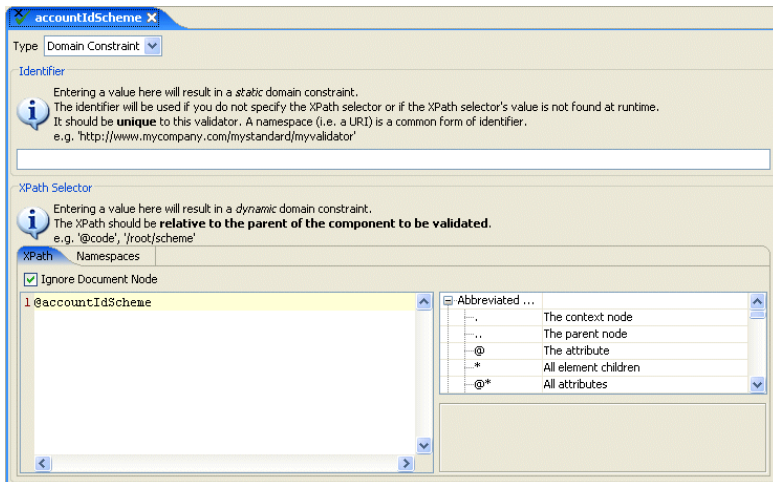
Domain constraints provide users with a way of specifying a list of possible values that an element can contain. This is very similar to the concept of [enumerations](#) but with some important differences:

- Enumerations are specified locally, within a data model.
- Domain Constraints reference lists of externally defined valid values, outside the ADS data model and thus outside the deployed code base.

Setting a validation rule to be of type domain constraint means that the validation rule can check that an element conforms to:

- Static domain constraints - the values against which an element will be checked are held outside the data model, in an XML file or a database, XPath is used to retrieve the list of possible valid values.
- Dynamic domain constraints - the values against which an element will be checked are dependent upon the origin of the instance document, for example two organizations might use the same XML schema to define a - unique identifier - relationship between unique identifier and a registered validator - look for the value of the unique identifier (often an attribute of an element in an XML scenario) and use that to pull out the required set of valid values

You can set both dynamic and static domain constraints simultaneously - at runtime the compiled ADS code will try to use the value of the dynamic domain constraint to validate a field and, if its XPath returns no such element, it will search for the static constraint and use that instead.



You can set the following options when expressing validation rules using domain constraints:

- **Ignore Document Node** - an XPath-specific property, specifies whether to ignore the document node in the XPath expression. If this property is selected, the XPath expression should not include the name of the component whose type has this rule applied to it. This is the preferred approach because the type to which this rule is applied might be used by multiple components.
- **Namespace Aware** - Specifies whether to take account of namespaces in the XPath expression. If you select this option in the Namespaces tab, the XPath expression must use the prefixes defined in the namespaces table below. If this property is not selected, the resulting XPath behaves as if elementFormDefault and f were both set to 'unqualified'.

Cardinality

The mandatory, repetitive or optional nature of the constituent parts of a data structure is defined by their cardinality.

Elements

You can pick one of the following six values:

- 0..1 - Optional - at most one
- 1 - Mandatory - exactly one
- 0..* - Optional and Repetitive - any number
- 1..* - Mandatory and Repetitive - any number, but at least one.
- n - Present an exact number of times which you will be prompted for
- m..n - Present between m and n times which you will be prompted for

The cardinality can be set by right-clicking the required element in the 'elements' tree of a complex type.

Attributes

You can pick one of the following three values:

- Optional - at most one
- Required - exactly one
- Prohibited - may NOT appear

The cardinality can be set via the "Use" column in the 'attributes' table of a complex type.

Includes, Imports, and Redefines

Imports, includes and redefines provide three ways to build up complex data models from small, self-contained subsets of the whole. All three provide a mechanism by which [model components](#) can be defined in one model and used by one or more others.

Includes

Includes are the simplest way to make cross-model references. However, the target namespace of the included model must be the same as the target namespace of the including model. By including a model you are effectively adding its components to the existing target namespace.

See <http://www.w3.org/tr/xmlschema-0#ref23> for more details on includes.

Imports

Imports should be used to reference data models whose target namespace is different from that of the importing model. Imported models can have a prefix assigned to their namespace in order to differentiate their components from those defined by the importing model.

See <http://www.w3.org/tr/xmlschema-0#ref31> for more details on includes.

Redefines

Redefines are very similar to includes except they allow you to redefine types and groups in the redefining schema. From that point on, the redefined type or group is used instead of the original.

See <http://www.w3.org/tr/xmlschema-0#ref31> for more details on redefines.

Namespaces

Concepts such as "Name", "Size", and "Location" are common among many different data models but can have widely different meanings. As models are exchanged and referenced by each other, it becomes important to qualify generic concepts like "Name", so users of the model know whether they are about to use an element which, for example, refers to the name of a person or the name of a country. By associating every model component with a namespace, and allowing models that combine namespaces to assign prefixes (for example, "person" and "country"), you can immediately tell the difference between "person:Name" and "country:Name".

Namespaces could be modeled as anything, so long as everyone regards them as unique. ADS has followed the W3C's recommendation of using Uniform Resource Indicators (URIs) (for example, <http://www.w3.org/TR/xmlschema-0/#NS>). To this end, ADS allows you to specify a URI as the data model's target namespace (that is, the namespace defined by the model). When creating new data models you should get in the habit of setting the target namespace and get to know when models should share a target namespace and when they should define their own.

See <http://www.w3.org/tr/1999/rec-xml-names-19990114/> for more on namespaces.

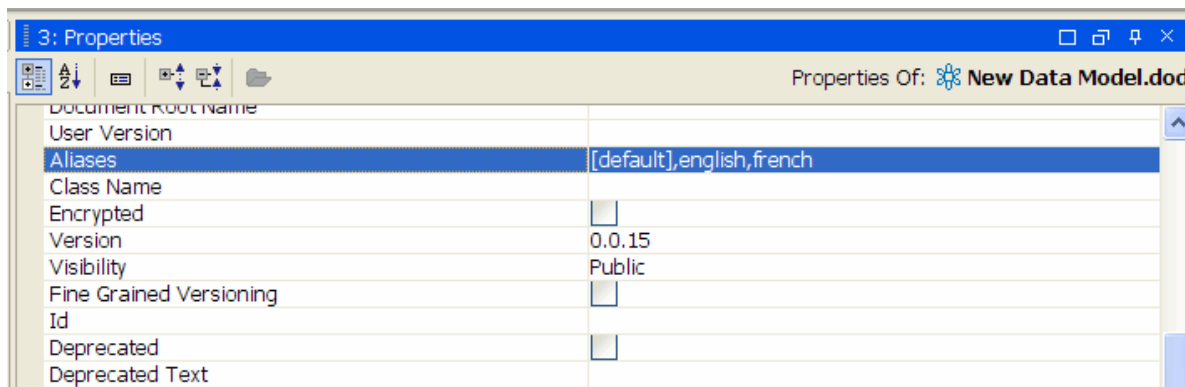
Preferred Aliases

Preferred aliases allow you to use different sets of names for the same components in a data model. This feature is useful, for example, if you want to create 'internationalized' models where different sets of names in different languages are defined for the same components. For example, you might wish to choose between seeing English and French names within the same data model. Using preferred aliases allows you to easily switch from one set of names to another.

Another example for the use of preferred aliases would be if you wanted to choose between seeing different sets of names for standards libraries such as SWIFT. In this case, you might wish to choose between seeing technical names and business names within the same data model.

Setting up Preferred Aliases for a Data Model

The [Aliases](#) property in the Properties window allows you to set up preferred aliases for a data model. For example:



In the above example, two aliases called "english" and "french" are specified in the Aliases field. (Notice how the values should be specified as a comma-delimited list.)


Classification Groups

Classification groups allow you to create a hierarchical structure within a [data model](#), by grouping related [model components](#) together.

Classification groups do not infer a sub-namespace or package hierarchy in the generated code. The [namespace](#) name uniqueness rules still apply across classification groups. It is illegal for two components of the same type to exist with the same name in the same namespace, even if they are in different classification groups.

Adding a Classification Group

To add a classification group:

1. Do one of the following:
 - Right-click the data model (.dod) file in the Explorer window and select **New > Classification Group**.
 - Click the data model (.dod) file in the Explorer window and click  on the data model palette.
2. In the New Classification Group dialog, type the name you want to assign to the classification group and click **OK**. The new component is added to the Explorer window.

Annotations

Annotations allow you to describe model components for other ADS Designer users, developers using the generated code, and consumers of certain types of exported metadata. Every model component can have an annotation. Additionally, parts of a data model can be annotated by including an annotation component inside a classification group.

There are two flavors of annotation:


- Documentation is used for human readable text and to create the Javadoc for classes and methods.
- Appinfo is designed for machine processing

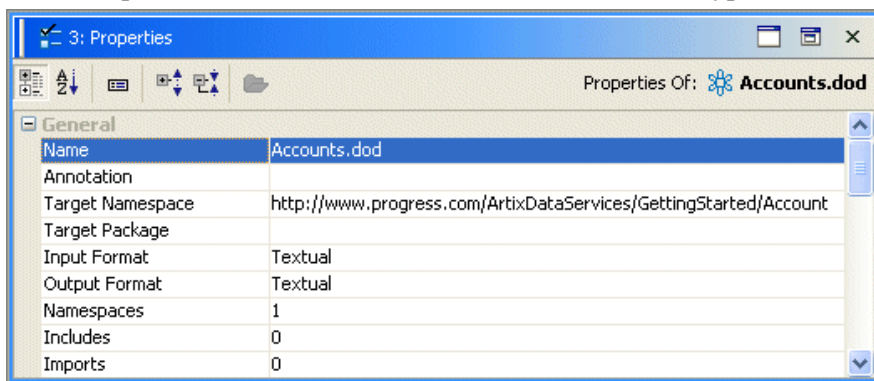
Any number and combination of these can be contained in a single annotation.

See <http://www.w3.org/TR/xmlschema-0/#CommVers> for details.


Adding an Annotation to a Data Model

To add an annotation to a data model:

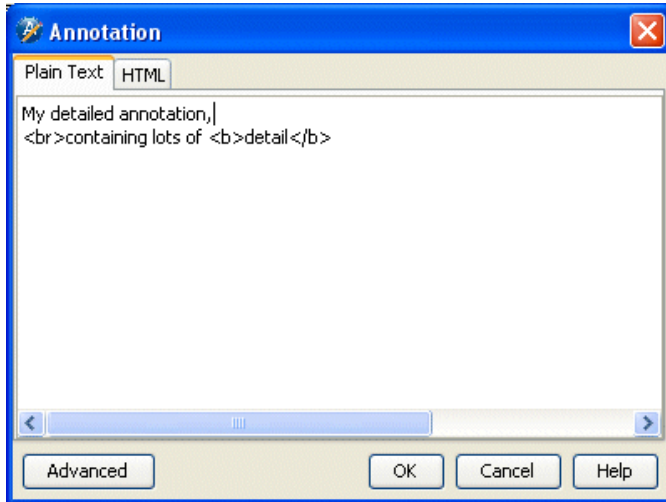
1. Open the New Annotation dialog in either of the following ways:
 - Right-click the data model (.dod) file in the Explorer window and select **New > Annotation**.
 - Click the data model (.dod) file in the Explorer window and click  on the data model palette.
2. Type the name you want to assign to the annotation and click **OK**. The new annotation is added to the Explorer window.
3. In the Properties window, click in the Annotation field and type the details of the annotation.



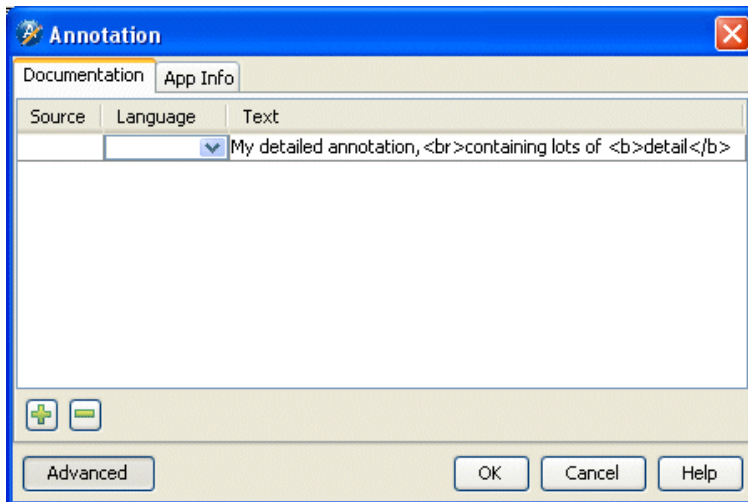
Note: Where an annotation's textual content is greater in length than the Annotation field in the Properties window, an ellipsis indicates that there is more content accessible through the Annotation dialog.



4. To enter a longer annotation, click the  icon to open the Annotation dialog.

5. Type the longer annotation in the Plain Text tab and preview it in the HTML tab.



6. If you wish to set annotation properties that conform to the structure of annotations in XML, click the **Advanced** button. This opens a more advanced version of the Annotation dialog.



7. Each tab contains a table of the annotation entries. The first two columns in the table are used to specify the language the annotation is written in (for example, en, en-US, fr-CA) and the source, which is a valid URI pointing to any relevant address.
Note: The language should preferably be a two-letter code followed by an optional country code (as defined by ISO639). For example, en, en-US, fr-CA, and so on. For languages not listed by ISO639, you can use the i-codes registered with IANA . Alternatively, you can make up your own language by using the prefix "x-" or "X-".
8. Click the  and  icons to add or remove documentation entries respectively. When editing the text of entries in the table, click the button at the end of the row to open a multi-line dialog if required. Because appinfo items are designed for machine processing, there is no language column on that tab.
9. Click **OK** to save your changes.

Adding an Annotation to a Component

You can add an annotation to a particular component using the Annotation property in the [General](#) section of the Properties window.



SWIFT Field Data Types

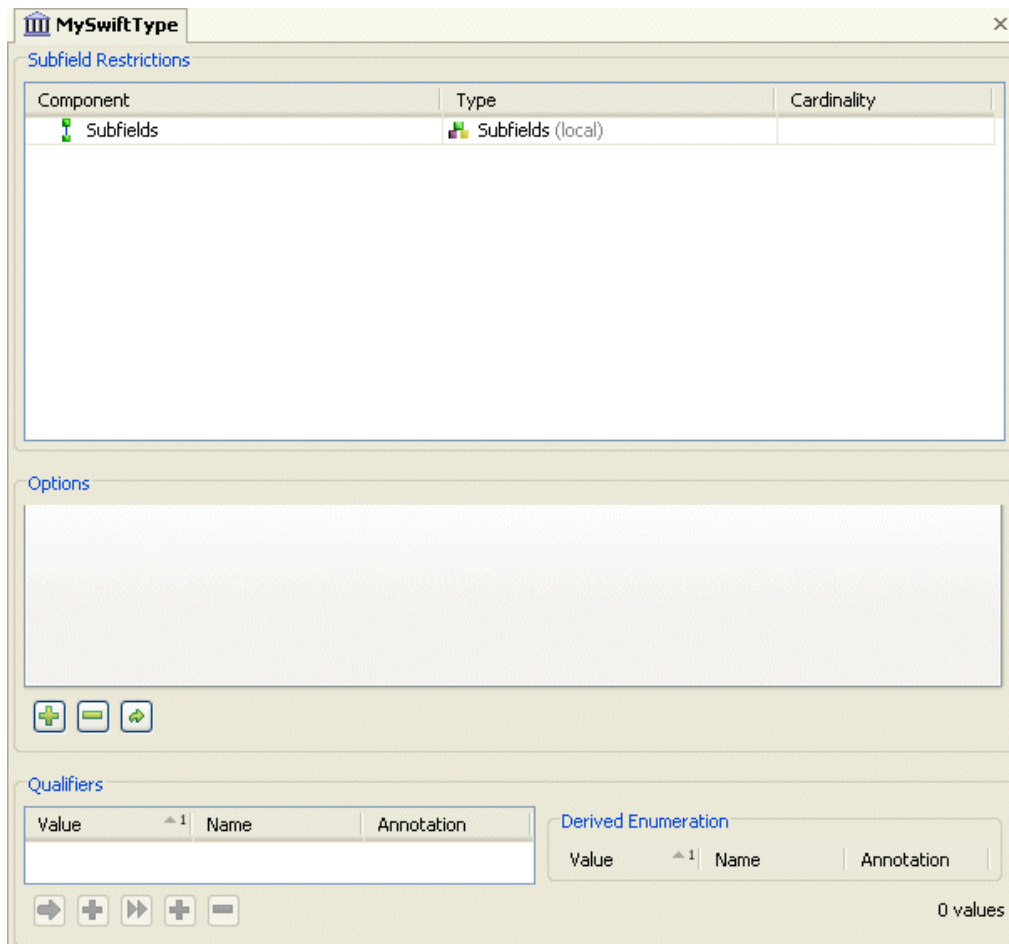
SWIFT Field Data Types are used to create Complex Data Types whose structure is based around the SWIFT specification for SWIFT message fields.

A SWIFT Field Data Type is based on a Complex Data Type, but you can add options, qualifiers, and enumerations to it.

Adding a SWIFT Field Type

To add a SWIFT field type:

1. Open the New SWIFT Data Type dialog in either of the following ways:
 - Right-click the data model (.dod) file in the Explorer window and select **New > SWIFT Field Type**.
 - Click the data model (.dod) file in the Explorer window and click  on the data model palette.
2. Type the name you want to assign to the SWIFT field type and click **OK**.
3. Enter the tag of the new SWIFT data type and click **OK**. The new type is added to the Explorer window. A new tab is also opened for the SWIFT field type in the main window.
4. In the options section, click the  icon to add options to this field, as appropriate.

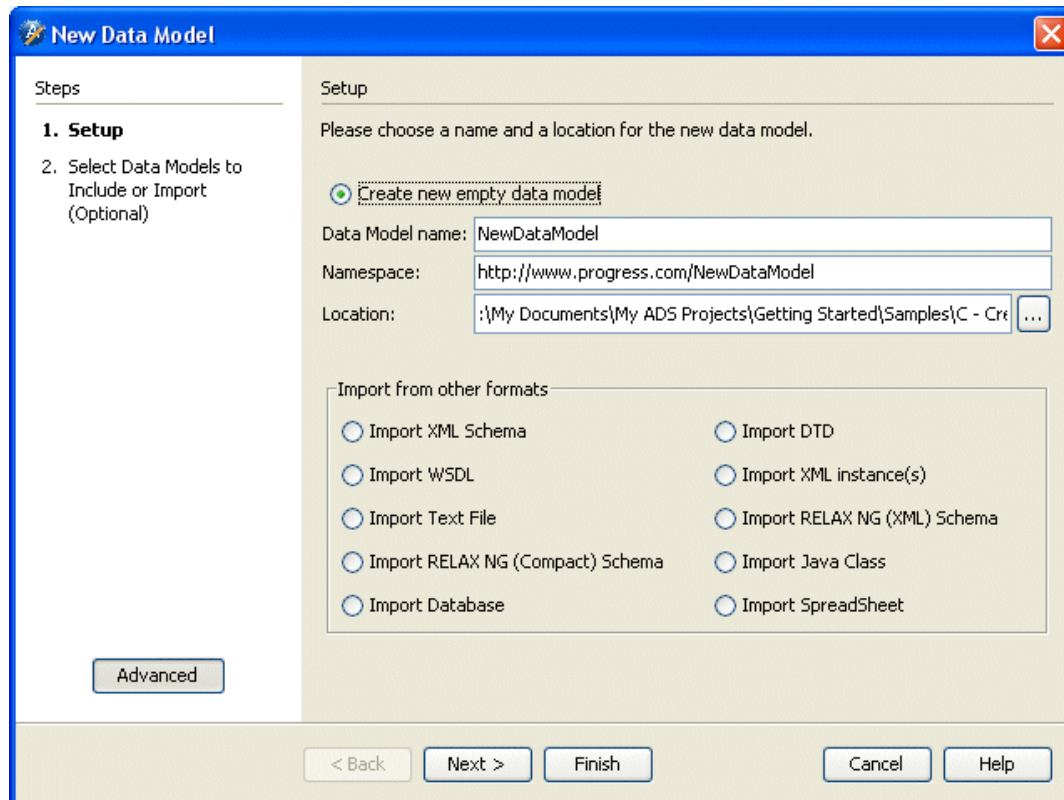


Importing Data Models from Other Formats

ADS allows you to create data models either manually or by importing from other formats.

To start creating a model by importing from another format:

1. Ensure that your project is open in the Project window of the workbench.
2. In the Project window, navigate to the folder where you wish to store the data model, right-click and select **New > Data Model**. Alternatively, select **File > New > Data Model** from the menu bar. This opens the New Data Model wizard.



3. In the Setup panel, click the **Import ...** radio button for the format that you want to import.
4. Click **Finish**.

Click a link below for instructions on importing each format:

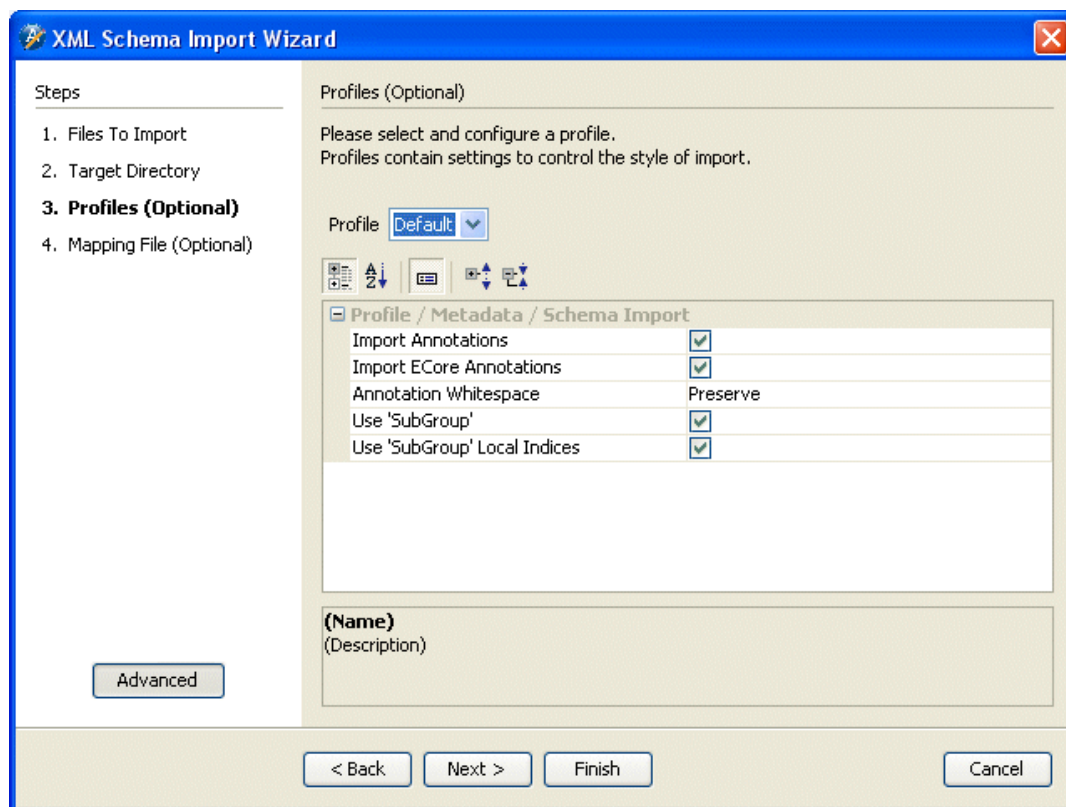
- [XML Schema](#)
- [DTD](#)
- [WSDL](#)
- [XML Instance\(s\)](#)
- [Text File](#)
- [Relax NG \(XML\) Schema](#)
- [Relax NG \(Compact\) Schema](#)
- [Java Class](#)
- [Spreadsheet](#)
- [Database](#)

Importing from an XML Schema

To import a model from an XML schema:

1. Open the XML Schema Import Wizard in any of the following ways:
 - Right-click a folder in the [Project window](#) and select **Import > Import XML Schema**.
 - Select **Tools > Import > Import XML Schema** from the menu bar.
 - Open the [New Data Model Wizard](#), select the Import XML Schema radio button, and click **Finish**.
2. In the Files To Import panel, navigate to the schema (.xsd) file you wish to import.

Note: If the schema to be imported is composed of multiple documents, select the one that contains the root of the instance documents you want to work with. This ensures that all other schema files that are referenced via imports and includes are also imported.
3. Click the file you wish to select and then click **Next**.
4. In the Target Directory panel, browse to the location where you want the new data model to be stored.
5. Click the **Advanced** button on the left of the dialog to enable some optional panels.
6. Click **Next**. This opens the Profiles panel where you can set various values to determine how the import will be handled.



The default profile settings are: [Import Annotations](#), [Import Ecore Annotations](#), [Annotation Whitespace](#), [Use Subgroup](#), and [Use Subgroup Local Indices](#).

7. Click **Next**.
8. In the optional Mapping File panel, browse to a [mapping file](#).
9. Click **Finish** to finish importing the XML schema.

The new data model is loaded and displayed in both the Project window and Explorer window. An exact replica of the schema's contents is displayed in the [Explorer window](#). Any referenced schemas will have been imported as new data models and will be displayed in the [Project window](#).

Note: The directory structure of these new models will reflect the directory structure of the original schemas.

Importing from a DTD

To import a model from a DTD:

1. Open the DTD Import Wizard in any of the following ways:
 - Right-click a folder in the [Project window](#) and select **Import > Import DTD**.
 - Select **Tools > Import > Import DTD** from the menu bar.
 - Open the [New Data Model Wizard](#), select the **Import DTD** radio button, and click **Finish**.
2. In the Files To Import panel, navigate to the XML DTD (.dtd) file you wish to import.
3. Click the file you wish to select and then click **Next**.
4. In the Target Directory panel browse to and select the location where you want the new data model to be stored.
5. Click the **Advanced** button on the left of the dialog to enable some optional panels.
6. Click **Next**. This opens the optional Profiles panel where you can set various values to determine how the import will be handled.
The available profile settings are [Import Via Castor](#) and [Import Mixed Content As String](#).
7. Click **Next**.
8. In the optional Mapping File panel, browse to a [mapping file](#).
9. Click **Finish** to finish importing the DTD.

The new data model is loaded and displayed in both the Project window and Explorer window.

Importing from a WSDL File

To import a model from a WSDL file:

1. Open the WSDL Import Wizard in any of the following ways:
 - Right-click a folder in the [Project window](#) and select **Import > Import WSDL**.
 - Select **Tools > Import > Import WSDL** from the menu bar.
 - Open the [New Data Model Wizard](#), select the **Import WSDL** radio button, and click **Finish**.
2. In the Files To Import panel, nNavigate to the WSDL (.wsdl) file you wish to import.
3. Click the file you wish to select and then click **Next**.
4. In the Target Directory panel, browse to the location where you want the new data model to be stored.
5. Click **Finish** to finish importing the WSDL file.

The new data model is loaded and displayed in both the Project window and Explorer window.

Importing from an XML Instance

To import a model from an XML instance:

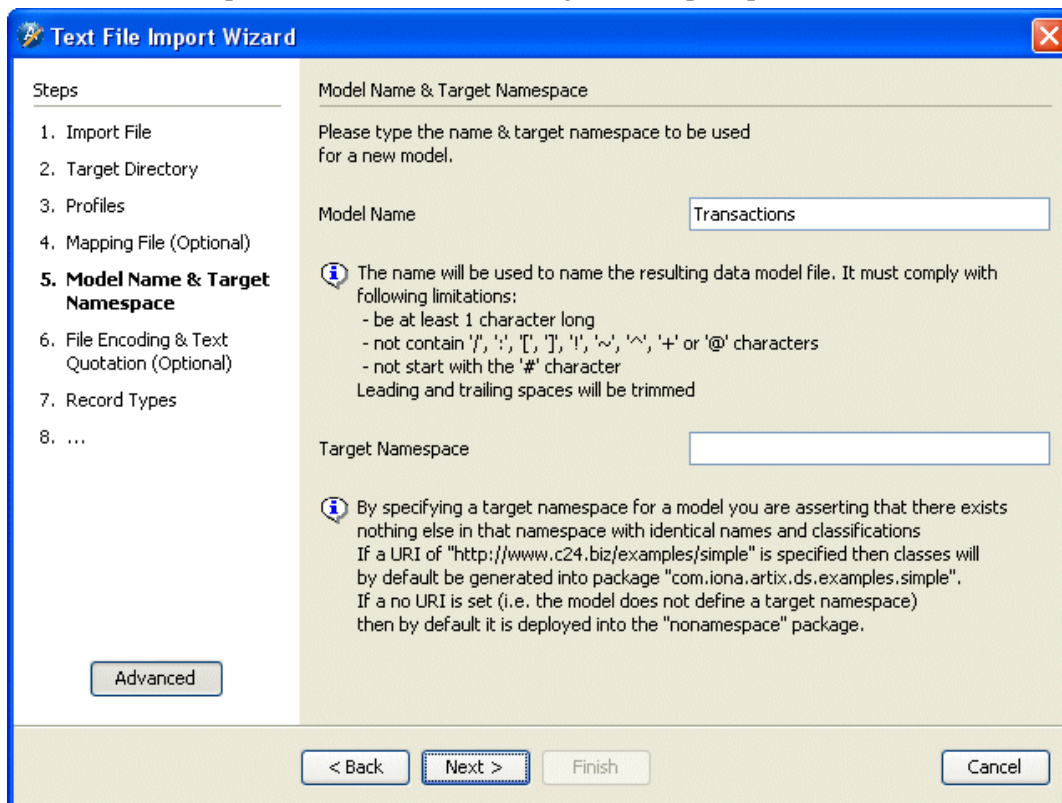
1. Open the WSDL Import Wizard in any of the following ways:
 - Right-click a folder in the [Project window](#) and select **Import > Import XML Instance(s)**.
 - Select **Tools > Import > Import XML Instance(s)** from the menu bar.
 - Open the [New Data Model Wizard](#), select the **Import XML Instance(s)** radio button, and click **Finish**.
2. In the Files To Import panel, navigate to the XML (.xml) file you wish to import.
3. Click the file you wish to select and then click **Next**.
4. In the Target Directory panel, browse to the location where you want the new data model to be stored.
5. Click **Finish** to finish importing the XML file.
6. The new data model is loaded and displayed in both the Project window and Explorer window.

Importing from a Text File

To import a model from a text file:

1. Open the Text File Import Wizard in any of the following ways:
 - Right-click a folder in the [Project window](#) and select **Import > Import Text File**.
 - Select **Tools > Import > Import Text File** from the menu bar.
 - Open the [New Data Model Wizard](#), select the **Import Text File** radio button, and click **Finish**.
2. In the Import File panel, navigate to the text (.txt) file you wish to import.
3. Click the file you wish to select and then click **Next**.
4. In the Target Directory panel, browse to the location where you want the new data model to be stored.
5. Click **Next**. This opens the Profiles panel where you can set various custom values to determine how the import will be handled.

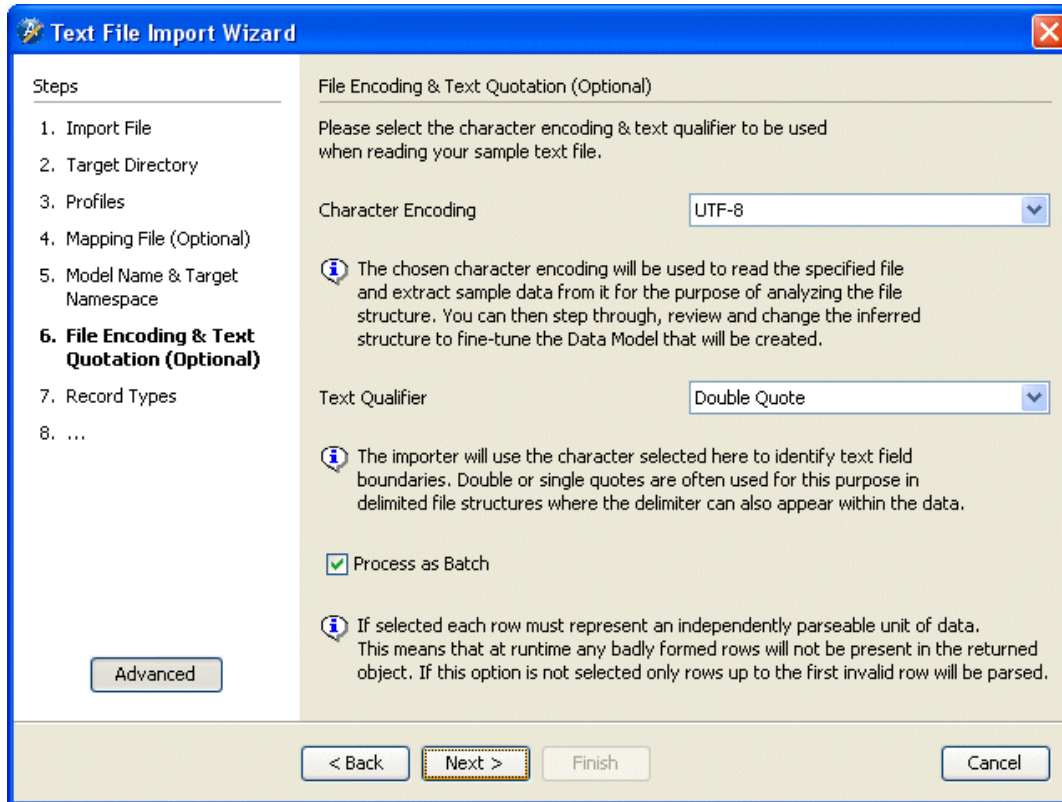
Note: There are no profile settings by default for importing a text file.
6. Click the **Advanced** button on the left of the dialog to enable some optional panels.
7. Click **Next**.
8. In the optional Mapping File panel, browse to a [mapping file](#).
9. Click **Next**. This opens the Model Name & Target Namespace panel.



10. The model name is based by default on the name of the text file you are importing (for example, "Transactions" as shown above). If you do not want to accept the default name, type an alternative in the Model Name field.
11. Type a namespace for the model in the Target Namespace, (for example, <http://www.progress.com/ArtixDataServices/GettingStarted/Transaction>). You do not need to specify a

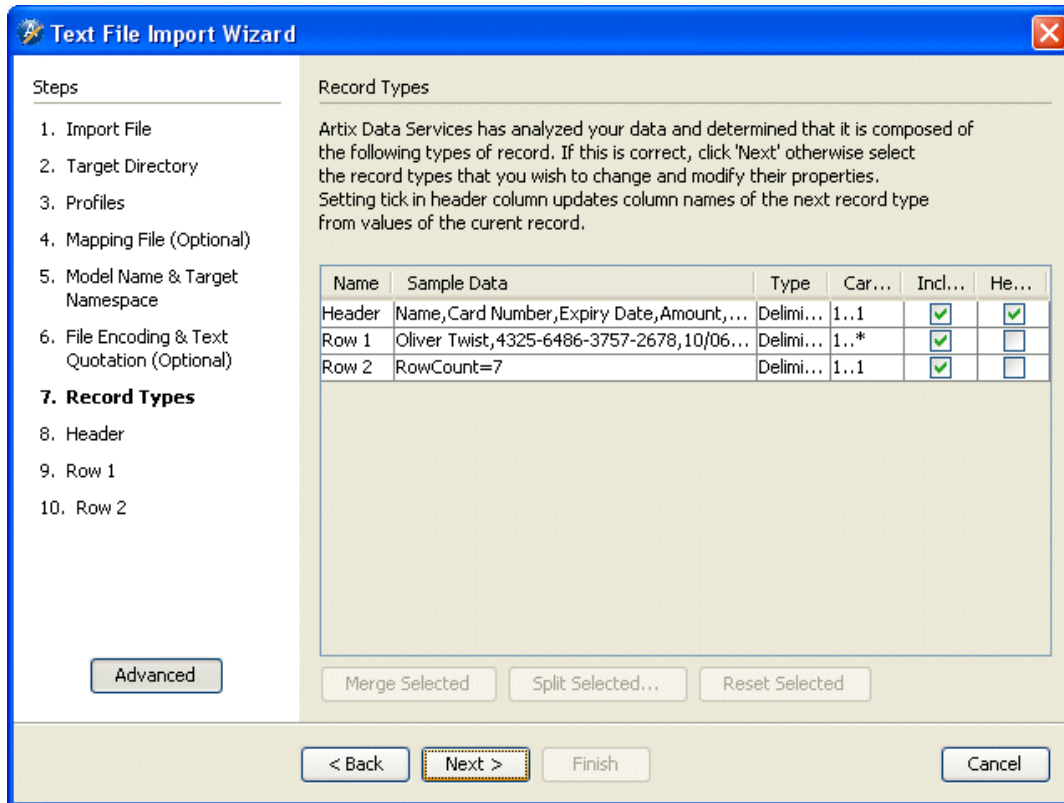
namespace, because you can always set one later in the Properties window for the model after you have finished importing the text file.

12. Click **Next**. This opens the File Encoding & Text Quotation panel.

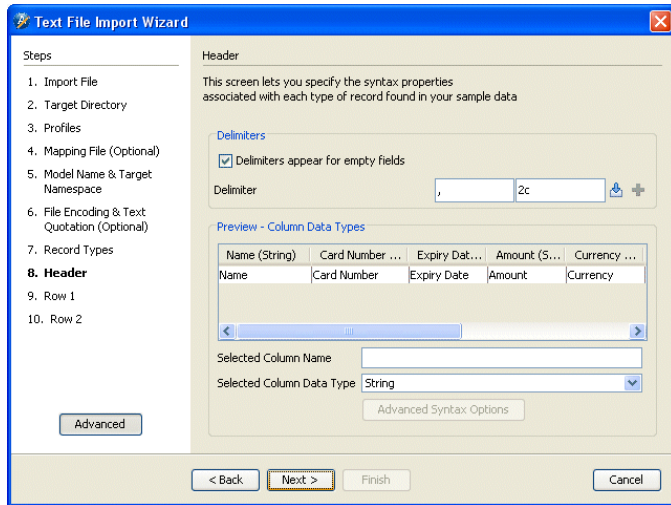


13. Select the character encoding format that you want to be used for the purposes of reading and analyzing the imported text file. The default format is UTF-8.
14. Select the text character that you want to be used to indicate text field boundaries. The default is a double quote.
15. Check the Process as Batch check box if you want each row of your text file to be parsed as an independent unit. Any invalid (badly formed) rows will not be parsed. This option is not selected by default. If this option is not selected, any valid rows following an invalid row will not be parsed either.

16. Click **Next**. This opens the Record Types panel.



17. The following panels allow you to edit the properties associated with the each type of record found in the sample data. Click **Finish** once you've checked the remaining panels.



Importing from a RELAX NG (XML) Schema

To import a model from a RELAX NG (XML) schema:

1. Open the RELAX NG (XML) Schema Import Wizard in any of the following ways:
 - Right-click a folder in the [Project window](#) and select **Import > Import RELAX NG (XML) Schema**.
 - Select **Tools > Import > Import RELAX NG (XML) Schema** from the menu bar.
 - Open the [New Data Model Wizard](#), select the **Import RELAX NG (XML) Schema** radio button, and click **Finish**.
2. In the Files To Import panel, navigate to the RELAX NG XML (.rng) schema you wish to import.
3. Click the file you wish to select and then click **Next**.
4. In the Target Directory panel, browse to the location where you want the new data model to be stored.
5. Click **Finish** to finish importing the XML file.

The new data model is loaded and displayed in both the Project window and Explorer window.

Importing from a RELAX NG (Compact) Schema

To import a model from a RELAX NG (Compact) schema:

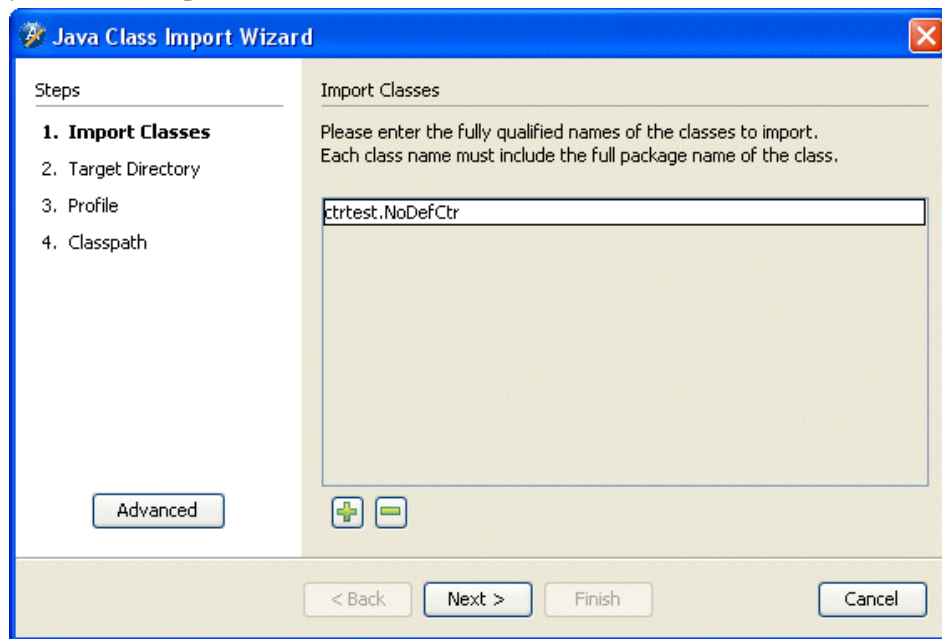
1. Open the RELAX NG (Compact) Schema Import Wizard in any of the following ways:
 - Right-click a folder in the [Project window](#) and select **Import > Import RELAX NG (Compact) Schema**.
 - Select **Tools > Import > Import RELAX NG (Compact) Schema** from the menu bar.
 - Open the [New Data Model Wizard](#), select the **Import RELAX NG (Compact) Schema** radio button, and click **Finish**.
2. In the Files To Import panel, navigate to the RELAX NG Compact (.rnc) schema you wish to import.
3. Click the file you wish to select and then click **Next**.
4. In the Target Directory panel, browse to the location where you want the new data model to be stored.
5. Click **Finish** to finish importing the RELAX NG (XML) schema.

The new data model is loaded and displayed in both the Project window and Explorer window.

Importing from a Java Class

To import a model from a Java class:

1. Open the Java Class Import Wizard in any of the following ways:
 - Right-click a folder in the [Project window](#) and select **Import > Import Java Class**.
 - Select **Tools > Import > Import Java Class** from the menu bar.
 - Open the [New Data Model Wizard](#), select the **Import Java Class** radio button, and click **Finish**.
2. In the Import Class panel, type the fully qualified name (including the full package name) of the Java class you wish to import in the Class Name field.

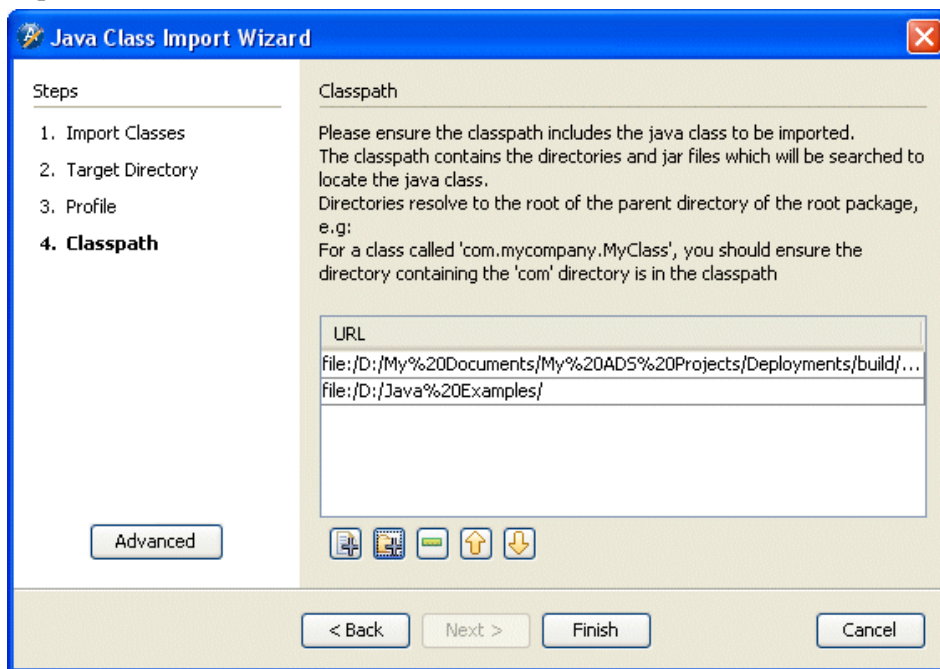


3. Click **Next**.
4. In the Target Directory panel, browse to the location where you want the new data model to be stored.
5. Click **Next**. This opens the Profile panel where you can set various values to determine how the import will be handled.








The default profile settings are: [Exclude Primitive Flags](#), [Access Modifier Level](#), [Include Transient Fields](#) and [Import Simple Data Types as Attributes](#).

6. Click **Next**. This opens the Classpath panel where you can include the classpath for the Java class you wish to import.



Note: The class must exist in the classpath of the Profile settings.

7. Click the  icon to add a .jar or .zip file to the classpath. Alternatively, click the  icon to add a directory to the classpath.

Note: You can delete a classpath entry by clicking that entry and then clicking the  icon. Click the  and  icons to move items up and down the list, as appropriate.
8. Click the **Advanced** button on the left of the dialog to enable the optional Mapping File panel, then click **Next**.
9. In the Mapping File panel, browse to and select a [mapping file](#).
10. Click **Finish** to finish importing the Java class.

The new data model is loaded and displayed in both the Project window and Explorer window.

Importing from a Database

To import a model from a database:

1. Open the Import Database Wizard in any of the following ways:
 - Right-click a folder in the [Project window](#) and select **Import > Import Database**.
 - Select **Tools > Import > Import Database** from the menu bar.
 - Open the [New Data Model Wizard](#) and select the **Import Database** radio button, and click **Finish**.
2. In the Target Directory panel, browse to the location where you want the new data model to be stored.
3. Click **Next**. This opens the Connection Properties panel where you can set the details of the database you wish to import from.

Enter the following details:

Model Name

Type the name of the data model you wish to create. Ensure that there is not already a model of the same name in the target directory you selected on the previous panel.

Target Namespace

Type a namespace for the model if you want. You do not need to specify a namespace because you can always set one later in the Properties window for the model after you have finished importing the database.

Database Dialect

Select a source database type from the drop-down list.

JDBC Driver Class Name

This is populated automatically, depending on what you select in the **Database Dialect** field.

Database URL

The initial components of the URL are provided for you, based on the database type that you selected. You need to complete the URL to the database that you are importing.

Username

Type the username that you use to connect to the database.

Password

Type the password that you use to connect to the database.






Catalog

Select the checkbox to restrict the database search by using a catalog and type the relevant catalog name in the corresponding text field.

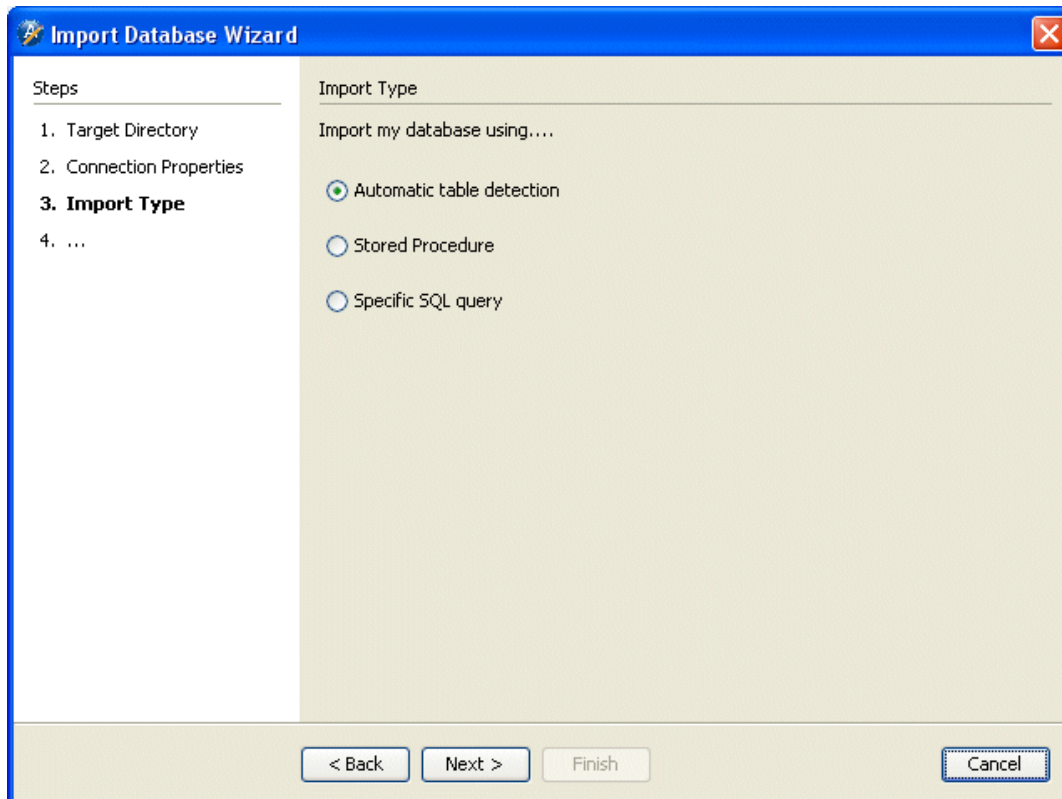
Schema

Select the checkbox to restrict the database search by using a schema and type the relevant schema name in the corresponding text field.

Edit Classpath

Click this if your JDBC driver is not already in the classpath. Click the  icon to add a .jar or .zip file to the classpath. Alternatively, click the  icon to add a directory to the classpath. You can delete a classpath entry by selecting it and clicking the  icon. Click the  and  icons to move items up and down the list, as appropriate. Click **Edit Classpath** again to return to the original Connection Properties panel.

4. Click **Next**. This opens the Import Type panel.

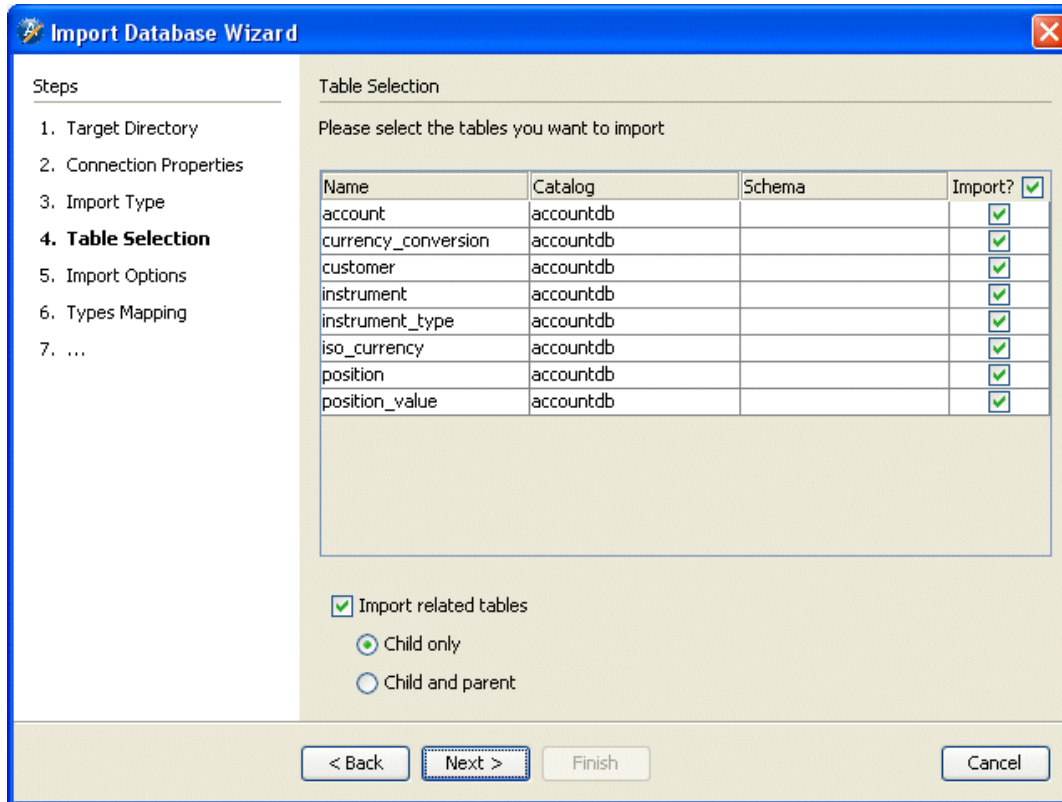


5. Select the radio button that corresponds to how you want to import your database. The options available are [Automatic table detection](#), [Stored Procedure](#), and [Specific SQL query](#).

Import Using Automatic Table Detection

If you select the Automatic table detection radio button:

1. Click **Next**. This opens the Table Selection panel.

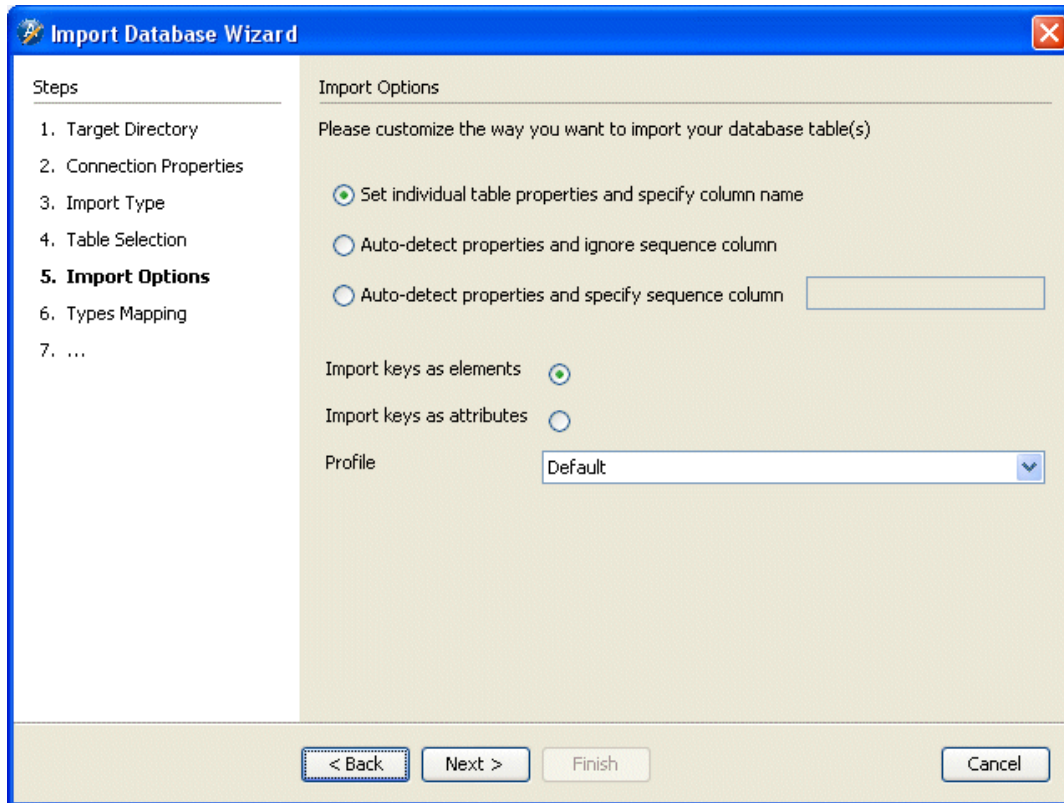


2. Select the database tables that you wish to import.
Note: All tables are selected by default. If you do not wish to import a particular table, uncheck the check box corresponding to that table in the list.
3. The Import related tables check box is checked by default. If you do not wish to import related tables uncheck that check box.

Note the following:

- If **Import related tables** is checked and **Child only** is selected, child tables of the selected tables will also be imported even if they are not selected.
- If **Import related tables** is checked and **Child and Parent** is selected, both child and parent related tables will be imported even if they are not selected.
- If **Import related tables** is not checked, only the selected tables in the list will be imported and relationships will not be maintained.

4. Click **Next**. This opens the Import Options panel.



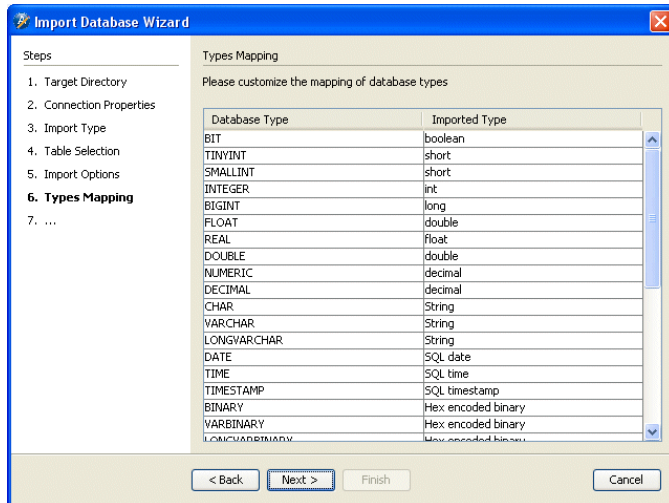
5. Select how you wish to import your database tables. There are three options available:

- Select **Set individual table properties and specify column name** if you want to step through all the tables and set the properties manually.
- Select **Auto-detect properties and ignore sequence column** if you want primary keys and table types to be detected and you want all columns to be set with the name equal to the one provided, as the child order index. (The child order is used by Hibernate to determine the index of a stored record.)
- Select **Auto-detect properties and specify sequence column** if you want primary keys and table types to be detected and you want the child order index to be ignored.

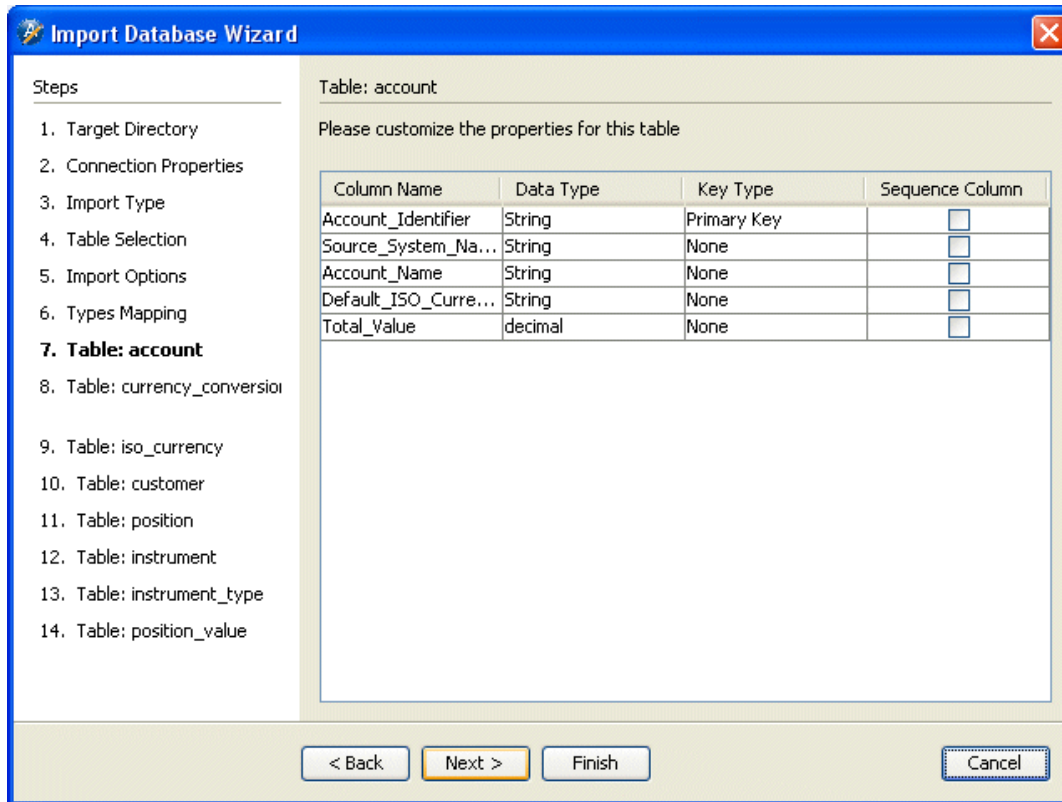
6. Select whether you want to import keys as [elements](#) or [attributes](#).

7. Select the profile setting that you want to use to determine how the import will be handled.

8. Click **Next**. This opens the Types Mapping panel which displays a table of database types and the imported (ADS) types that they equate to.



9. If you want to change the mapping for a particular database type, click the corresponding cell in the Imported Type column and click the down arrow to select from the list of available types.
10. Click **Next**. Panels become available for each of the tables involved in the database import. For example:



This means that you can go through each table in turn and change the data type for individual columns.

11. To change the data type for a particular column, click in the corresponding Data Type cell and click the down arrow to select from the list of available types, as appropriate.
12. If you wish to indicate that a particular column is a sequence column, check the check box in the corresponding Sequence Column cell.

Note: You can only do this for columns that are not primary keys or foreign keys.

Note: A sequence column in this case should not be confused with the concept of a sequence column in database parlance. In the ADS context, specifying a column as a sequence column means that records read from this table will be ordered according to the ordering specified in the sequence column. Typically a sequence column will be some kind of numeric type, so that the numerical order of records in this column will determine the order in which the rest of the records in the table are loaded into a list or array.

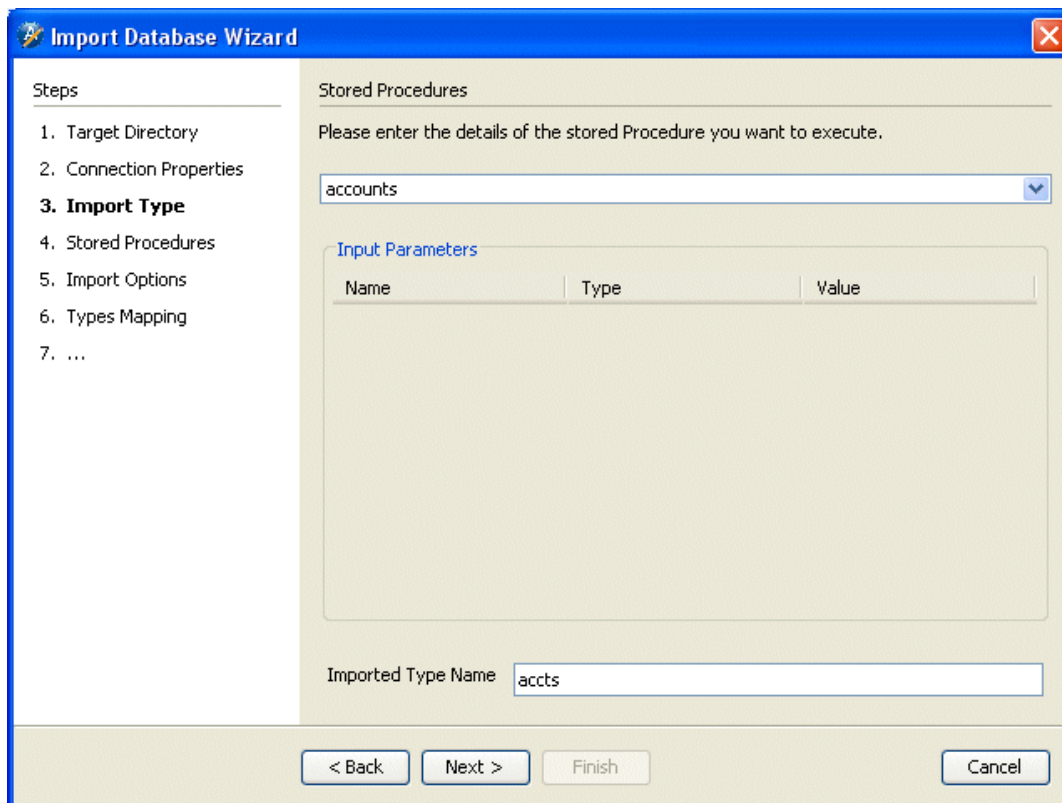
- When you have finished going through the panel for each table in turn, click **Finish** to finish importing the database.

The new data model is loaded and displayed in both the Project window and Explorer window.

Import Using Stored Procedure

If you select the Stored Procedure radio button:

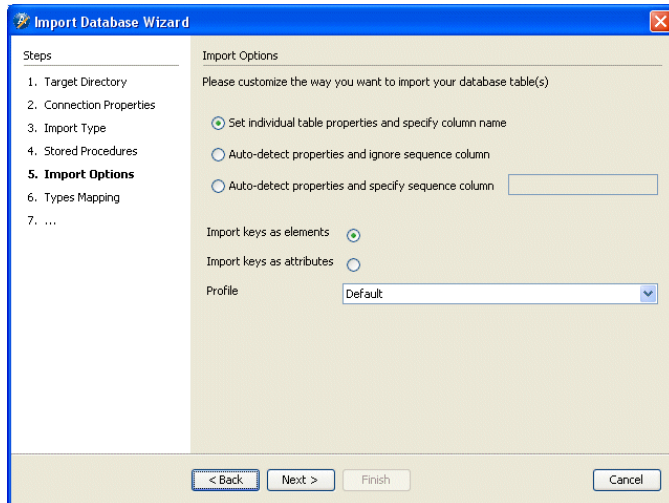
- Click **Next**. This opens the Stored Procedures panel.



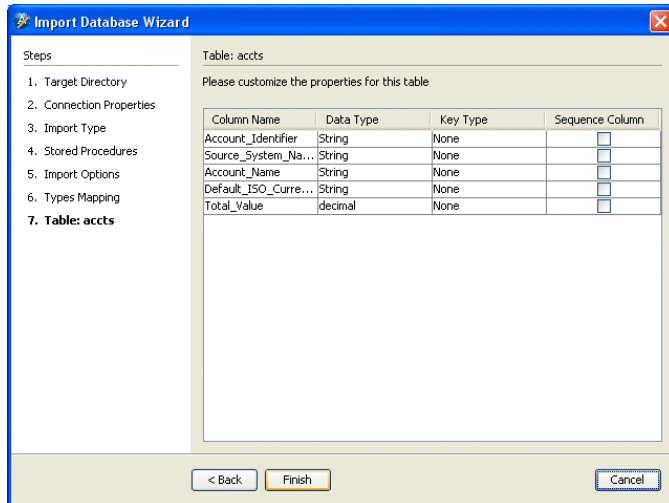
- Click the down arrow to select the stored procedure you wish to use.

The relevant parameters for the stored procedure are displayed.

- Enter relevant values for the parameters in the Value column.
- Enter the name of the complex type you want to be created in the Imported Type Name field.
- Click **Next**. This opens the Import Options panel.



6. Select how you wish to import your database tables. There are three options available:
 - Select **Set individual table properties and specify column name** if you want to step through all the tables and set the properties manually.
 - Select **Auto-detect properties and ignore sequence column** if you want primary keys and table types to be detected and you want all columns to be set with the name equal to the one provided, as the child order index. (The child order is used by Hibernate to determine the index of a stored record.)
 - Select **Auto-detect properties and specify sequence column** if you want primary keys and table types to be detected and you want the child order index to be ignored.
7. Select whether you want to import keys as [elements](#) or [attributes](#).
8. Select the profile setting that you want to use to determine how the import will be handled.
9. Click **Next**. This opens the Types Mapping panel which displays a table of database types and the imported (ADS) types that they equate to.
10. If you want to change the mapping for a particular database type, click the corresponding cell in the Imported Type column and click the down arrow to select from the list of available types.
11. If you selected **Auto-detect properties and ignore sequence column** or **Auto-detect properties and specify sequence column** on the Import Options panel, skip to step 13.
If you selected **Set individual table properties and specify column name** on the Import Options panel, click **Next** to open a panel for the table involved in the database import. For example:



12. If you wish to change the data type for a particular column, click in the corresponding Data Type cell and click the down arrow to select from the list of available types, as appropriate.
13. If you wish to indicate that a particular column is a sequence column, check the check box in the corresponding Sequence Column cell.

Note: You can only do this for columns that are not primary keys or foreign keys.

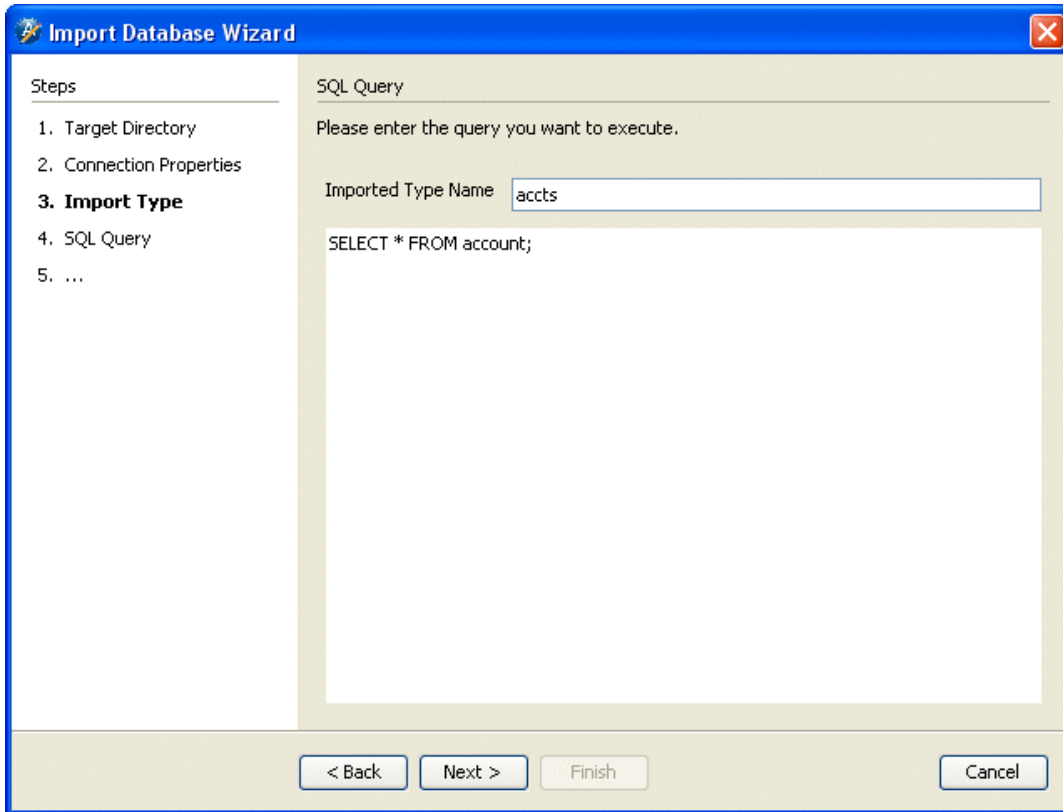
Note: A sequence column in this case should not be confused with the concept of a sequence column in database parlance. In the ADS context, specifying a column as a sequence column means that records read from this table will be ordered according to the ordering specified in the sequence column. Typically a sequence column will be some kind of numeric type, so that the numerical order of records in this column will determine the order in which the rest of the records in the table are loaded into a list or array.

14. Click **Finish** to finish importing the database.
15. The new data model is loaded and displayed in both the Project window and Explorer window.

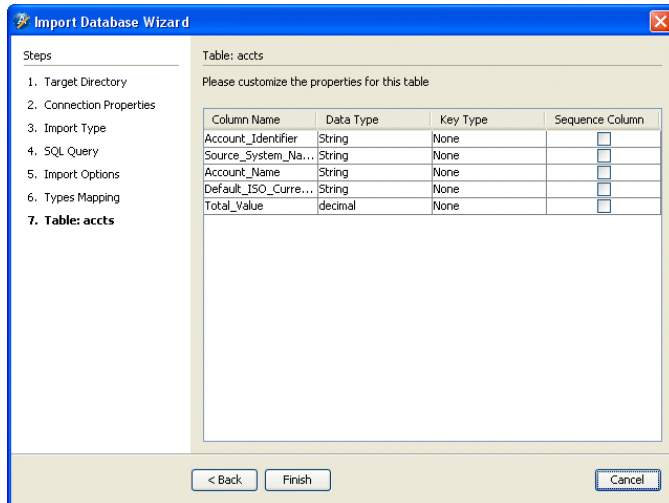
Import Using Specific SQL Query

If you select the Specific SQL query radio button:

1. Click **Next**. This opens the SQL Query panel.



2. Enter the name of the complex type you want to be created in the Imported Type Name field.
3. Enter the SQL query you want to be executed in the available text box.
4. Click **Next**. This opens the Import Options panel.
5. Select how you wish to import your database tables. There are three options available:
 - Select **Set individual table properties and specify column name** if you want to step through all the tables and set the properties manually.
 - Select **Auto-detect properties and ignore sequence column** if you want primary keys and table types to be detected and you want all columns to be set with the name equal to the one provided, as the child order index. (The child order is used by Hibernate to determine the index of a stored record.)
 - Select **Auto-detect properties and specify sequence column** if you want primary keys and table types to be detected and you want the child order index to be ignored.
6. Select whether you want to import keys as [elements](#) or [attributes](#).
7. Select the profile setting that you want to use to determine how the import will be handled.
8. Click **Next**. This opens the Types Mapping panel which displays a table of database types and the imported (ADS) types that they equate to.
9. If you want to change the mapping for a particular database type, click the corresponding cell in the Imported Type column and click the down arrow to select from the list of available types.
10. If you selected **Auto-detect properties and ignore sequence column** or **Auto-detect properties and specify sequence column** on the Import Options panel, skip to step 13.
If you selected **Set individual table properties and specify column name** on the Import Options panel, click **Next** to open a panel for the table involved in the database import. For example:



11. If you wish to change the data type for a particular column, click in the corresponding Data Type cell and click the down arrow to select from the list of available types, as appropriate.
12. If you wish to indicate that a particular column is a sequence column, check the check box in the corresponding Sequence Column cell.

Note: You may only do this for columns that are not primary keys or foreign keys.

Note: A sequence column in this case should not be confused with the concept of a sequence column in database parlance. In the ADS context, specifying a column as a sequence column means that records read from this table will be ordered according to the ordering specified in the sequence column. Typically a sequence column will be some kind of numeric type, so that the numerical order of records in this column will determine the order in which the rest of the records in the table are loaded into a list or array.

13. Click **Finish** to finish importing the database.

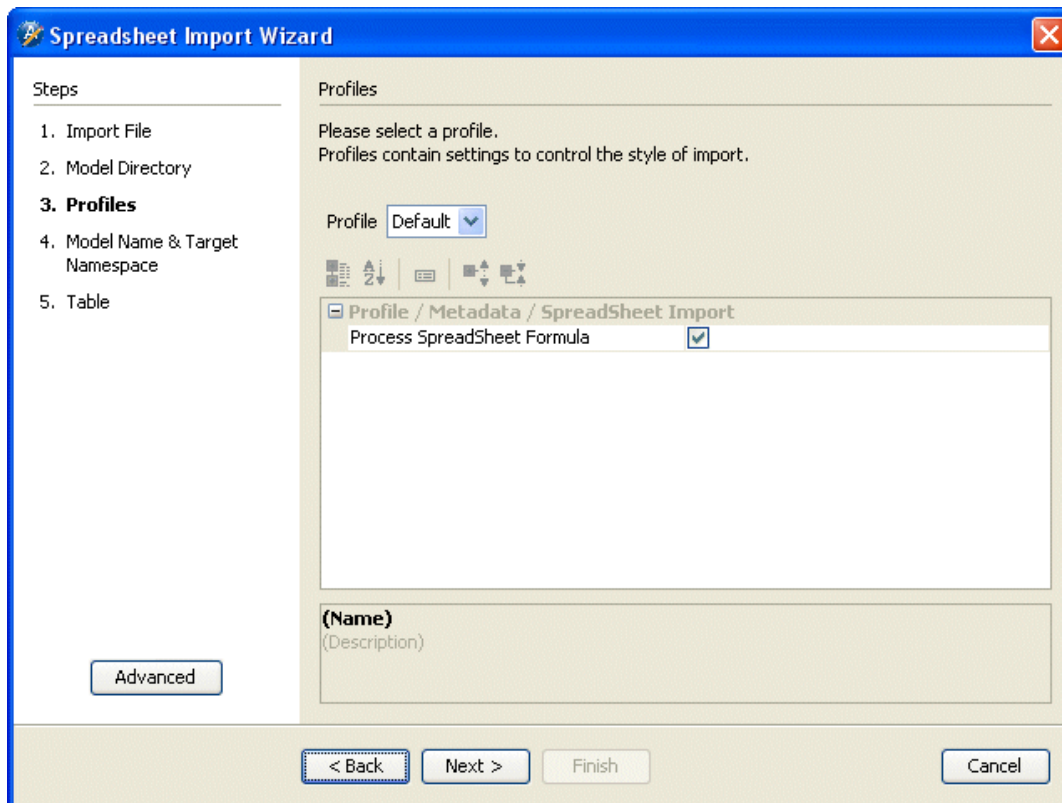
The new data model is loaded and displayed in both the Project window and Explorer window.

Importing from an Excel Spreadsheet

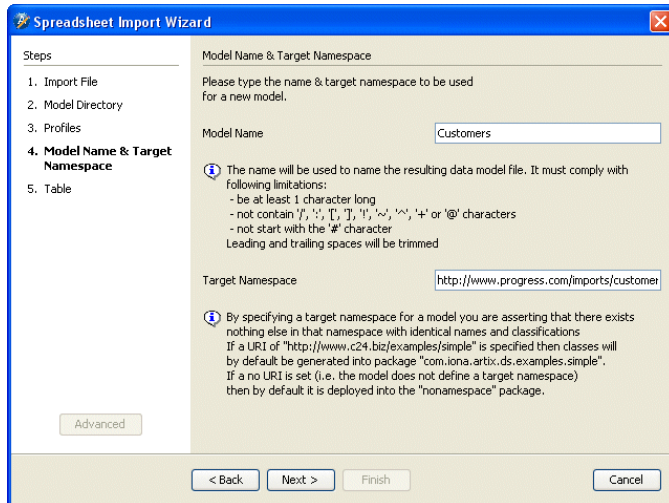
You can import a data model from a Microsoft Excel spreadsheet created in the default file format using Excel 97 or later.

To import a model from an Excel spreadsheet:

1. Open the Spreadsheet Import Wizard in any of the following ways:
 - Right-click a folder in the [Project window](#) and select **Import > Import Spreadsheet** .
 - Select **Tools > Import > Import Spreadsheet** from the menu bar.
 - Open the [New Data Model Wizard](#) and select the **Import Spreadsheet** radio button, and click **Finish**.
2. In the Import File panel, browse to and select the source spreadsheet (.xls or .xlsx) file.
3. In the Model Directory panel, browse to the location where you want the new data model to be stored, then click **Next**.
4. In the Profiles panel, enable or disable the [Process Spreadsheet Formula](#) option.



5. Click the **Advanced** button on the left of the dialog to enable the optional Mapping File panel, then click **Next**.
6. In the Mapping File panel, browse to and select a [mapping file](#), then click **Next**.
7. In the Model Name & Target Namespace panel, enter the name and target namespace to be used for the new model, then click **Next**.



The wizard now scans the first 50 rows of the spreadsheet to detect the tables. Click **Next**.

- In the Table panel, you can review the structure of the imported data. Each row represents a table. You can change the following options as required.

Cells

The cell scope of the imported table. The importer uses the R1C1 referencing method. See the informational note in the Table panel explaining how each cell is scoped.

Name

The name of the imported data type

Spread

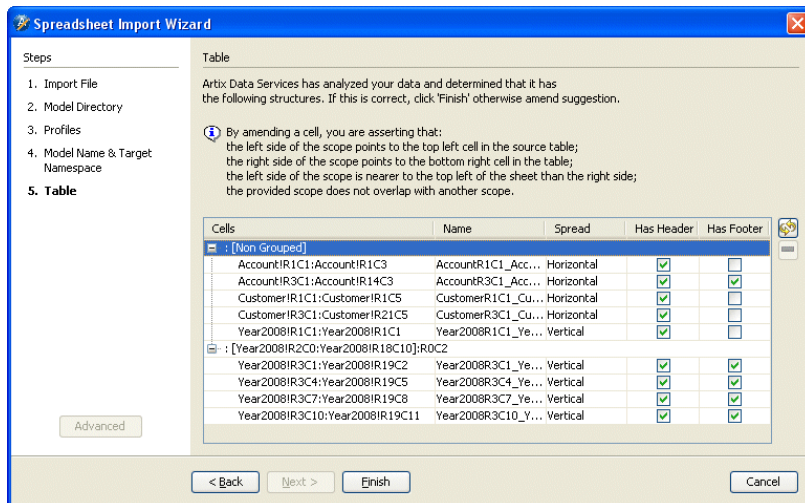
Whether the data in the imported table is arranged vertically (by row) or horizontally (by column)


Has Header


Whether the imported table includes a header

Has Footer

Whether the imported table includes a footer



To rerun the data analysis, click the  button.

To remove a table, click on the table and click the  button.

9. Click **Finish** to complete the import process.

Using a Mapping File

When creating a data model by [importing from another format](#), you can write your own XML mapping file to overwrite or add to the properties or components in the imported data model. This technique is often used in ADS to apply validation rules that are specified in a different model.

You can find the schema for mapping files in the `ADS_HOME/lib/artix-ds-schema-version.jar` in the `META-INF/schema` folder. The file is called

For an example of a mapping file, see the following file in the FpML standards library;

My ADS Projects/Standards Libraries/FpML/FpML Version 4.4/FpML 4.4 WD 2007-12-24/FpML Rules 4.4 WD 2007-12-24.xml

Exporting Models to Other Formats

ADS allows you to export your data models to any of the following formats for external consumption:

- [XML Schema](#)
- [DTD](#)
- [RELAX NG \(XML\) Schema](#)
- [RELAX NG \(Compact\) Schema](#)
- [HTML](#)

Note: Exporting a data model to a database is only supported through Hibernate. You can generate Hibernate Mappings using ADS Designer.

Exporting to XML Schema

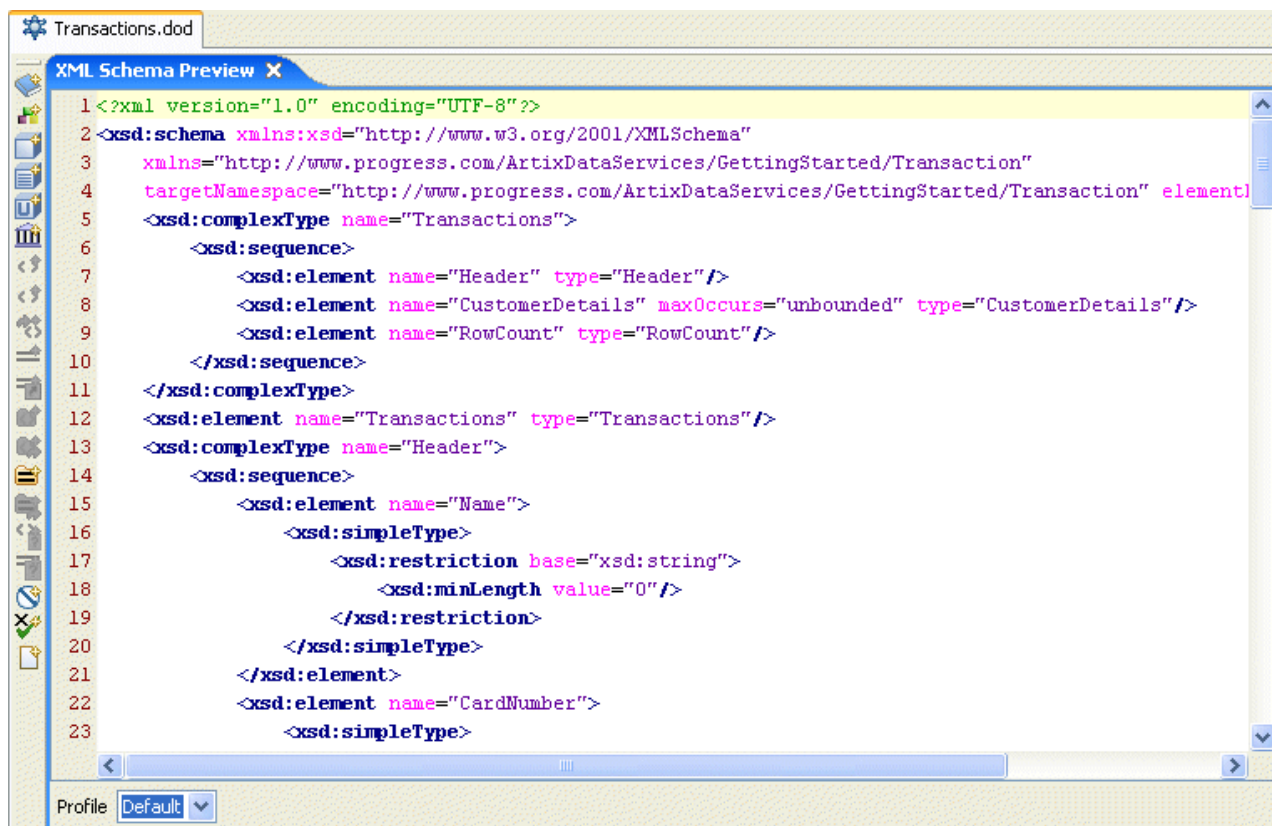
Before you export a data model as a W3C XML schema document, you can preview it.

Preview

To preview an XML schema, do either of the following:

- Right-click the model (.dod) file in the [Project window](#) and select **Preview > Preview XML Schema**.
- Alternatively, click the model (.dod) file in the [Project window](#) and select **Tools > Preview > Preview XML Schema** from the menu bar.

This opens an XML Schema Preview tab in the main window of the workbench.



Export

To export a model as an XML schema:

1. Open the XML Schema Export Wizard in either of the following ways:
 - Right-click the model (.dod) file in the [Project window](#) and select **Export > Export XML Schema**.

- Click the model (.dod) file in the [Project window](#) and select **Tools > Export > Export XML Schema** from the menu bar.

This opens the Export Models panel of the XML Schema Export Wizard.

2. Navigate to the model(s) (.dod) file(s) you wish to export.
3. Click the file(s) you wish to select. To select more than one file, shift-click each filename.
4. Click **Next**.
5. In the Target Directory panel, browse to the location where you want the new schema to be stored.
6. Click the **Advanced** button on the left of the dialog to enable the Profile panel.
7. Click **Next**. This opens the Profile panel where you can set various values to determine how the export will be handled.

The default profile settings are: [Export Annotations](#), [Export ECore Annotations](#), [Export Default Occurrences](#), [Flatten Directories](#), [Inline](#), [Inline Minimal](#), [XML Namespace Schema Location](#).

8. Click **Finish** to finish exporting the XML schema.

The new schema is saved to the location you have specified. If a model you are exporting contains references to other models, they will be exported to schema files in packages whose structure mimics their original ADS models.

Exporting to a DTD

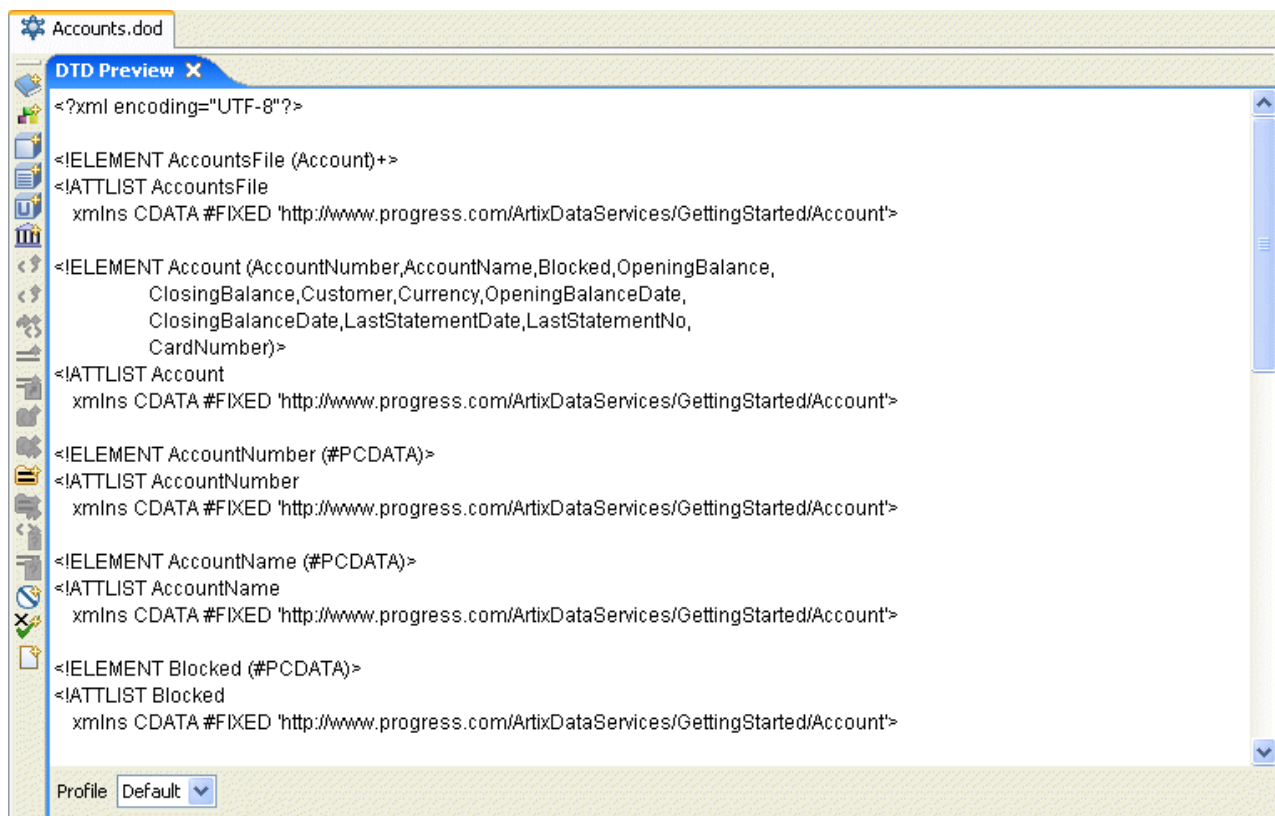
Before you export a data model as a DTD, you can preview it.

Preview

To preview a DTD, do either of the following:

- Right-click the model (.dod) file in the [Project window](#) and select **Preview > Preview DTD**.
- Alternatively, click the model (.dod) file in the [Project window](#) and select **Tools > Preview > Preview DTD** from the menu bar.

This opens a DTD Preview tab similar to the following in the main window of the workbench:



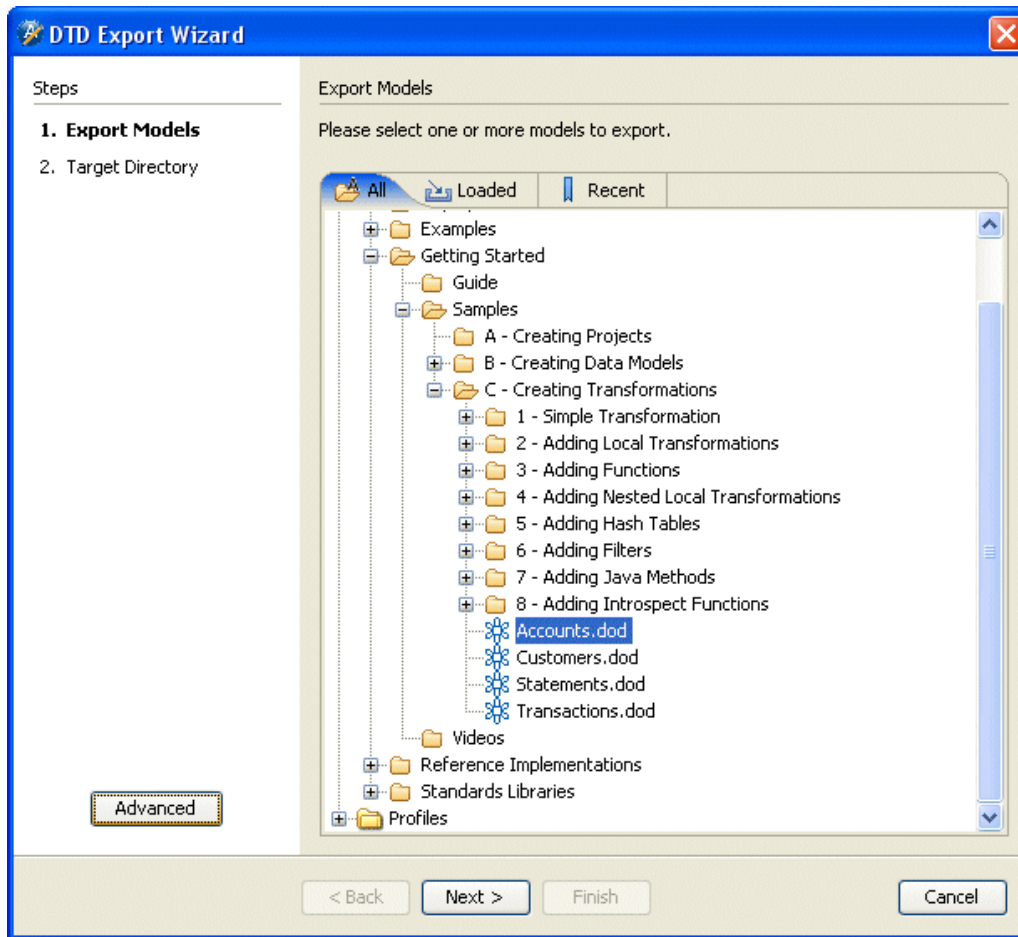
Export

To export a model as a DTD:

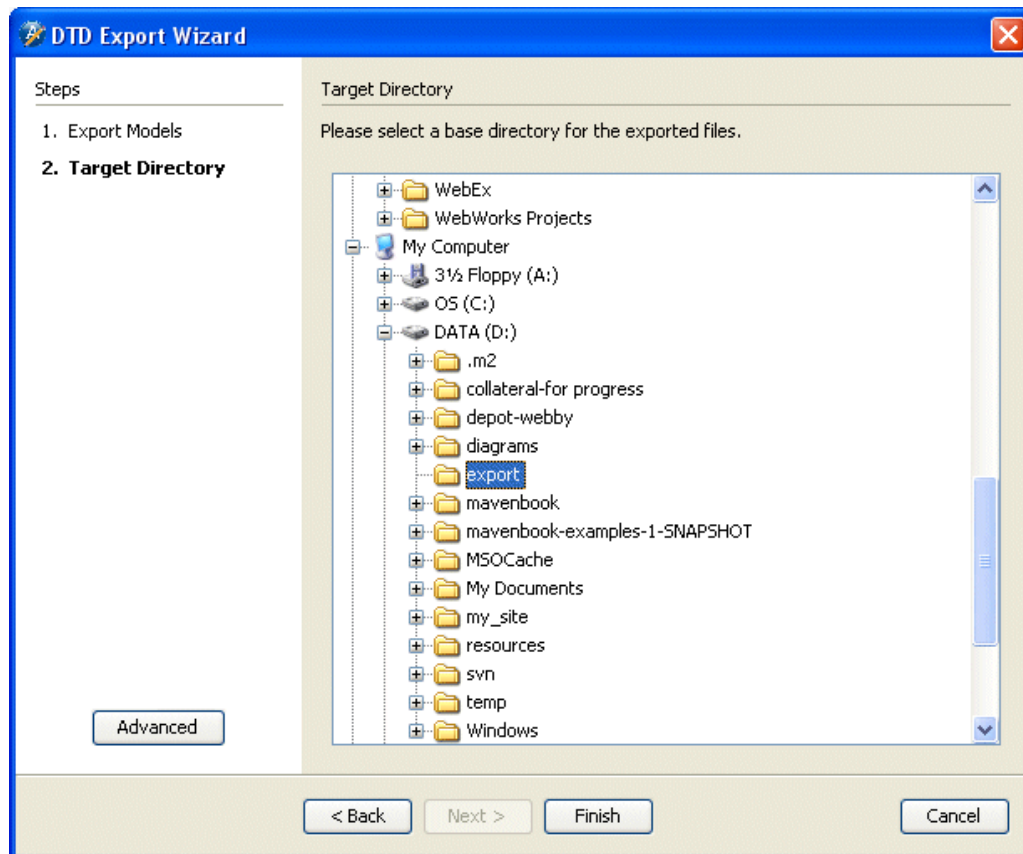
1. Open the DTD Export Wizard in either of the following ways:
 - Right-click the model (.dod) file in the [Project window](#) and select **Export > Export DTD**.

- Click the model (.dod) file in the [Project window](#) and select **Tools > Export > Export DTD** from the menu bar.

This opens the Export Models panel of the DTD Export Wizard.



2. Navigate to the model(s) (.dod) file(s) you wish to export.
3. Click the file you wish to select. To select more than one file, Shift-click each filename.

4. Click **Next**.

5. In the Target Directory panel, browse to the location where you want the new DTD to be stored.
6. Click **Finish** to finish exporting the DTD.

The new DTD is saved to the location you have specified.

Exporting to a RELAX NG (XML) Schema

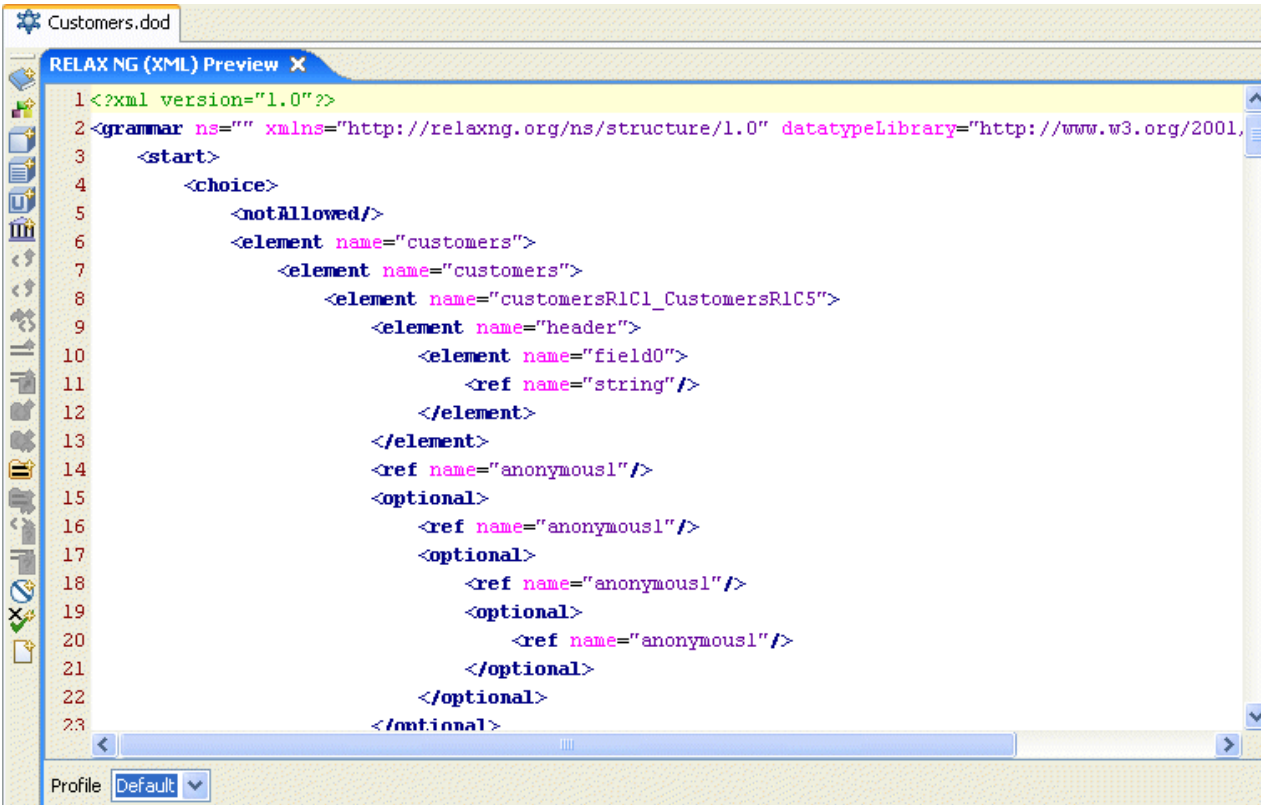
Before you export a data model as a RELAX NG (XML) schema, you can preview it.

Preview

To preview a RELAX NG (XML) schema, do either of the following:

- Right-click the model (.dod) file in the [Project window](#) and select **Preview > Preview RELAX NG (XML)**.
- Alternatively, click the model (.dod) file in the [Project window](#) and select **Tools > Preview > Preview RELAX NG (XML)** from the menu bar.

This opens a RELAX NG (XML) Preview tab similar to the following in the main window of the workbench:



```

1 <?xml version="1.0"?>
2 <grammar ns="" xmlns="http://relaxng.org/ns/structure/1.0" datatypeLibrary="http://www.w3.org/2001,
3   <start>
4     <choice>
5       <notAllowed/>
6       <element name="customers">
7         <element name="customers">
8           <element name="customersR1C1_CustomersR1C5">
9             <element name="header">
10              <element name="field0">
11                <ref name="string"/>
12              </element>
13            </element>
14            <ref name="anonymous1"/>
15            <optional>
16              <ref name="anonymous1"/>
17            </optional>
18            <optional>
19              <ref name="anonymous1"/>
20            </optional>
21            <optional>
22              <ref name="anonymous1"/>
23            </optional>
24          </optional>
25        </element>
26      </choice>
27    </start>
  
```

Profile: Default

Export

To export a model as a RELAX NG (XML) schema:

1. Open the RELAX NG (XML) Export Wizard in either of the following ways:
 - Right-click the model (.dod) file in the [Project window](#) and select **Export > Export RELAX NG (XML)**.

- Click the model (.dod) file in the [Project window](#) and select **Tools > Export > Export RELAX NG (XML)** from the menu bar.

This opens the Export Models panel of the RELAX NG (XML) Export Wizard.

2. Navigate to the model(s) (.dod) file(s) you wish to export.
3. Click the file(s) you wish to select. If you wish to select more than one file, press the Shift key while clicking on each filename.
4. Click **Next**. This opens the Target Directory panel.
5. Navigate to and select the location where you want the new RELAX NG (XML) schema to be stored.
6. Click **Finish** to finish exporting the RELAX NG (XML) schema.

The new RELAX NG (XML) schema is saved to the location you have specified.

Exporting to a RELAX NG (Compact) Schema

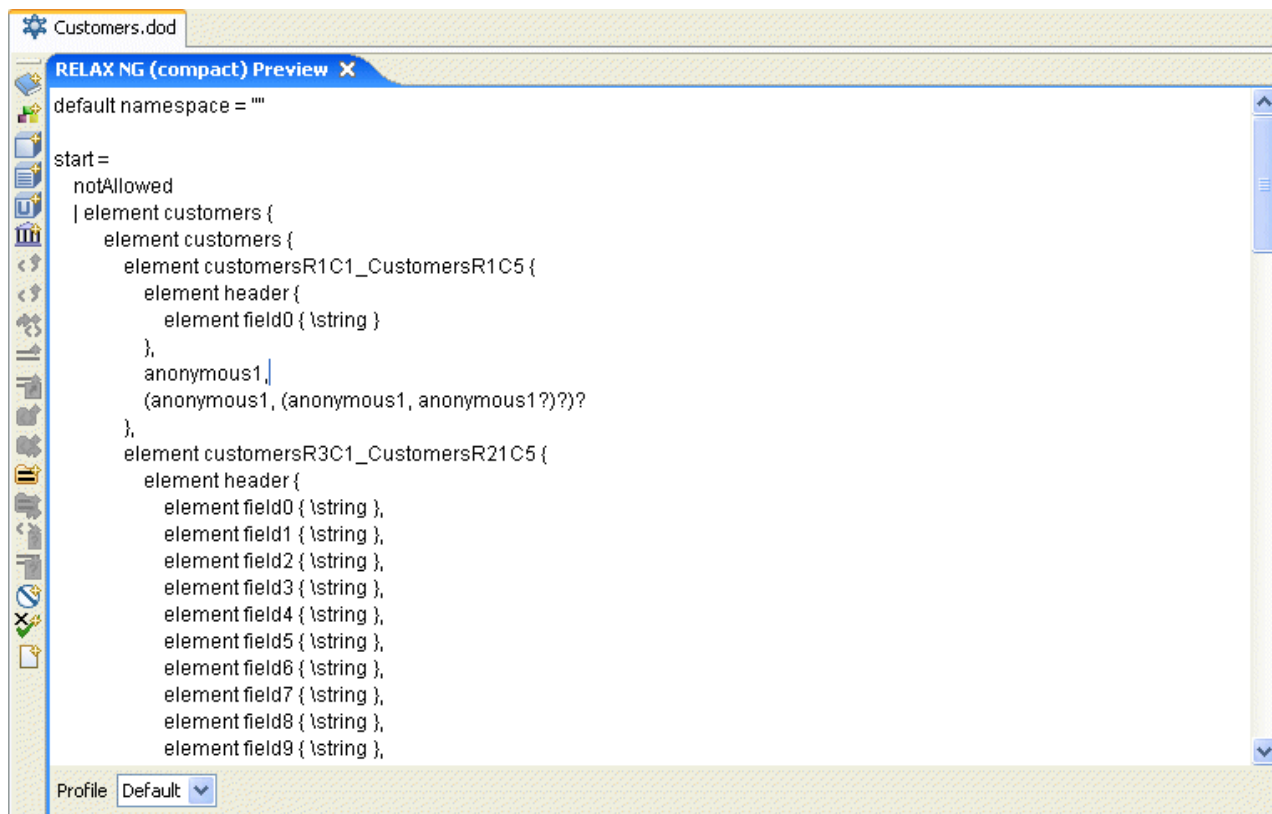
Before you export a data model as a RELAX NG (Compact) schema, you can preview it.

Preview

To preview a RELAX NG (Compact) schema, do either of the following:

- Right-click the model (.dod) file in the [Project window](#) and select **Preview > Preview RELAX NG (Compact)**.
- Alternatively, click the model (.dod) file in the [Project window](#) and select **Tools > Preview > Preview RELAX NG (Compact)** from the menu bar.

This opens a RELAX NG (Compact) Preview tab similar to the following in the main window of the workbench:



Export

To export a model as a RELAX NG (Compact) schema:

1. Open the RELAX NG (Compact) Export Wizard in either of the following ways:
 - Right-click the model (.dod) file in the [Project window](#) and select **Export > Export RELAX NG (Compact)**.
 - Click the model (.dod) file in the [Project window](#) and select **Tools > Export > Export RELAX NG (Compact)** from the menu bar.

This opens the Export Models panel of the RELAX NG (Compact) Export Wizard.

2. Navigate to the model(s) (.dod) file(s) you wish to export.
3. Click the file(s) you wish to select. If you wish to select more than one file, press the Shift key while clicking on each filename.
4. Click **Next**.
5. In the Target Directory panel, browse to the location where you want the new RELAX NG (Compact) schema to be stored.
6. Click **Finish** to finish exporting the RELAX NG (Compact) schema.

The new RELAX NG (Compact) schema is saved to the location you have specified.

Exporting to HTML

ADS enables you to generate an HTML report from a data model, detailing all the model's components, properties, and namespaces. Business analysts can use these reports to confirm that business models and rules are defined appropriately.

Preview

Before you export a data model to HTML, you can preview it, by doing one of the following:

- Right-click the model (.dod) file in the [Project window](#) and select **Preview > Preview HTML**.
- Alternatively, click the model (.dod) file in the [Project window](#) and select **Tools > Preview > Preview HTML** from the menu bar.

This opens an HTML Preview tab similar to the following in the main window of the workbench:

The screenshot shows the HTML Preview window for the 'Accounts.dod' model. The window title is 'Accounts.dod' and the tab is 'HTML Preview'. The main content area displays the title 'Progress Artix Data Services Data Model Documentation - Accounts.dod' and a 'Table of Content' with links to Properties, Namespaces, Global Components, Document Root, Component Definitions, and Glossary. Below the table of content is a 'Properties' section with a table of model properties.

Attribute Form Default	Unqualified
Element Form Default	Qualified
Encoding	UTF-8
Encrypted	no
Hibernate ID Generator Method Default	native
Target Namespace	http://www.progress.com/ArtixDataServices/GettingStarted/Account
Version	0.0.11
Visibility	Public

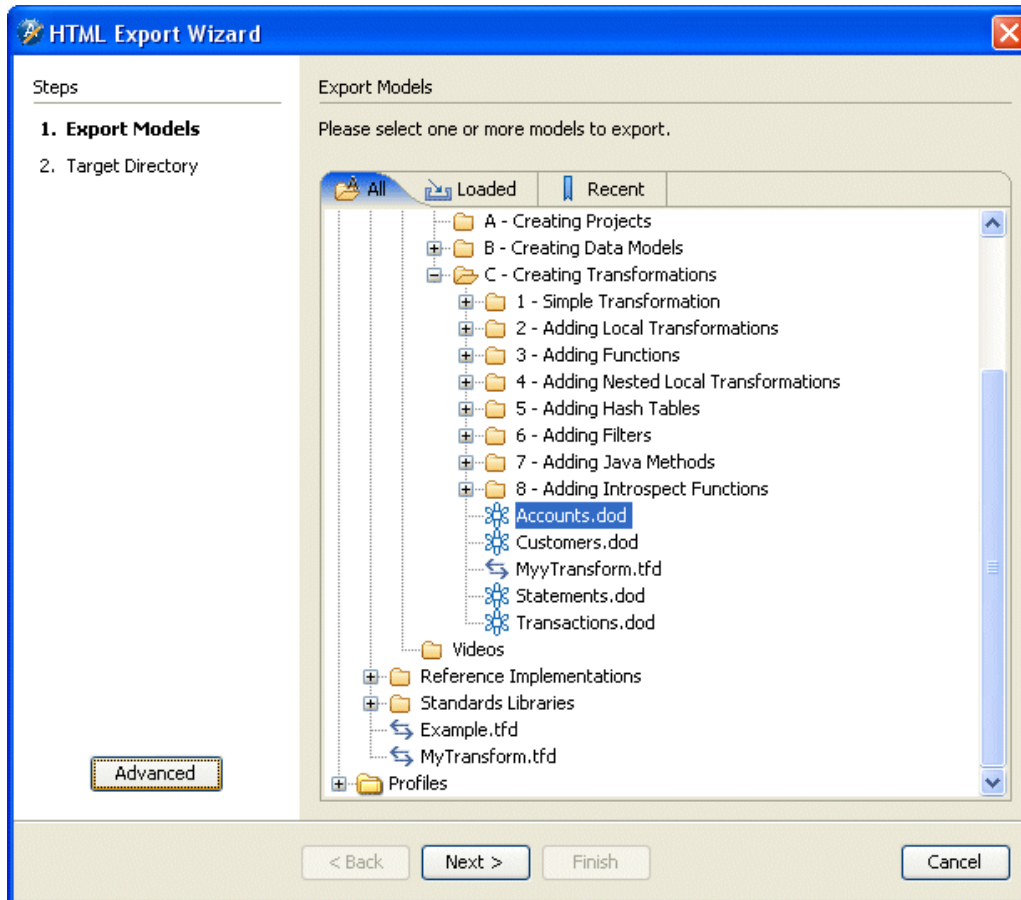
Profile: Default

Export

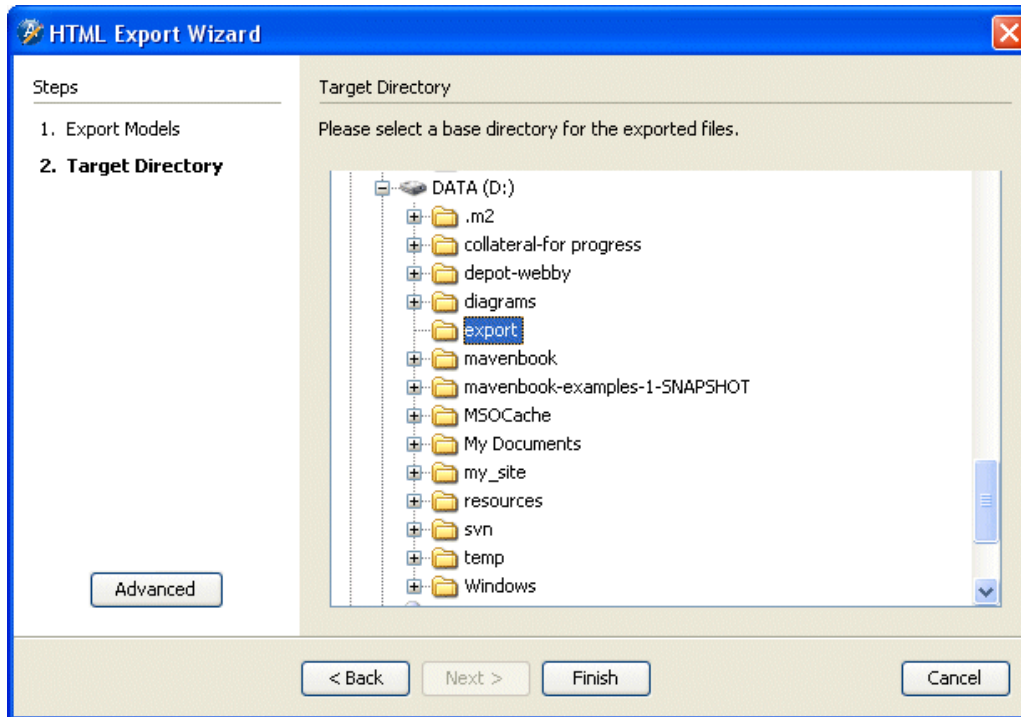
To export a model to HTML:

1. Open the HTML Export Wizard in either of the following ways:
 - Right-click the model (.dod) file in the [Project window](#) and select **Export > Export HTML**.
 - Click the model (.dod) file in the [Project window](#) and select **Tools > Export > Export HTML** from the menu bar.

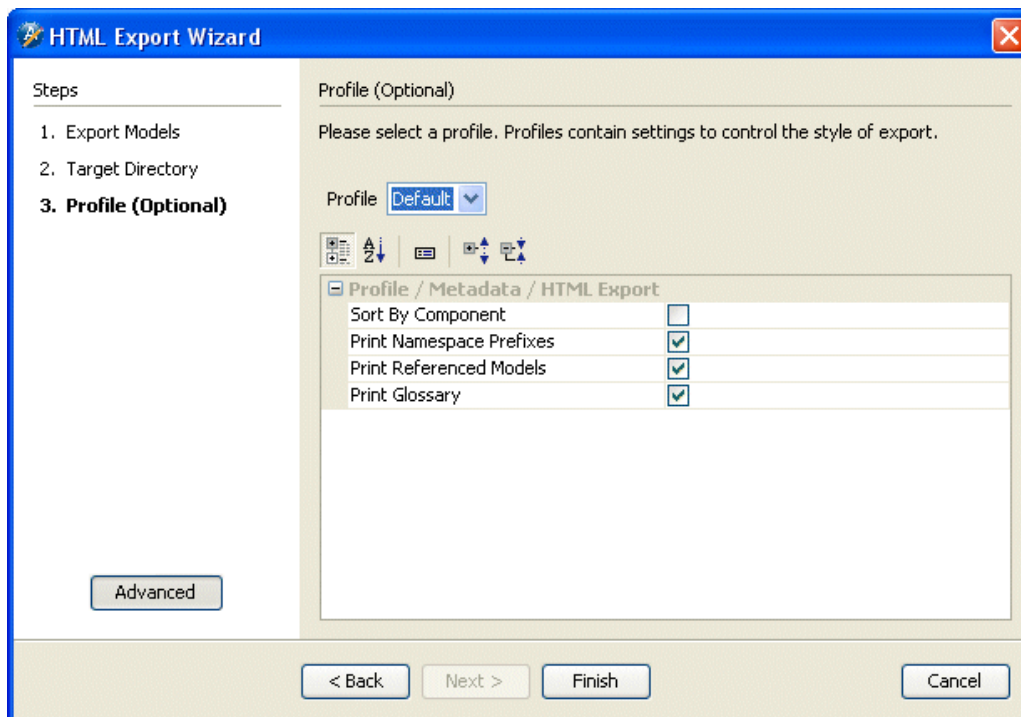
This opens the Export Models panel of the HTML Export Wizard.



2. Navigate to the model(s) (.dod) file(s) you wish to export.
3. Click the file(s) you wish to select. To select more than one file, shift-click each filename.

4. Click **Next**.

5. In the Target Directory panel, browse to the location where you want the new HTML file to be stored.

6. Click the **Advanced** button on the left of the dialog to enable the Profile panel.7. Click **Next**. This opens the Profile panel where you can set various values to determine how the export will be handled.

The default profile settings are: [Sort By Component](#), [Print Namespace Prefixes](#), [Print Referenced Models](#), and [Print Glossary](#).

8. Click **Finish** to finish exporting the HTML file.

The new HTML file is saved to the location you have specified.

Document Roots

If you have various global elements and attributes attached to a data model, document roots allow you to build and parse either type without having to decide which type to use until parse time. In the following screen example, you can see the document root in the Explorer window. This document root is opened in the main tab.

A document root should be present when one or more global elements or attributes are present. It should include all global elements and attributes in the model as well as all models imported, included and redefined.

You cannot edit or delete document roots. If you want to make a change to a global attribute or element, make the change in that attribute or element. This change will then be reflected in the document root.

The screenshot shows the Artix Data Services software interface. The main window is titled "MyProject.iop - [C:\Documents and Settings\ldporter\My ADS Projects] - ...ngs\ldporter\My ADS Projects\Gettin...". The interface includes a menu bar (File, Edit, View, Find, Tools, Deploy, Window, Help), a toolbar, and a main workspace. On the left, there is a "Project Explorer" window showing a tree view of the project structure, including files like "Accounts.txt", "Customers.dod", "Customers.txt", "Statements.dod", "Statements.xsd", "Transactions.dod", "Transactions.txt", "Videos", "Reference Implementations", "Standards Libraries", "Example.tfd", "MyTransform.tfd", and "Profiles". Below the project explorer is an "Explorer" window showing a tree view of the "Native" namespace, including "Transactions.dod", "File", "Records", "rowCheckRule", "Document Root", and "Document Root (local)". The main workspace displays a "Document Root (Read Only)" window with a table of components. The table has columns for Component, Type, Cardinality, and Size. The components listed are: Document Root (Document Root (local), 49 - *), Transactions (Transactions, 1, 49 - *), Header (Header, 1, 19), Name (Name (local), 1, 1), Card Number (Card Number (local), 1, 1), Expiry Date (Expiry Date (local), 1, 1), Amount (Amount (local), 1, 1), Currency (Currency (local), 1, 1), Transaction Date (Transaction Date (local), 1, 1), Commission (Commission (local), 1, 1), Vendor ID (Vendor ID (local), 1, 1), Country (Country (local), 1, 1), Customer Details (Customer Details, 1..*, 19), and Row Count (Row Count, 1, 11 - 12). The status bar at the bottom shows "Ready" and "117M of 170M".

Component	Type	Cardinality	Size
Document Root	Document Root (local)		49 - *
Transactions	Transactions	1	49 - *
Header	Header	1	19
Name	Name (local)	1	1
Card Number	Card Number (local)	1	1
Expiry Date	Expiry Date (local)	1	1
Amount	Amount (local)	1	1
Currency	Currency (local)	1	1
Transaction Date	Transaction Date (local)	1	1
Commission	Commission (local)	1	1
Vendor ID	Vendor ID (local)	1	1
Country	Country (local)	1	1
Customer Details	Customer Details	1..*	19
Row Count	Row Count	1	11 - 12

Within a transformation, you can map directly from document root to document root.

Note: Where there is a 'simple typed' element in a data model, the Run Wizard allows you to parse XML files whose root element is simple typed. It is possible to store the comments in an XML file which appear outside of the root element.

Excluding Global Elements from a Document Root

There may be times when you do not want to incur the overhead of parsing all global elements in a document root, for example, where a data model references elements from other models. ADS Designer allows you to exclude selected global elements from the document root.

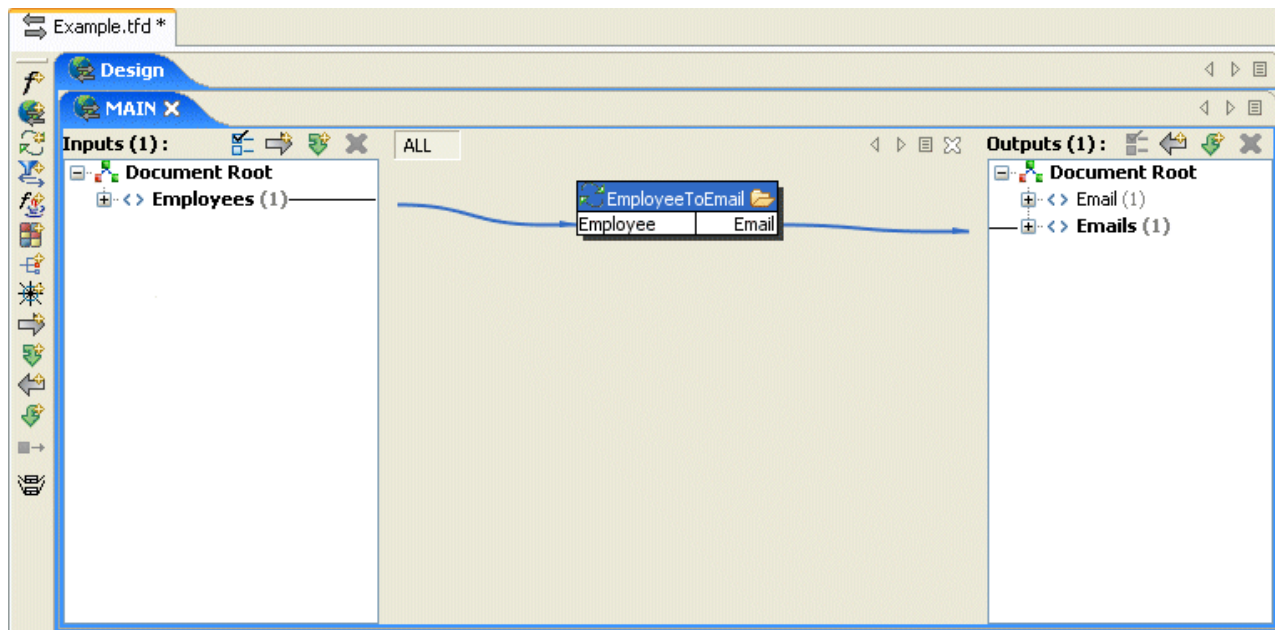
To hide referenced elements, right-click the document root and select **Toggle Hiding Referenced Elements**.

Transformations

Transformations consist of at least two data models that represent input and output data. They allow you to read data into an input model and map this data to elements of an output model, for the purposes of transforming your data in some way.

Transformations can consist of multiple input and output models. They are built up from functions that are chained together to convert one or more values from the input models to a node in one of the output models.

The following is an example of a simple transformation.

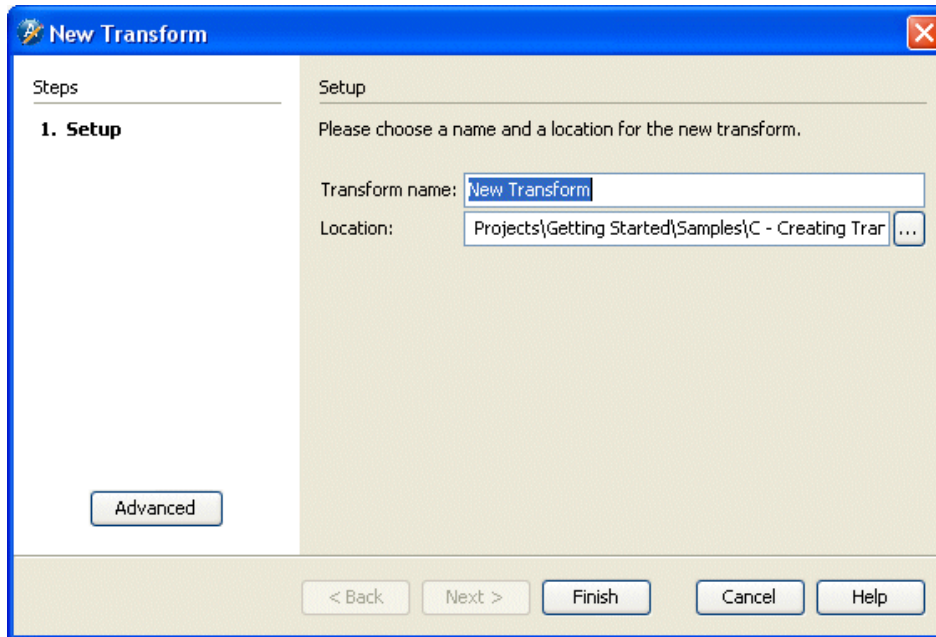


- [Creating a Transformation](#)
- [Adding Models to a Transformation](#)
- [Adding Components to a Transformation](#)
- [Setting up Translations](#)
- [Using Curved Connectors](#)
- [Looking for Unmapped Components](#)
- [Automatically Aligning a Transformation](#)
- [Searching for Components in a Transformation](#)
- [Globalizing Local Transformations](#)
- [Moving Mappings](#)
- [Exporting HTML from a Transformation](#)
- [Collaborating on Transformations](#)

Creating a Transformation

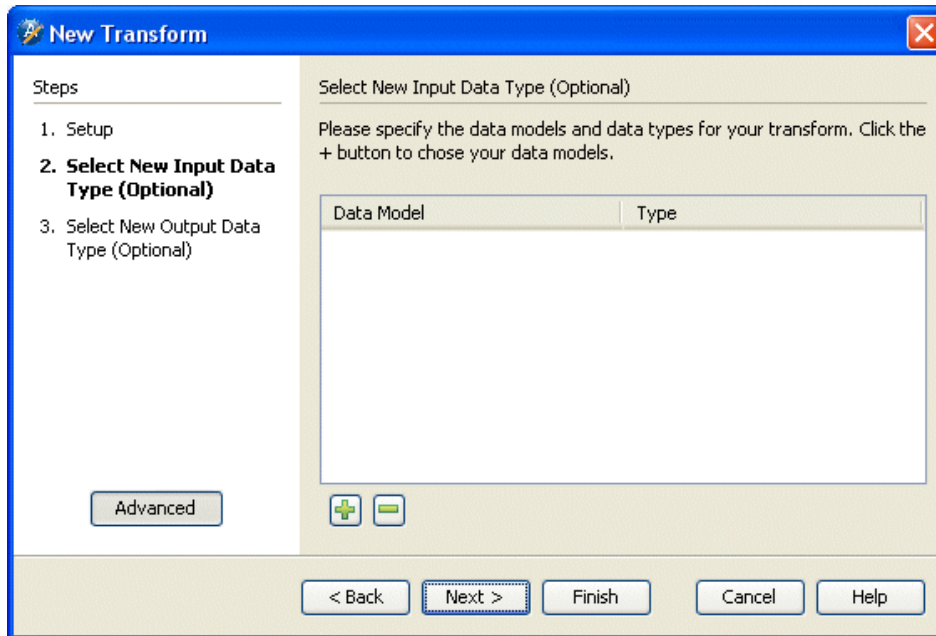
To create a transformation:


1. Ensure that your project is open in the Project window of the workbench.
2. In the Project window, navigate to the folder where you wish to store the transformation, right-click and select **New > Transform**. Alternatively, select **File > New > Transform** from the menu bar. This opens the Setup panel of the New Transform wizard.

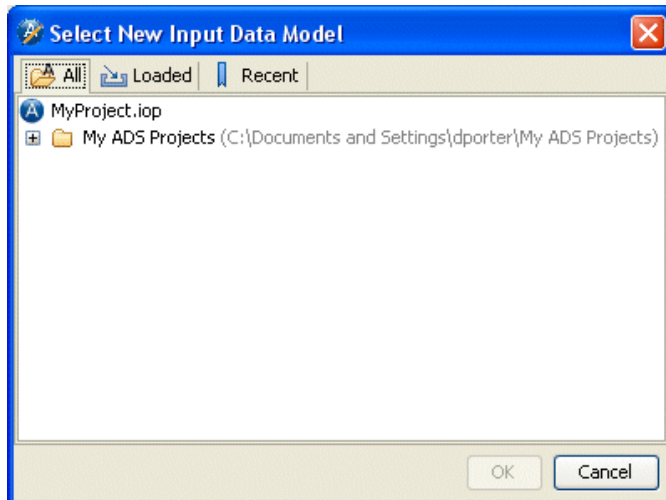


3. Type the name you want to assign to your transformation in the Transform name field.
4. If you do not wish to accept the default location, click the button beside the Location field to select an alternative location for your transformation.
5. If you do not wish to add input and output models to your transformation, click **Finish**. If you wish to add input and output models to your transformation, click the **Advanced** button.

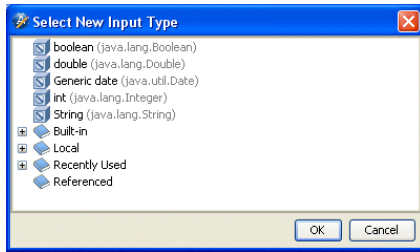
- Click **Next**. This opens the Select New Input Data Type panel.






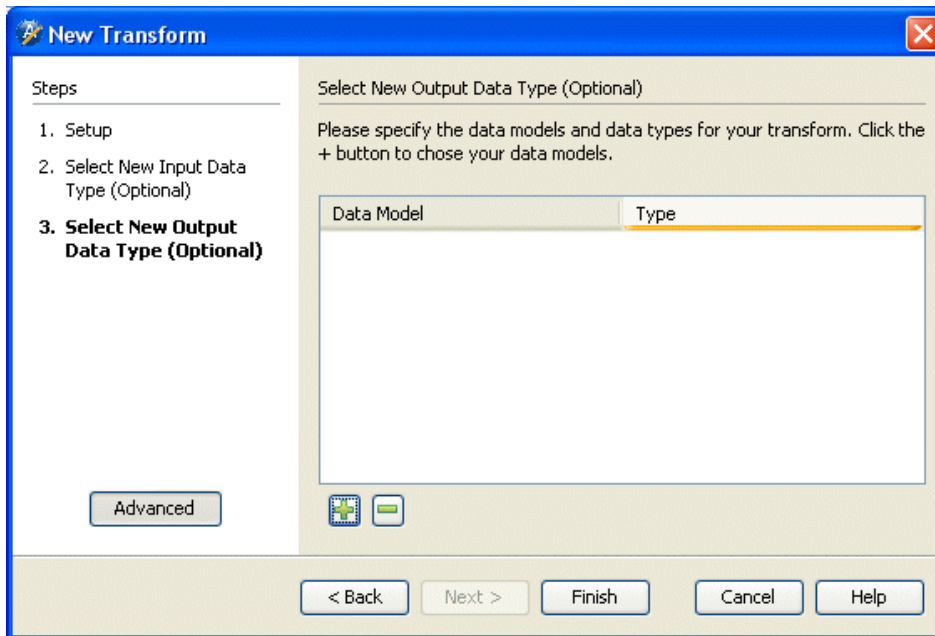
- Click the  icon to add an input model to your transformation. This opens the Select New Input Data Model dialog.




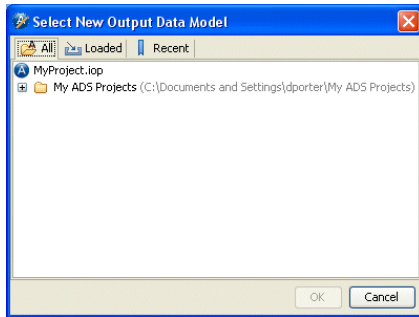
- Navigate to the data model you wish to use as an input model and click **OK**. This opens the Select New Input Type dialog.



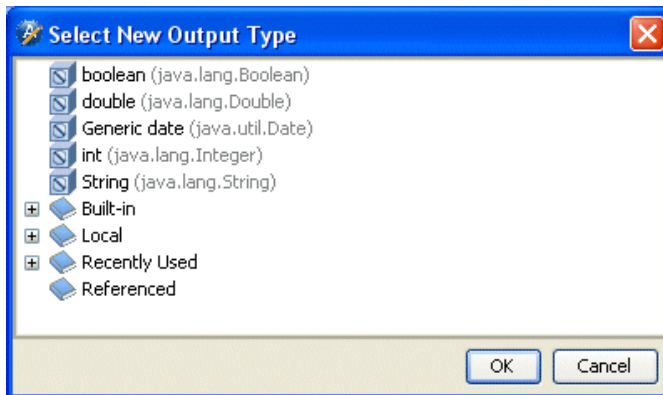
9. Select the input type you wish to use and click **OK**. The selected model and type are then added to the Select New Input Data Type panel.
10. If you wish to add more input models, repeat steps 7-9 as appropriate.
Note: You may add as many input models as you wish to a transformation. To subsequently remove an input model, click that model in the list and then click the  icon. Click the  and  icons to move models up and down the list. This will determine the order in which models are displayed in the transformation tab.
11. Click **Next**. This opens the Select New Output Data Type panel.






12. Click the  icon to add an output model to your transformation. This opens the Select New Output Data Model dialog.

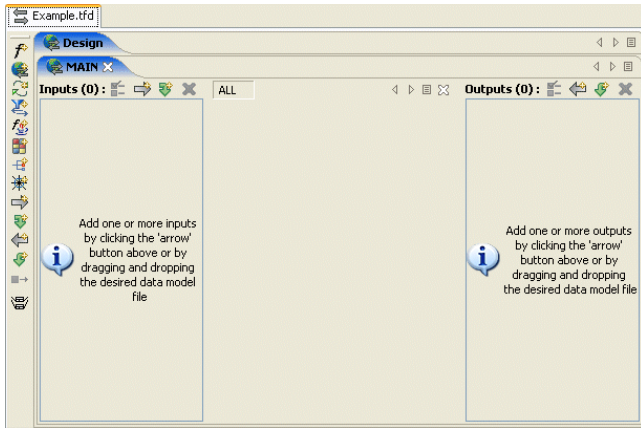


13. Navigate to the data model you wish to use as an output model and click **OK**. This opens the Select New Output Type dialog.



14. Select the output type you wish to use and click **OK**. The selected model and type are then added to the Select New Output Data Type panel.
15. If you wish to add more output models, repeat steps 12-14 as appropriate.
Note: You may add as many output models as you wish to a transformation. To subsequently remove an output model, click that model in the list and then click the  icon. Click the  and  icons to move models up and down the list. This will determine the order in which models are displayed in the transformation tab.
16. Click **Finish** to finish creating the transformation. At this point, a tab is opened for the new transformation.

The following screen example shows an empty transformation tab (that is, with no input or output models added). If you have already added models to the transformation, they will be displayed accordingly in the Inputs and Outputs sections.




The target namespace of a transformation is set in exactly the same way as it is for a [data model](#) and is used to determine the package into which the transformation should be deployed.

Note: Transformations are loosely analogous to XSLT and are saved as .tfd files.


Adding Models to a Transformation

The transformation [tab](#) contains three panels. The panel on the left contains the input data model you are mapping from. The panel on the right contains the output model you are mapping to. The central area is where rules are defined.

To add an input model

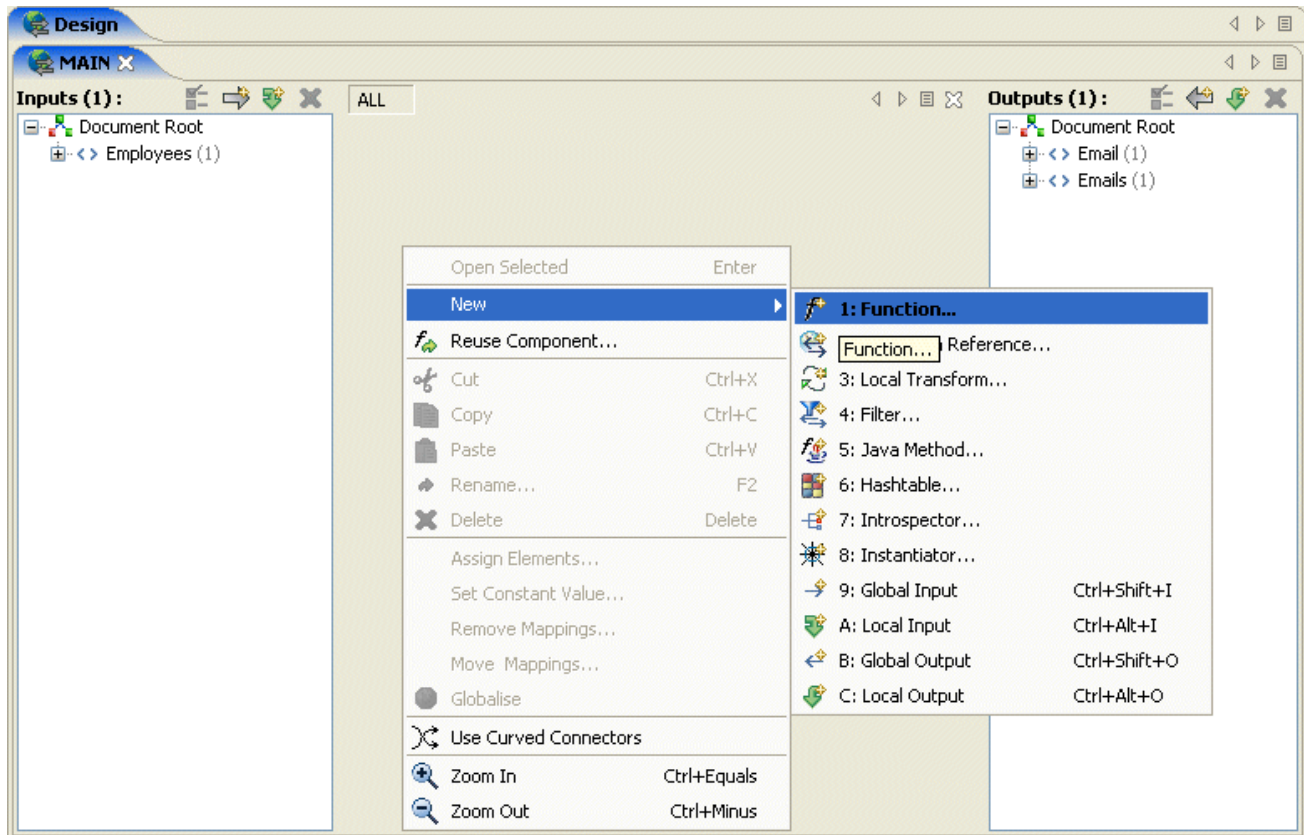
1. Click the  icon on the Inputs panel and select the model. You are prompted for an element.
2. Select the root of the data structure that you want to transform from. You can add multiple input models to a transformation.

To add an output model:

1. Click the  icon on the Outputs panel and select the model. You are prompted for an element
2. Select the root of the data structure that you want to transform to. You can add multiple output models to a transformation.

Adding Components to a Transformation

When you have added an input and an output model, start building up your rules. Work down the output model nodes and consider how you are going to map to each mandatory object. When you have decided what mappings are needed, right-click in the ALL panel (the central blank panel) and select **New** to see a list of components that you can add to your transformation to make it functional:



The components available are:


- [Function](#)
- [Transform Reference](#)
- [Local Transform](#)
- [Filter](#)
- [Java Method](#)
- [Hashtable](#)
- [Introspector](#)
- [Instantiator](#)
- Global Input
- Local Input
- Global Output
- Local Output

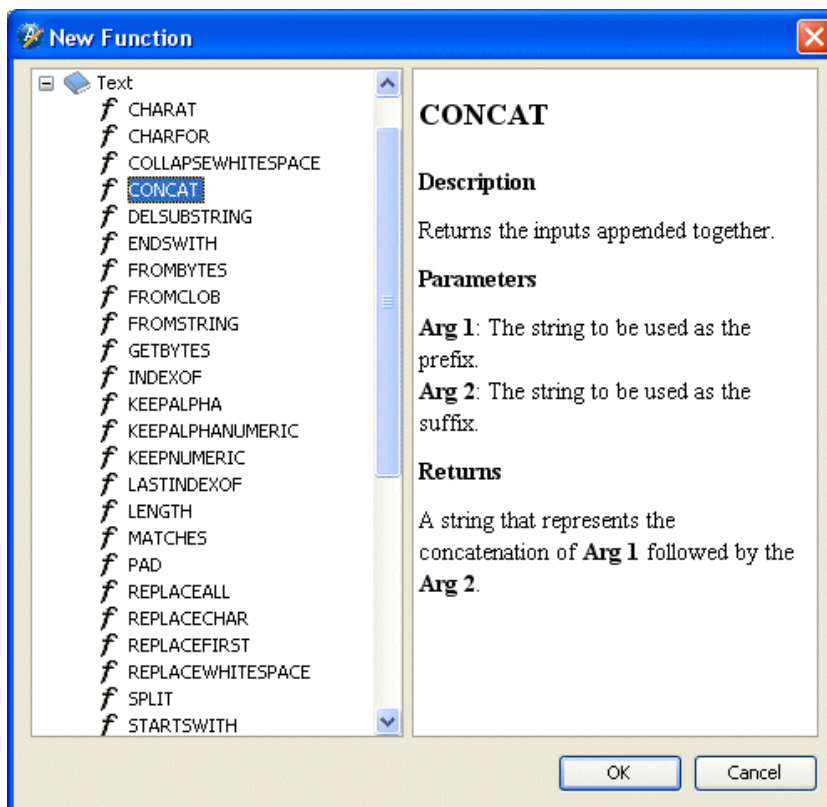
Adding Functions to a Transform

ADS provides a series of off-the-shelf functions that you may use within your transformations, as appropriate. Transformations are built up from functions that are chained together to convert one or more values in the input model(s) to relevant nodes in the output model(s).

If, for example, a data model that contains address lines has been broken into two different address lines (that is, address lines 1 and 2), but the output model only has one address node, you can use the CONCAT function to concatenate the two address lines into one.

To add a function to a transformation:

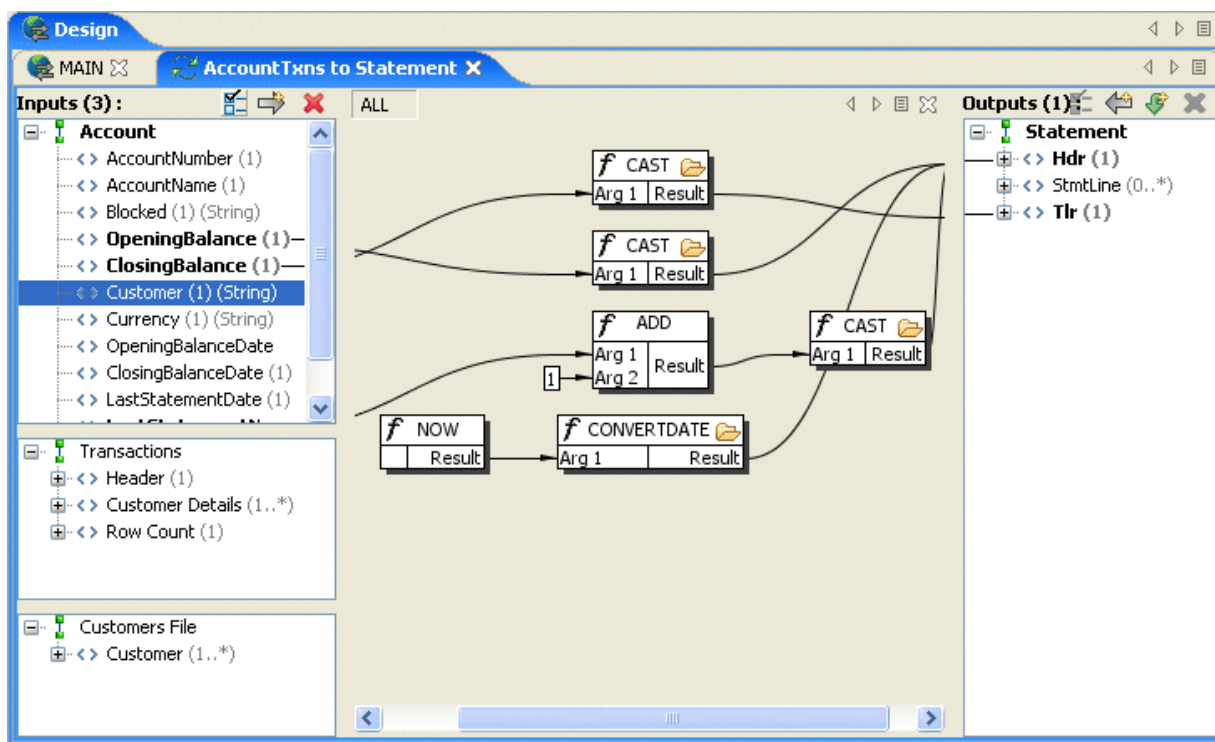
1. Open the New Function dialog in either of the following ways:
 - Right-click in the ALL section of the transformation and select **New > Function**.
 - Click  on the Transformation palette.



Note: A context-sensitive description of the currently selected function is displayed on the right of the New Function dialog.

2. Select the relevant function and click **OK**. The selected function is added to the ALL section of the transformation.
3. Set up mappings between the function parameters and the input and output models or other functions, as appropriate.


The following screen shows a series of functions that have been added to a transformation and where the function parameters are mapping either to model elements or to other function parameters.

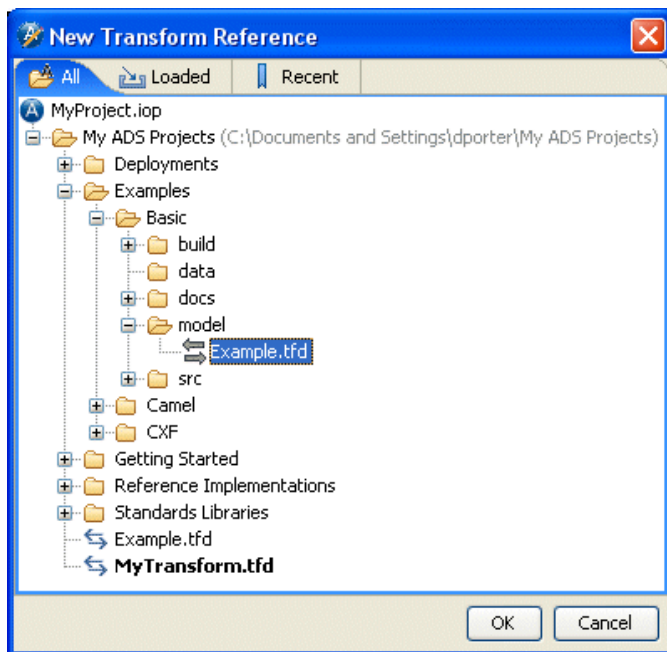


Adding a Transformation Reference

A transformation can reference another (global) transformation, to avail of the functionality provided by the referenced transformation.

To add a transformation reference to a transformation:

1. Open the New Transform Reference dialog in either of the following ways:
 - Right-click in the ALL section of the transformation and select **New > Transform Reference**.
 - Click  on the Transformation palette.
2. Browse to an existing transformation that you want to reference within the currently open transformation. The selected transformation is added to the ALL section of the currently open transformation.




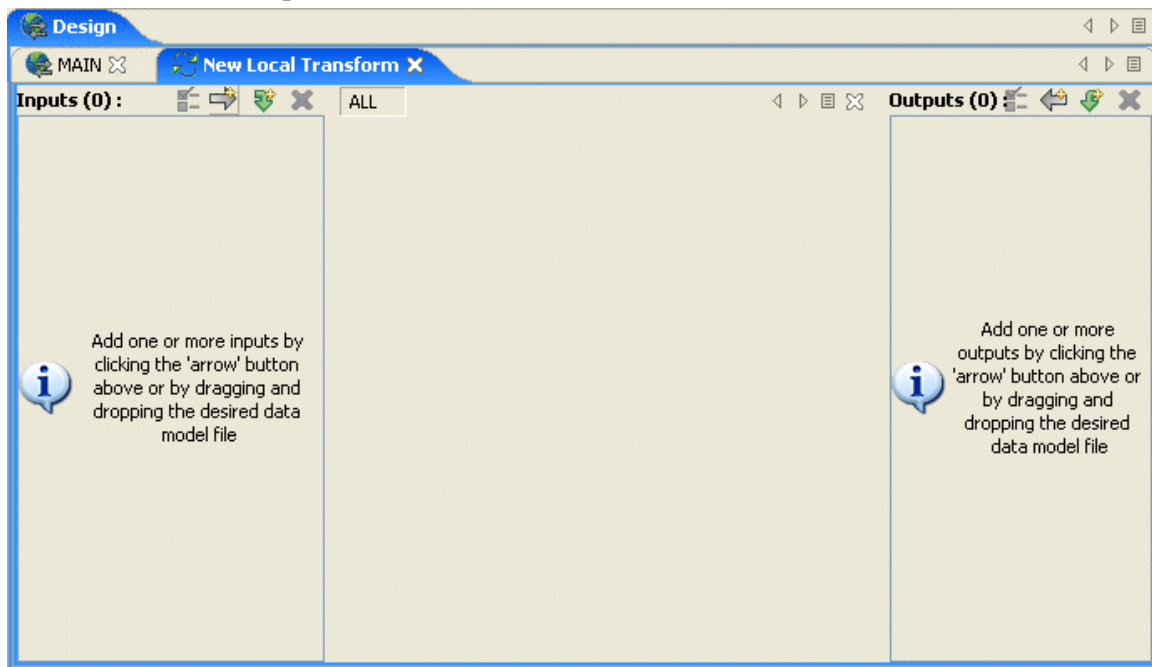
3. [Set up mappings](#) as appropriate for the referenced transformation to work with.



Adding a Local Transformation

A local transformation is created within another transformation. Local transformations are used to represent an individual operation and encapsulate functionality that can be reused within the parent 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. You can nest as many transformations within each other as required.

To add a local transformation:

1. Open the New Local Transform dialog in either of the following ways:
 - Right-click in the ALL section of the transformation and select **New > Local Transform**.
 - Click  on the palette.
2. In the New Local Transform dialog, type the name you want to assign to the local transformation and click **OK**. The new local transformation is added to the ALL section of its parent transformation. The new local transformation is also opened in its own tab.



3. Click the  and  icons in the Inputs and Outputs sections respectively to add local input and output models.


Note: A local transformation works in exactly the same way as a normal transformation. You set up mappings between local input and output models in a local transformation in the same way as mappings between global input and output models in a global transformation.

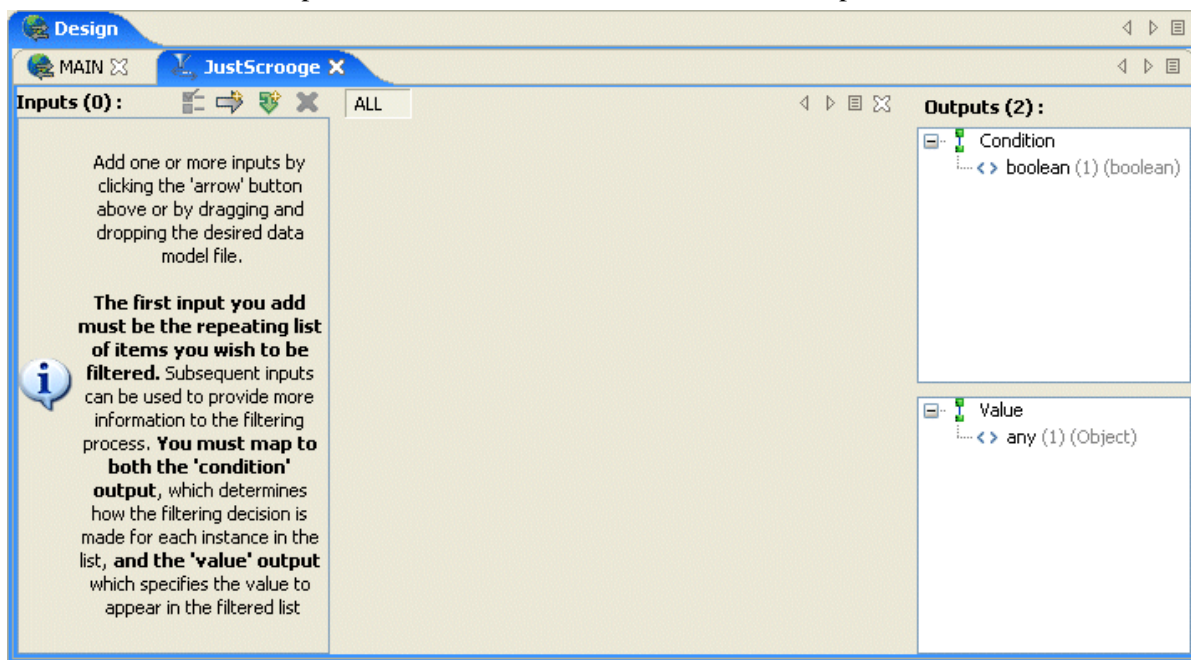
Adding Filters to a Transform

You can use filters to create mappings for recurring elements, so that only a subset of a group of recurring elements is returned as part of the transformation. The filter defines the logic for deciding what is included in the subset returned by the filtering function.


For example, you might have a Record element, which is a recurring element, that contains dates and string fields. You might want to use a filter to return a collection of all dates where the string field's length is equal to 5. This collection (the output from the filter function) can then be used as output for an element which might be a collection of dates.

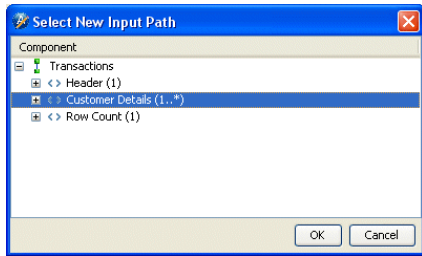
To add a filter to a transformation:

1. Do one of the following:
 - Right-click in the ALL section of the transformation and select **New > Filter**.
 - Click  on the Transformation palette.
2. In the New Filter dialog, type the name you want to assign to the filter and click **OK**. The new filter is added to the ALL section of its parent transformation. The new filter is also opened in its own tab.

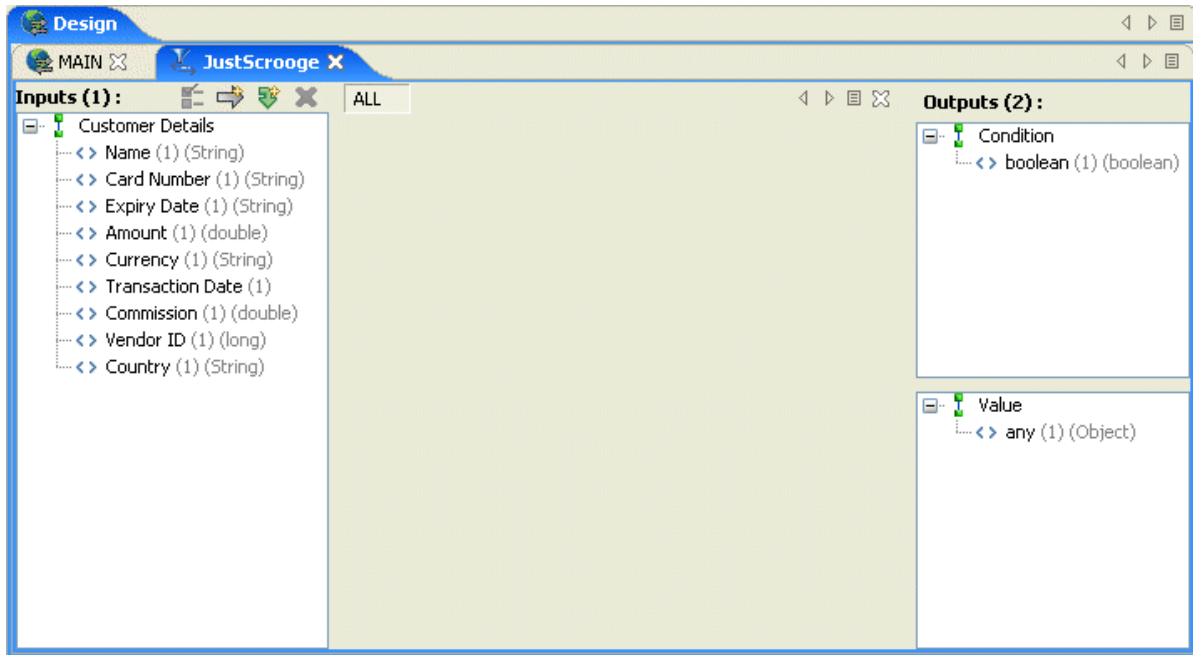


The Inputs section of the tab expects a data model which the filter logic can operate on. The Outputs section of the tab is divided in two: the top section is the boolean logic section, and the bottom section specifies what the output should be.

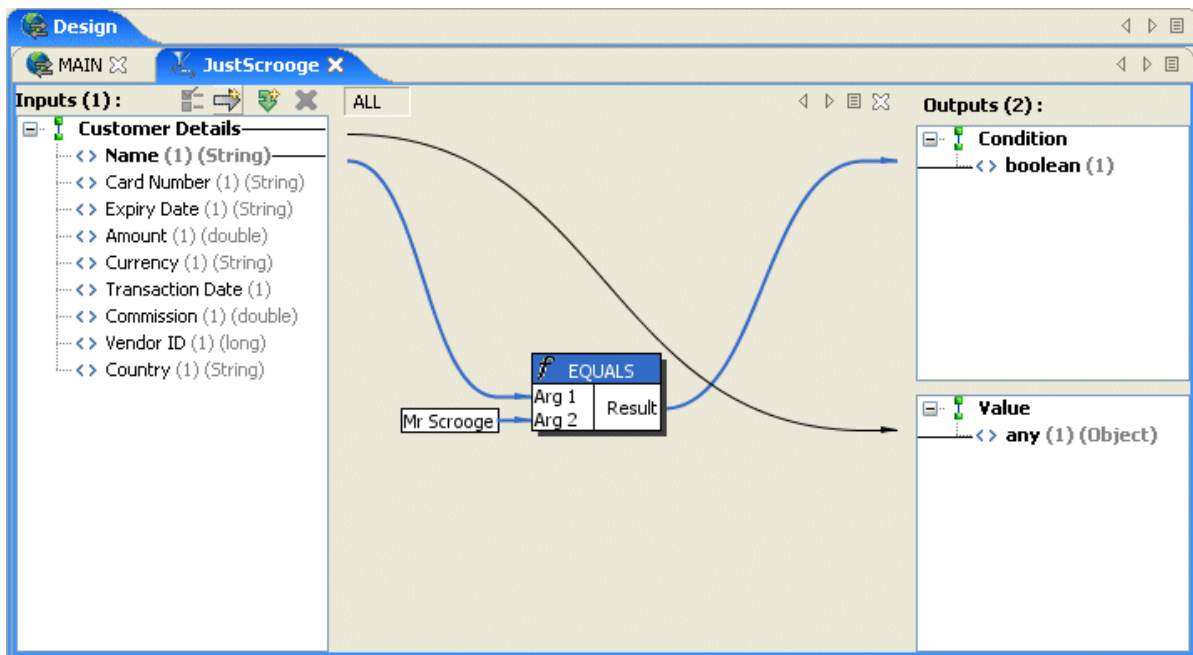
3. Click the  icon in the Inputs section. This opens the Add input dialog with a list of existing input models.
4. Select the input model you want for the filter and click **OK**. Depending on the model you select, this in turn might open a Select New Input Path dialog.



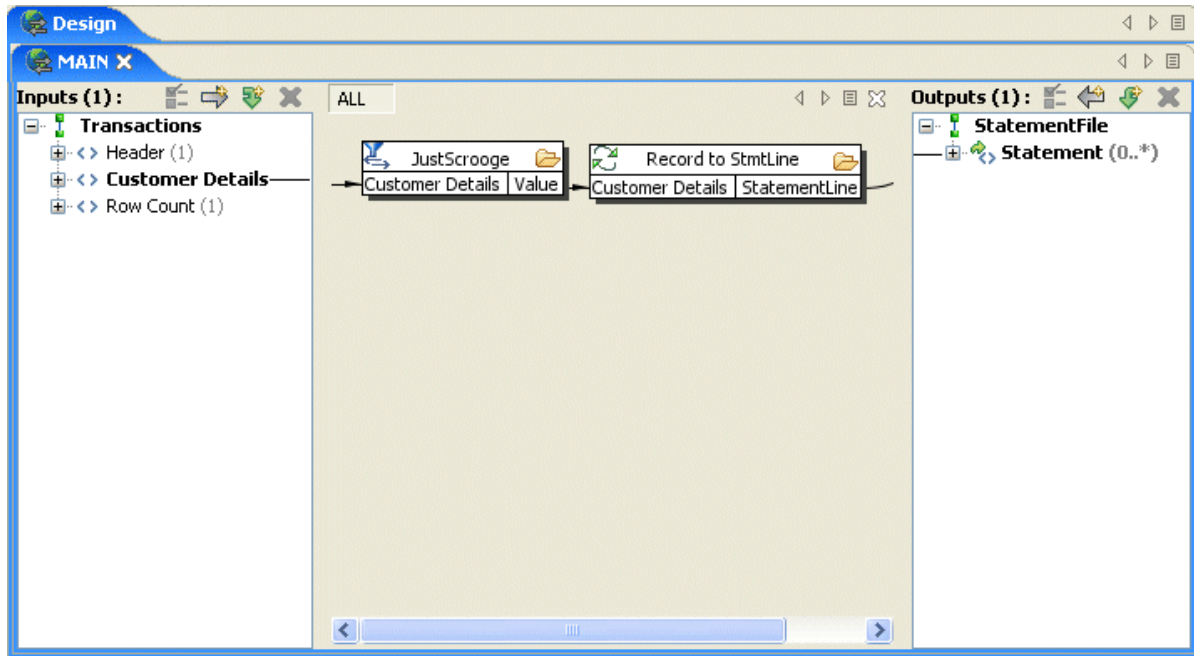
5. Select the relevant complex type and click **OK**. The selected input is added to the Inputs section in the filter's tab.



6. You can now add functions and mappings in the filter's tab.




The filter is now ready for use in the Main transformation tab as shown:

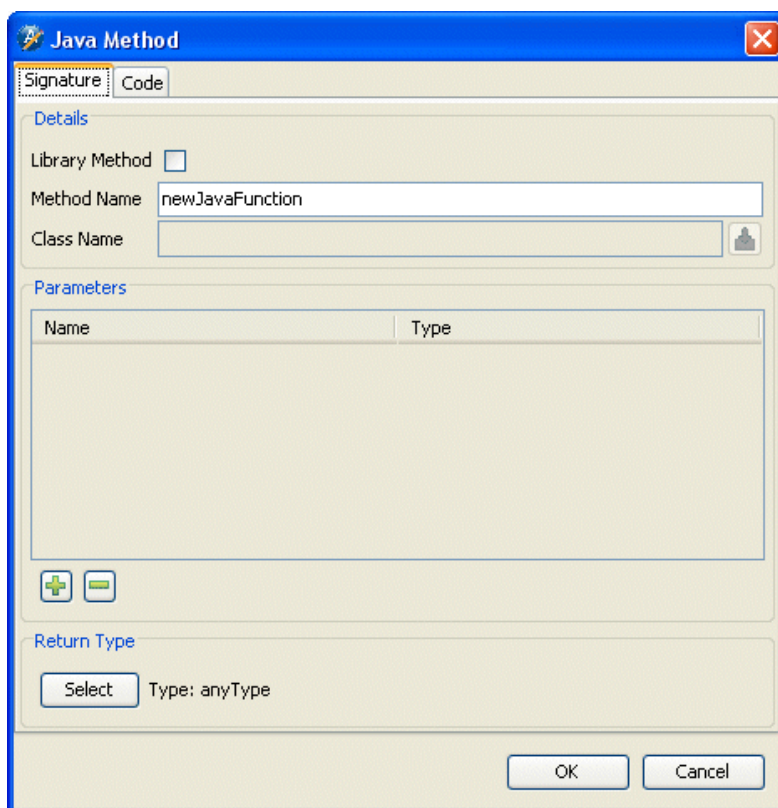



Adding Java Methods to a Transform

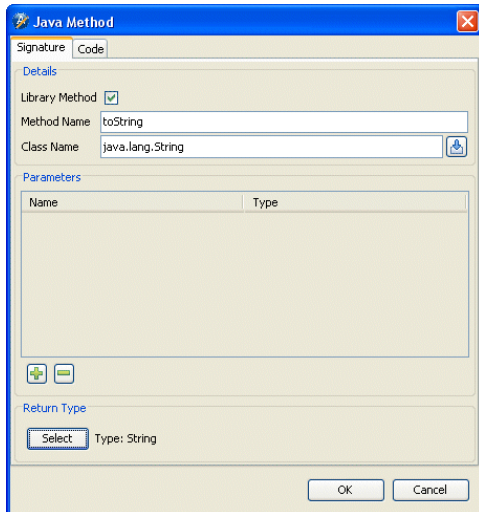
The new Java method function allows you to either reference a static method in some Java class on the Designer classpath or to write a new Java method from scratch that will be embedded in the class that represents the transformation at build time. The following example shows how to use a Java method to look up a transaction from a vendor and then assign it to a vendorID.


To add a Java method:

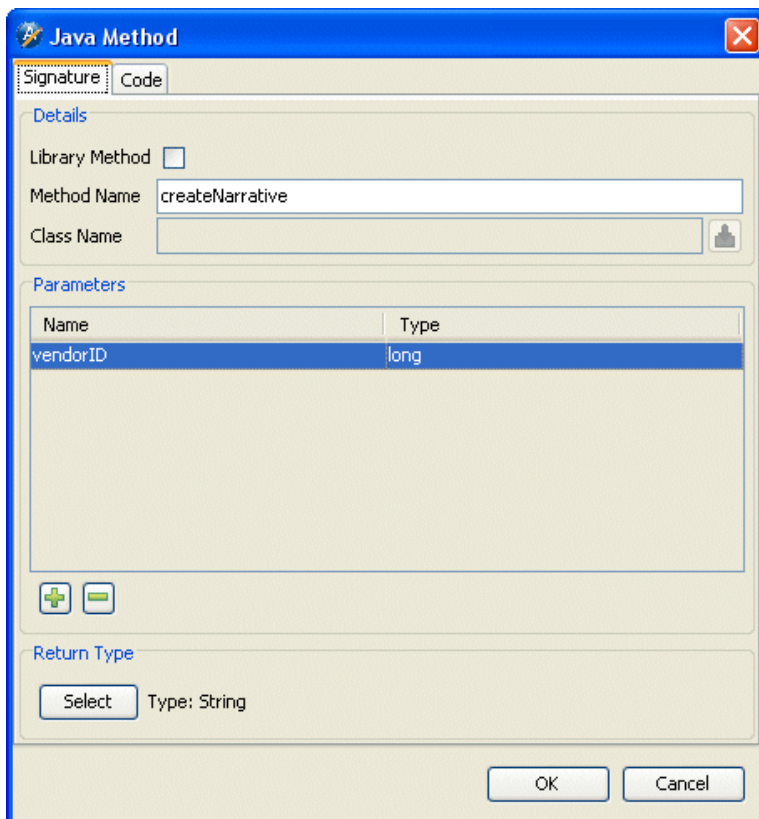
1. Open the Java Method dialog in either of the following ways:
 - Right-click in the ALL section of the transformation and select **New > Java Method**.
 - Click  on the palette.



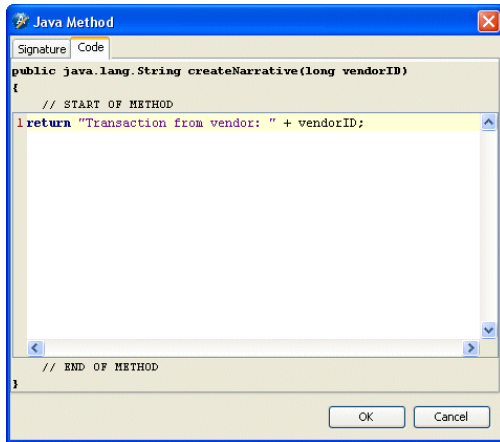
2. Ensure that the Signature tab is selected. The Signature tab can be used in two ways, as follows:
 - Reference a static method on an already existing class.
 - Check the Library Method checkbox and type the name of the class and method in the relevant fields. Finally, click the  icon to load the specified class and method.



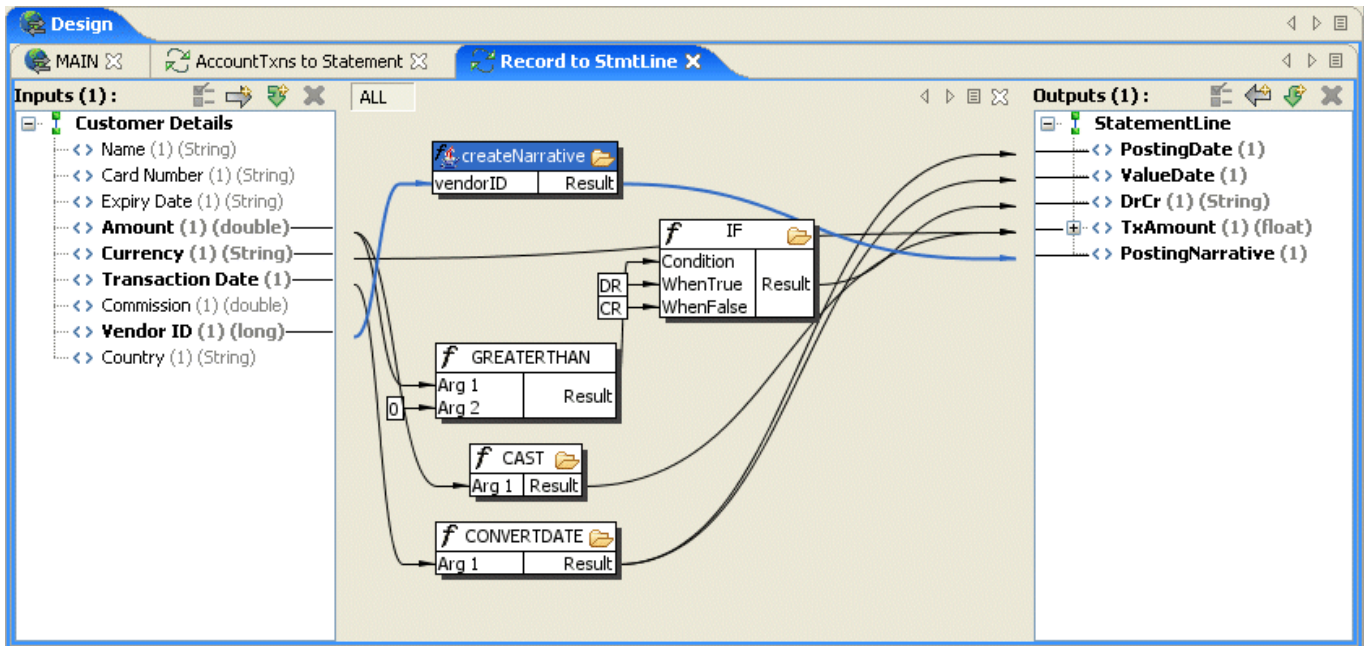
Type the name of the method in the Method Name field, click the  icon to add a new parameter row and set a name and type for this parameter, and click the **Select** button to select a return type. For example, the following screen specifies a method called CreateNarrative with a vendorID parameter of type long that returns a string:



3. After populating the relevant fields in the Signature tab, the outline for the Java method is created and the actual implementation of the method can be written in the Code tab.




- click **OK** to save your changes. The new Java method is added to the ALL section of its parent transformation where mappings can be set up for it, as appropriate. For example:

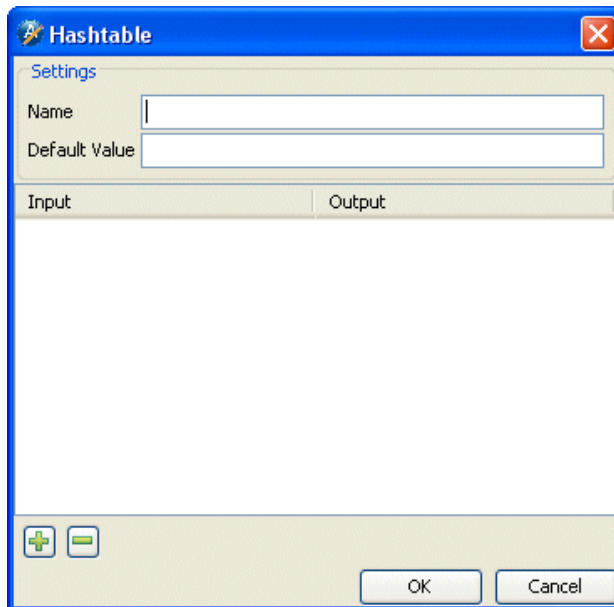




Adding Hashtable Functions to a Transform

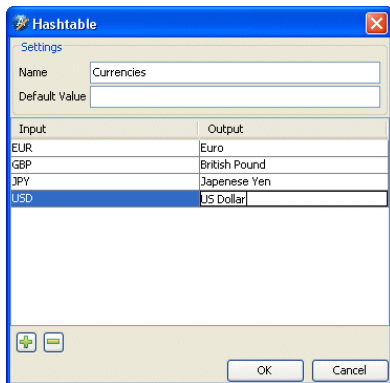
The Hashtable function allows you to create a hash table of values that can be referenced by the transformation code. This is useful in cases where you want an input string value (for example, "USD") to act as the key to an output string value (for example, "US Dollar"), so the hash table operates as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

To add a hash table:

1. Open the Hashtable dialog in either of the following ways:
 - Right-click in the ALL section of the transformation and select **New > Hashtable**.
 - Click  on the palette.
2. Type the name you want to assign to the hash table in the **Name** field.

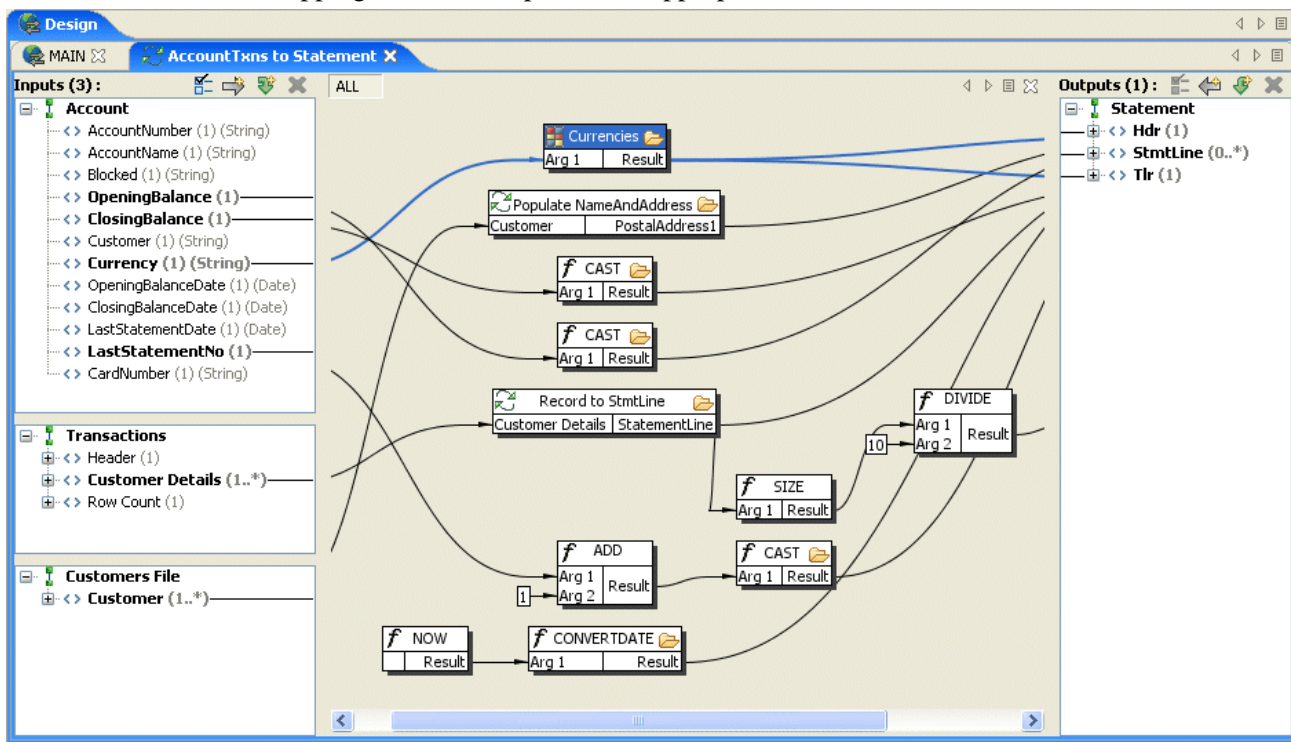


3. Click the  icon to add a new row and type a value in both the Input and Output column. Click the  icon again to add further rows.



Note: To delete a row, select it and click the  icon.


- When you have finished adding rows, click **OK**. The new hash table is added to the ALL section of its parent transformation where mappings can be set up for it, as appropriate.



Adding Introspect Functions to a Transform

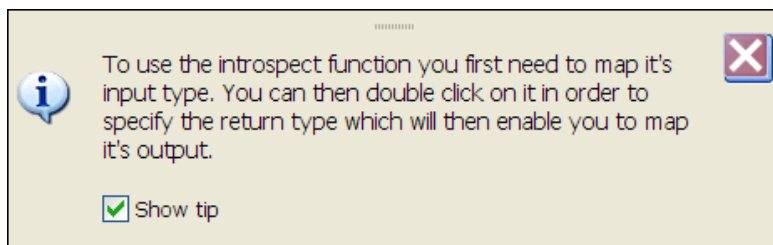
The Introspector function allows you to specify that the value from a specific descendant of a given data type should be mapped to your output data model. In other words, the introspector allow you to return a value of the part of a complex type value which can then be mapped to an output model. For example, you can use an introspect function to extract a country of residence from a Customer model and concatenate it with an account number to identify the location of a customer's account.

To add an introspector:

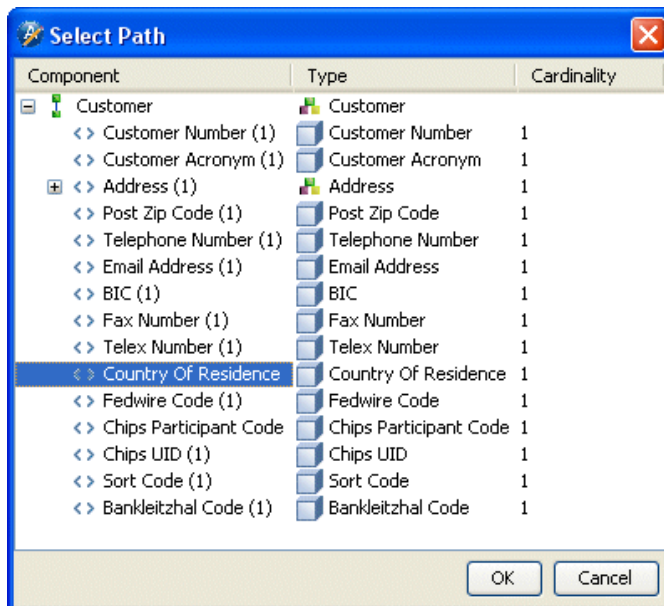
1. Do either of the following:
 - Right-click in the ALL section of the transformation and select **New > Introspector**.
 - Click  on the palette.

The new introspector is added to the ALL section of its parent transformation where mappings can be set up for it, as appropriate.

Note: If tooltips are enabled, the following tooltip is displayed when you go to add an introspector:

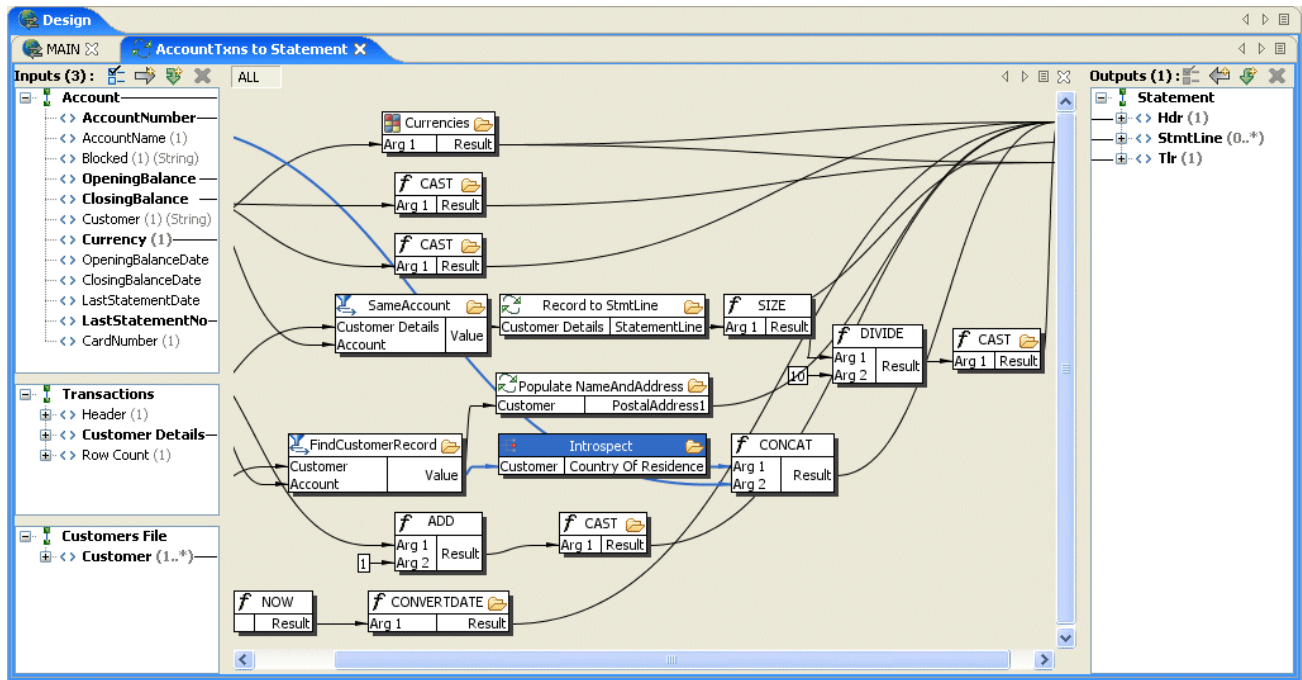


2. Map an element in the input model to the input parameter (Arg1) for the introspector, as appropriate.
3. Double-click **Arg1** in the introspector. This opens the Select Path dialog displaying the selected input type. For example:



4. Select the relevant input type and click **OK**. The name of the selected type is then displayed as the input argument for the introspector.

5. Double-click **Result** in the introspector. This reopens the Select Path dialog.
6. Select the relevant result type and click **OK**. The name of the selected type is then displayed as the result for the introspector.
7. Add any other functions, filters, and so on that you might want to connect to the introspector. The following example shows an Introspect function that takes the value of a FindCustomerRecord filter as its input, and maps its result to the input argument of a CONCAT function.




Adding Instantiate Functions to a Transform

The Instantiate function instantiates an object in the output data model, depending on whether some condition is true. For example, you might have an optional Dates field in your output model (with a cardinality of 0..1) and you want to ensure that it is instantiated if there is an optional Field98a2 field (also with that cardinality of 0..1) in your input model.

To add an instantiator:

1. Do either of the following:

- Right-click in the ALL section of the transformation and select New > Instantiator.
- Select  from the toolbar.

This opens the Select Path dialog showing the elements of the output model.

2. Select the relevant field in the output model that you want to be instantiated and click **OK**. The new instantiator is added to the ALL section of its parent transformation.

Note: In this case, the name of the selected output type is displayed as the result of the instantiator, and it is mapped accordingly to the output model.

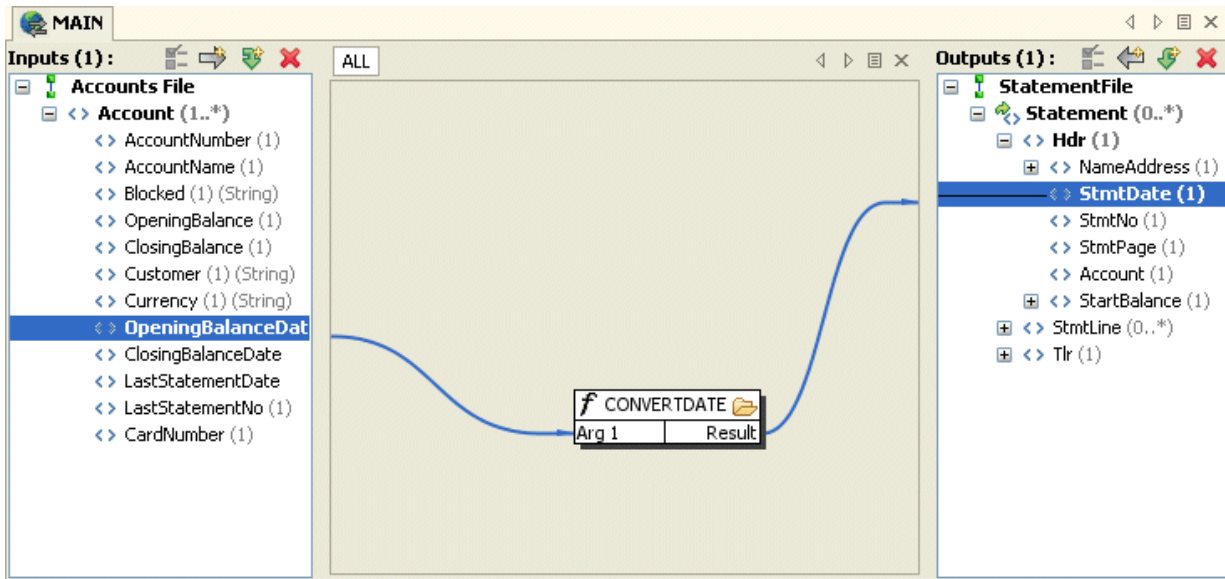
3. To set up the condition under which the output field should be instantiated, you can add a CONTAINS function as follows:

- a. Add a CONTAINS function to the transformation.
- b. Map the complex type that contains the relevant input element (in the Inputs section) to Arg1 in the CONTAINS function.
- c. Map the relevant input element itself (in the Inputs section) to Arg2 in the CONTAINS function.
- d. Map the Result of the CONTAINS function to the input Condition parameter of the instantiator.

Setting up Translations

Once you have added the required components to your transformation, you can connect them up with translations.

Translations are the connecting lines (or mappings) between transformation input and output models, and the various components applicable to a transformation model. The following example shows how the translation maps an OpeningBalanceDate field from the input data model into a CONVERTDATE function, and then maps the output of the function to a StmtDate field in the output data model. A transformation may contain multiple functions, other components, and translations between various elements in the input and output models.



Note: When you click a particular component within a transformation, the mappings associated with that component are highlighted. This allows you to more easily see the logical relationships between particular mappings and components.

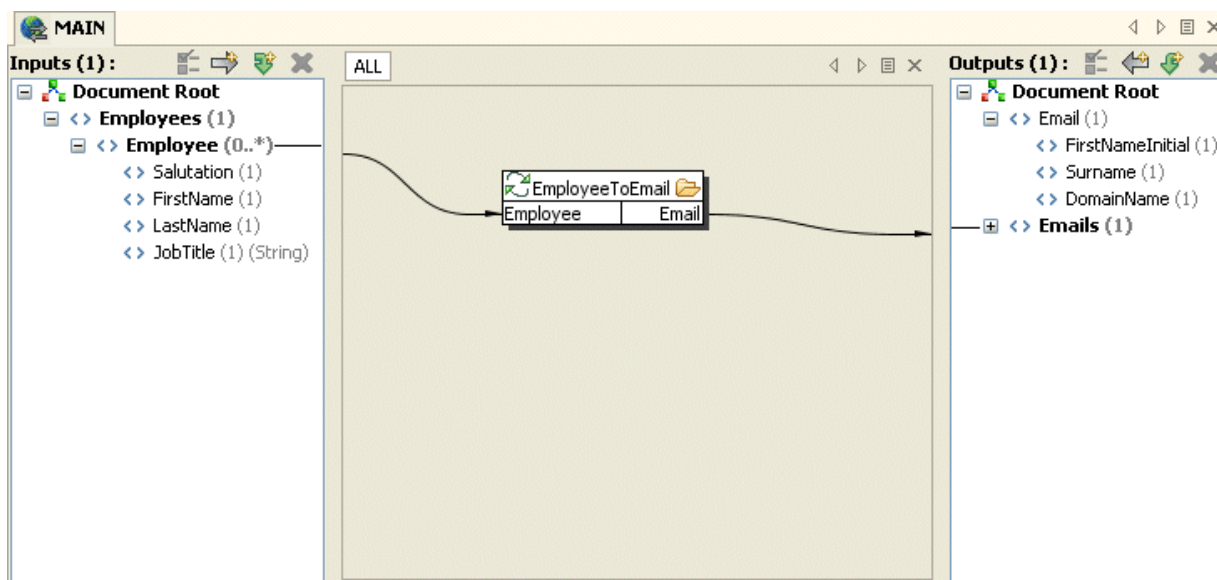
Using Curved Connectors

Select **View > Use Curved Connectors** to display your transformation mappings as curved lines rather than straight lines. Alternatively, right-click in the ALL (middle) section of a transformation tab and select **Use Curved Connectors**.

Finding Unmapped Components

You can perform some perfunctory validation of transformations. This validation takes the form of a search for mandatory components that are unmapped and therefore invalidate the transformation. Click the root node in the Outputs section and selecting **Find > Find Unmapped Components** from the menu bar.

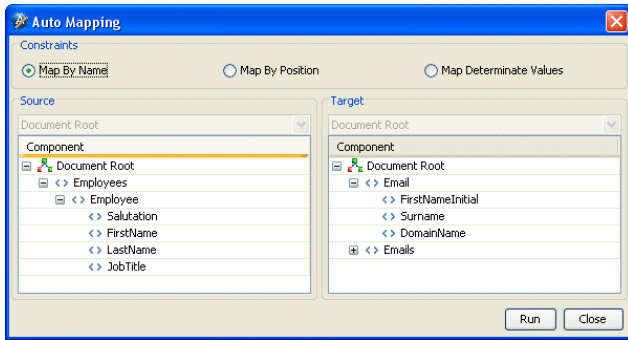
The following example illustrates usage of the Find Unmapped Components functionality:



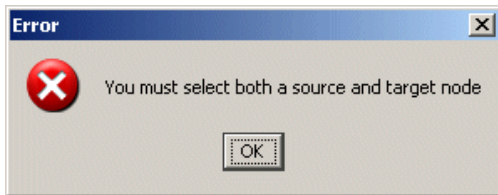
The output elements FirstNameInitial, Surname, and DomainName all have mandatory cardinality but are unmapped. In this case, if you click **Document Root** and select **Find > Find Unmapped Components**, the unmapped nodes are displayed as follows in the Find window:

Component	Model
1: Document Root/Email/FirstNameInitial	D:\My Documents\My ADS Projects\Examples\Basic\model\Example.tfd
1: Document Root/Email/Surname	D:\My Documents\My ADS Projects\Examples\Basic\model\Example.tfd
1: Document Root/Email/DomainName	D:\My Documents\My ADS Projects\Examples\Basic\model\Example.tfd


Any unmapped mandatory components can be mapped using the Auto Mapping functionality. Select **Tools > Auto Mapping** from the menu bar to open the Auto Mapping dialog:



The Constraints section of the dialog allows you to decide how the Auto Mapping process will create mappings between elements. Select the **Map By Name** radio button to create mappings between components with the same names. Select the **Map By Position** radio button to map between fields according to their order in the tree (that is, first to first, second to second, and so on). To initiate the mapping, select both nodes in the dialog. Otherwise, the following error message is displayed:



Automatically Aligning Transformation Components

You can automatically align the various components of a transformation by clicking the  icon on the palette. This allows you to select different layouts and reset the components of the transformation within the tab. This is particularly handy for very large transformations with possibly hundreds of different functions that can render the transformation very difficult to read.

Finding Components in a Transformation

Select the **Find > Find Components** option to open the [Find Components dialog](#) to search for a particular component or value within a transformation.

Globalizing a Local Transformation

Right-click on a local transformation and select **Globalize** to globalize that local transformation so that it can be reused in other parts of the transformation.

Moving Transformation Mappings

You can move particular components of a transformation to within a new local transformation, by dragging your mouse while holding the **Shift** key to select the component you wish to move. Then right-click the relevant component and select **Move Mappings**. You will then be prompted to type the name of the new local transformation to which you want to move the components. This is a very handy way of moving mappings and components without having to copy and paste them manually.

Exporting HTML from Transformations

You may export HTML documentation from your transformation by right-clicking the relevant transformation file in the Project window and selecting **Export > Export HTML**. This opens the HTML Export Wizard which will allow you to produce documentation files, including images, based on the transformation. This means that you can automatically document the logic behind the structure of a transformation.

Collaborating on Transformations

Different developers can collaborate on the same transformation file using their source code management (SCM) system.

As of ADS 3.9, you can merge any changes that you have made to a transformation (.tfd) file as you would with any other XML file.

Note: This means that you cannot open a transformation that has been saved in ADS Designer 3.9 in an earlier version of ADS.

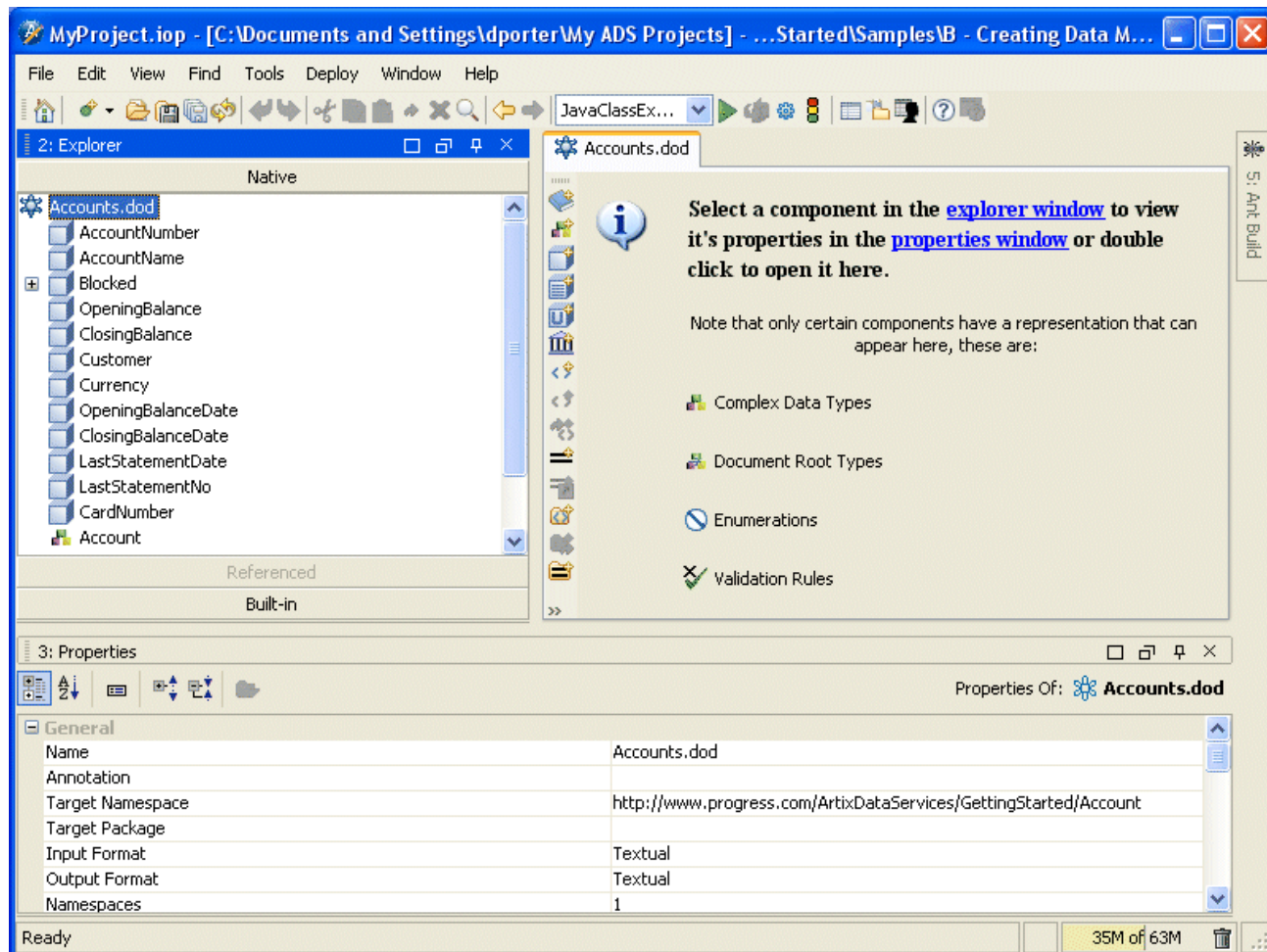
If you plan to check in your transformation files to an SCM, we recommend that you disable the [Save Layout](#) property for the transformation. Otherwise, merge tools will view any updates to the transformation components' x and y coordinates as a source change, thus making merging more difficult.

Model and Transformation Properties

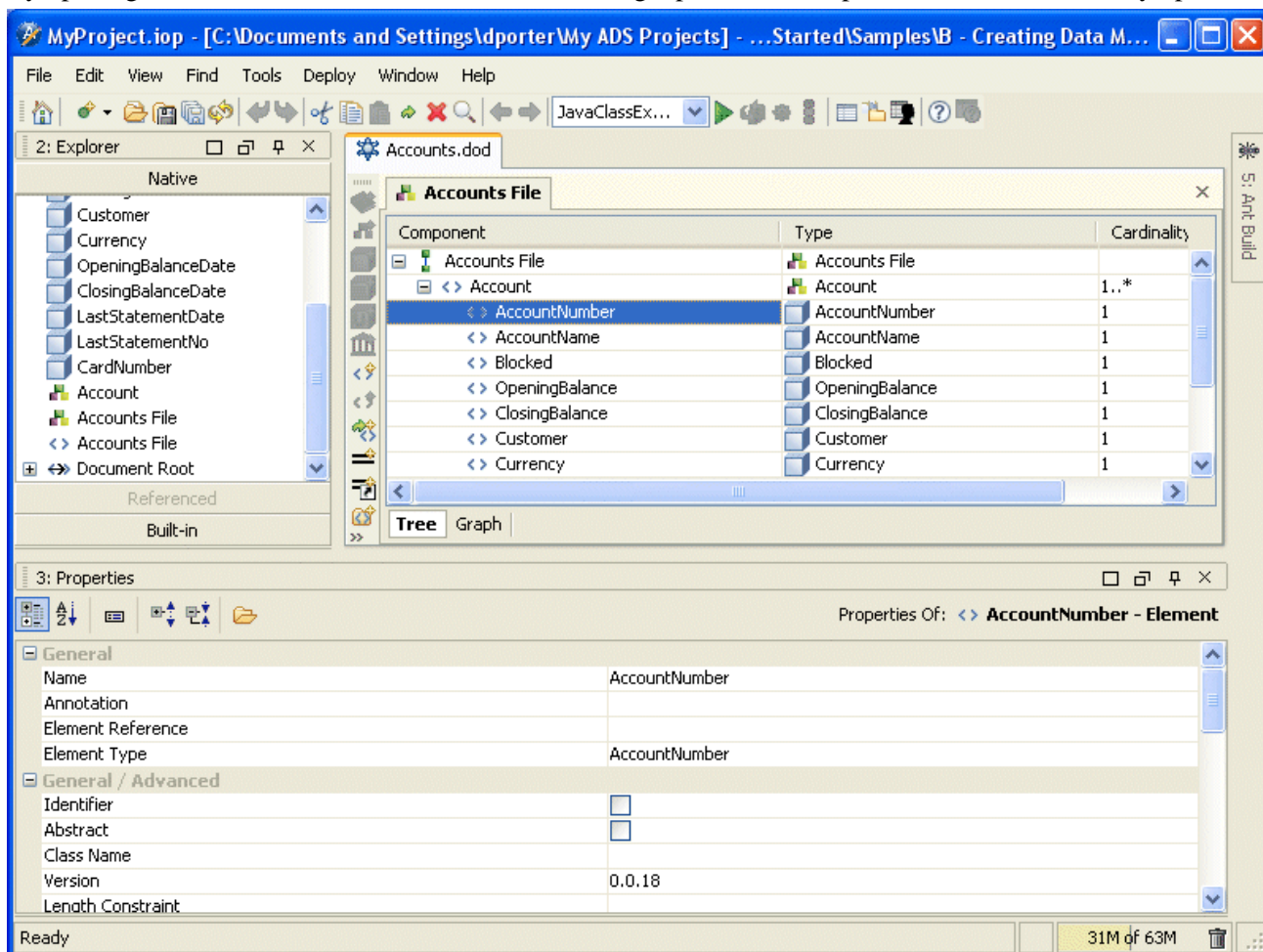
In ADS, data models, transformations and their components have properties associated with them. You can set these properties by clicking in the fields in the right-hand column of the [Properties window](#). Clicking in these fields triggers drop-down menus, dialogs, and a variety of other mechanisms for editing these values.

There are two ways of accessing the Properties for a particular data component:

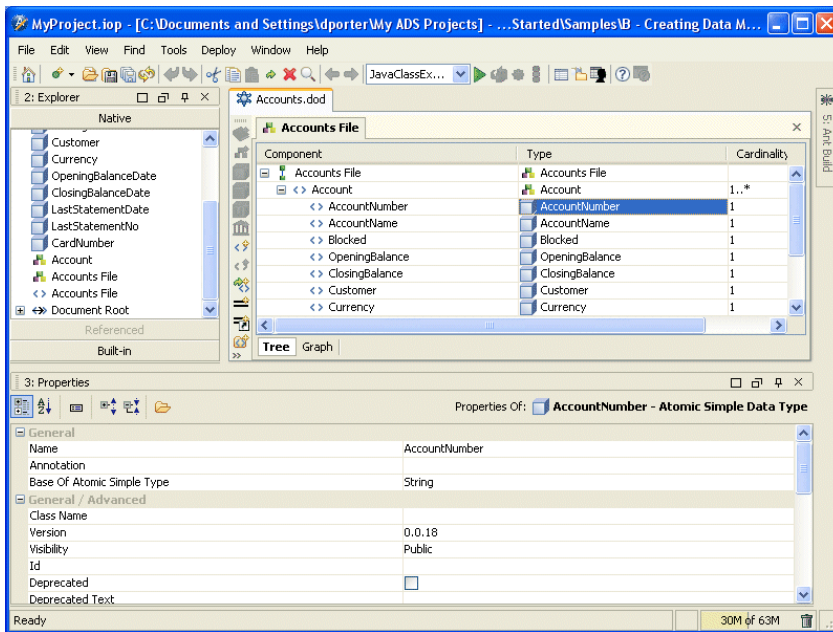
- By clicking the component in the Explorer window



- By opening a data model or transformation and clicking a particular component within the currently open tab.



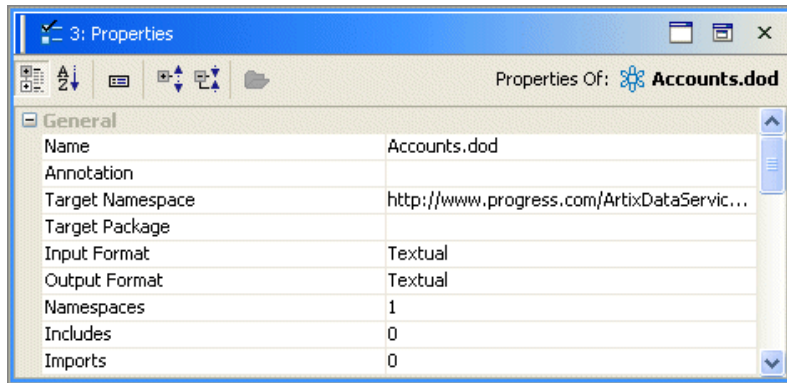
Note that the properties of the AccountNumber element in the above example are different from the properties of the Atomic Simple Data Type of the element (also called AccountNumber) as shown below.



Many ADS components, data models and transformations share the same sets of properties but often with subtle differences. Properties are categorized into the following groups:

- [General](#)
- [General/Advanced](#)
- [Presentation](#)
- [Presentation/Advanced](#)
- [Validation](#)
- [Validation/Advanced](#)
- [XML](#)
- [Database](#)
- [SWIFT](#)

General Properties



The properties available in the General section of the Properties window include:

- [Annotation](#)
- [Name](#)
- [Imports](#)
- [Includes](#)
- [Input Mask](#)
- [Output Mask](#)
- [Namespaces](#)
- [Target Namespace](#)
- [Base of Complex Type](#)
- [Bean Interface\(s\)](#)
- [Bean Superclass](#)
- [Content Model](#)
- [Derivation Method](#)
- [Element Reference](#)
- [Element Group Reference](#)
- [Element Type](#)
- [Base of Simple Type](#)
- [List Member Type](#)
- [Case Sensitive](#)
- [Union Member Types](#)
- [Attribute Type](#)
- [Attribute Reference](#)
- [Attribute Group Reference](#)

Note: The properties available vary depending on the currently selected component.

Annotation

Description: A human and/or machine readable annotation.

Used by: [Data Models](#), [Transformations](#), all [Simple Data Types](#), [Complex Data Types](#), [Elements](#), [Attributes](#), [Attribute Groups](#), [Enumerations](#), Validation Rules, and [Annotations](#)

Usage: See [Annotations](#).

Name

Description: Displays the name of the currently selected component and allows you to edit its name.

Used By: All components

Usage: Click in the right-hand cell to edit the name of the currently selected component.

Imports

Description: Specifies the number of models imported by this model and provides a way of editing the list of models, see [Includes, Imports and Redefines](#) for more detail.

Used By: [Data Models](#) and [Transformations](#)

Usage: Click in the right-hand cell to open a dialog allowing you to add or remove imported Data Models.

Includes

Description: A list of models included by a model (all included models must have the same target namespace as the main model) see [Includes, Imports and Redefines](#) for more details

Used By: [Data Models](#) and [Transformations](#)

Usage: You can edit this property in the same way as the Imports property but you must ensure that models included have the same target namespace as the current model or transform otherwise the model will not be added to the list of included models.

Input Mask

Description: The Input Mask property specifies the default mask for the input data. The input mask defines the format of data that will be expected to be parsed and validated by the deployed Data Model or Transformation. There are three possible input masks, Default, XML and Java Class.

Used By: [Data Models](#) and [Transformations](#)

Usage: The right-hand cell is a drop-down menu consisting of the possible input masks available.

Output Mask

Description: The Output Mask property specifies the default mask for the output data, the output mask defines the format of data that will be written out from a deployed Data Model or Transformation. There are five possible output masks, Default, XML, Java Class, Interchange and Tag Value Pair.


Used By: [Data Models](#) and [Transformations](#)

Usage: The right-hand cell is a drop-down menu consisting of the possible output masks available.

Namespaces

Description: Specify the number of [Namespace\(s\)](#) for the current Data Model and add and remove Namespaces.

Used By: [Data Models](#)

Usage: Click the right-hand cell to open the Namespaces dialog. Click the plus  icon to open a dialog where you can enter the URL of the namespace you are adding to the list. You can edit the Prefix value by clicking in the cell and adding text. For each namespace added, the number of namespaces displayed is incremented by 1.

Target Namespace

Description: Specify the [Namespace](#) specified by a [Data Model](#) or [Transformation](#)

Used By: [Data Models](#) and [Transformations](#)

Usage: Click in the right-hand cell to edit the [Target Namespace](#)

Base of Complex Type

Description: Specify the Complex Data Type that this Complex Data Type is based on or in Java terms the Complex Data Type that this Complex Data Type extends.

Used By: [Complex Data Type](#)

Usage: Click in the right-hand cell to open a dialog where you can select a Complex Data Type as the base type for this component. Initially this is set to **Native** to indicate that the base type is currently user-defined. Select **Built-in** to display common types such as string and integer. When you select a base type, a dialog displays warning you that settings will change. The name of the base type is shown in the field.

Bean Interface(s)

Description: Enables you to specify a comma separated list of interfaces that will be implemented/extended by the Bean interface/class representation of this type. The interfaces must be specified with their fully-qualified names.

Used By: [Complex Data Type](#) and [SWIFT Field Type](#)

Usage: Enter a comma separated list of fully-qualified interface names.

Bean Superclass

Description: Specify the fully qualified name of a class that will be the superclass of the Bean Class generated from this type.

Used By: [Complex Data Type](#) and [SWIFT Field Type](#)

Usage: Enter a fully-qualified class name for the superclass.

Content Model

Description: The Content Model specifies the ordering scheme to be used for children of this Complex Data Type.

Used By: [Complex Data Type](#) and [SWIFT Field Type](#)

Usage: Click in the right-hand cell and select one of the following from the drop-down list:

- **Sequence** - all children of this Complex Data Type must appear in the order in which they have been defined.
- **All** - the children of this Complex Data Type can appear in any order but can only occur once and only once.
- **Choice** - One and only one of the children of this Complex Data Type can appear at a time.

Derivation Method

Description: The Derivation method allows you to specify how the current type will be derived from its base type. It specifies two methods of derivation which correspond to derivation methods specified in the XML Schema specification:

- **Extension** - specify that this type is an extension of its base, for example, a 'Car' is an extension of a 'Vehicle'
- **Restriction** - specify that this type is a restriction of its base, for example, a '3 Wheeled Car' is a restriction of a 'Vehicle', specifically on the number of wheels.

Used By: [Complex Data Type](#) and [SWIFT Field Type](#)

Usage: Click in the right-hand cell and select from the drop-down list.

Element Reference

Description: As an alternative to specifying an explicit type for an element, you can refer to another element whose type will also be the type of this element.

Used By: [Elements](#)

Usage: Click in the right-hand cell to open a dialog from where you can select an element to reference.

Element Group Reference

Description: Same functionality as the Element Reference, allows an Element Group to reference another Element Group and use its base type definition.

Used By: [Element Groups](#)

Usage: Click in the right-hand cell to open a dialog from where you can select an element to reference.

Element Type

Description: Specify the type of this element.

Used By: [Elements](#)

Usage: Click in the right-hand cell to open a dialog from where you can select the type of this element.

Base of Simple Type

Description: Specify the base type of this Simple Type

Used By: [Simple Data Type](#)

Usage: Click in the right-hand cell to open a dialog from where you can select the base of this Simple Type:

List Member Type

Description: Specify the type of members that this List Member Type will contain.

Used By: [List Data Types](#)

Usage: Click in the right-hand cell to open a dialog from which you can specify the type for all members of this List Member Type.

Case Sensitive

Description: This property specifies whether or not the equality checks should ignore case difference. It is set to true by default.

Used By: [Enumerations](#)

Usage: Select the check box to set to true.

Union Member Types

Description: Displays the number of data types that are members of this Union Data Type and edits the member types that can be included in this Union Member Type.

Used By: [Union Data Types](#)

Usage: Click in the right-hand cell to open a dialog displaying the types that are permissible in this Union Data Type. Click the Plus and Minus icons to add and data types to and remove data types from the list of member types.

Attribute Type

Description: Specify the type for this Attribute

Used By: [Attributes](#)

Usage: Click in the right-hand cell to open a dialog displaying the types that can be specified for this Attribute.

Attribute Reference

Description: As an alternative to specifying a type for an attribute, you can reference an attribute defined elsewhere as the defining type for this Attribute

Used By: [Attributes](#)

Usage: Click in the right-hand cell to open a dialog displaying the types that can be specified for this Attribute Reference.

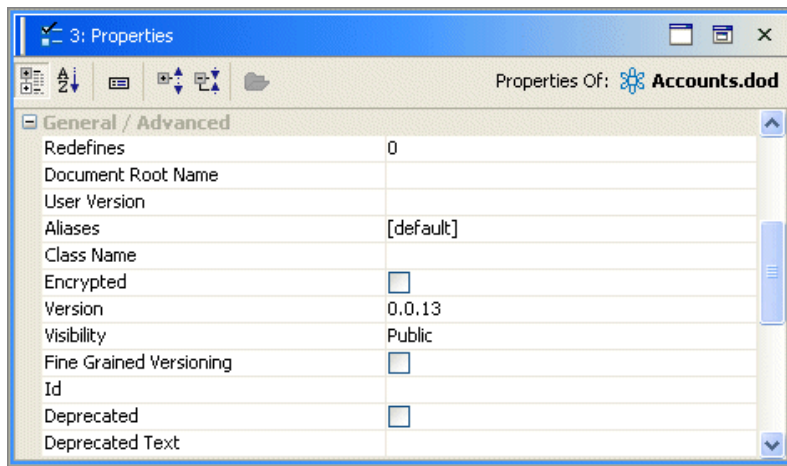
Attribute Group Reference

Description: Works in exactly the same way as the Attribute Reference.

Used By: [Attribute Groups](#)

Usage: Works in exactly the same way as the Attribute Reference.

General/Advanced Properties



The properties available in the General/Advanced section of the Properties window are a subset of the General properties group and include:

- [Process As Batch](#)
- [Deprecated](#)
- [Deprecated Text](#)
- [Encrypted](#)
- [Fine Grained Versioning](#)
- [Id](#)
- [Redefines](#)
- [User Version](#)
- [Version](#)
- [Visibility](#)
- [Length Constraint](#)
- [Cardinality Constraint](#)
- [Composition](#)
- [Abstract](#)
- [Ordering Type](#)
- [Transient](#)
- [Processing Order](#)
- [Validate Inputs](#)
- [Validate Outputs](#)
- [Document Root Name](#)
- [Base Transform](#)
- [Post Transform Action](#)
- [Pre Transform Action](#)
- [Transform Superclass](#)
- [Aliases](#)
- [Save Layout](#)

Note: The properties available vary depending on the currently selected component.

Process As batch

Description: This boolean property indicates whether the object should be processed as a batch

Used By: [Data models](#)

Usage: Select the check box to set the property to true.

Deprecated

Description: Boolean property that indicates whether this component has become deprecated in the Java sense of the word.

Used By: [Data Models](#), all Data Types, [Data Classification Groups](#), [Elements](#), [Enumerations](#), Validation Rules, [Attributes](#), [Annotations](#)

Usage: Select the check box to set the property to true.

Deprecated Text

Description: A field for specifying why a component has become deprecated and what to use instead of the deprecated component.

Used By: [Data Models](#), all Data Types, [Elements](#), [Enumerations](#), Validation Rules, [Attributes](#), [Annotations](#)

Usage: Enter the explanation for the component being deprecated in the text field.

Encrypted

Description: A Boolean property for setting whether the current Data Model or Transformation will be encrypted and compressed or not.

Used By: [Data Models](#) and [Transformations](#)

Usage: Select the check box to set the property to true.

Fine Grained Versioning

Description: A Boolean property for specifying a model's component parts are individually versioned.

Used By: [Data Models](#)

Usage: Select the check box to set the property to true.

Id

Description: The Id field allows you to create a unique identifier for the current component. The concept is founded on the XML concept of an id, whereby every element in an XML document has an id attribute that specifies a unique id for an element.



Used By: [Data Models](#), all Data Types, [Elements](#), [Enumerations](#), Validation Rules, [Attributes](#), [Annotations](#)

Usage: Enter an ID in the text field.

Redefines

Description: Specifies a list of models that are redefined by this model, showing how many models are redefined currently.

Used By: [Data Models](#)

Usage: Click in the right-hand cell to open a dialog allowing you to view the list of models that are currently redefined by this model. Click the  or  to add or remove Data Models from the list of models redefined by this Data Model from the dialog of available models.

User Version

Description: A field specifying textual information relating to the version of this Data Model

Used By: [Data Models](#)

Usage: This is a simple text field that can be edited by just clicking in the field and editing as normal.

Version

Description: This field is only enabled if the Fine Grained Versioning property has been set to true for the current Data Model.

Used By: [Data Models](#), all Data Types, [Elements](#), [Enumerations](#), Validation Rules, [Attributes](#), [Annotations](#) and [Transformations](#)

Usage: In conjunction with the **Fine Grained Versioning** property the **Version** field displays the current version number for the component in question. You can increment it manually by clicking into the field and editing the number.

Visibility

Description: Specify the visibility of a component to other component's outside its namespace. Levels of visibility are roughly analogous to the those used in Java.

Used By: [Data Models](#), all Data Types, [Elements](#), [Enumerations](#), Validation Rules, [Attributes](#), [Annotations](#)

Usage: Select a visibility level from the drop-down list.

Length Constraint

Description: An XPath expression that can be resolved to a numeric value which is then used to validate the length of the type in question

Used By: [Elements](#) and [Attributes](#)

Usage: Click in the right-hand cell to open an [XPath Editor](#) where you can set the length constraint.

Cardinality Constraint

Description: An XPath expression which is used to locate a numeric field that validates the cardinality of an element or attribute

Used By: [Elements](#) and [Attributes](#)

Usage: Click in the right-hand cell to open an [XPath Editor](#) where you can set the cardinality constraint.

Composition

Description: A boolean flag to determine whether a parent/child relationship (from a complex data object to an element/attribute) is a containment relationship or an association

Used By: [Elements](#)

Usage: Select the check box to set the property to true.

Abstract

Description: The Abstract property is a Boolean value that specifies whether an element can be used directly in an instance document.

Used By: [Complex Data Types](#) and [Elements](#)

Usage: Select the check box to set the property to true.

Ordering Type

Description: Enables you to specify that this Complex Data Type's children are ordered according to the order defined by another Complex Data Type whose Content Model is set to All

Used By: [Complex Data Type](#)

Usage: Click in the right-hand field to open a dialog from which you can select the Complex Data Type, whose order will dictate the ordering of this Complex Data Type's children. Once selected the name of the Complex Data Type appears in the field.

Transient

Description: Boolean setting that specifies whether an element should be serialized with its containing class

Used By: [Elements](#)

Usage: Select the check box to set the property to true.

Processing Order

Description: Setting that specifies whether transform mappings should be processed in the order of inputs or outputs

Used By: [Transformations](#)

Usage: Click in the right-hand field to open a drop-down menu with input/output settings.

Validate Inputs

Description: Boolean setting that specifies whether transform inputs should be validated before proceeding with a transformation

Used By: [Transformations](#)

Usage: Select the check box to set the property to true.

Validate Outputs

Description: Boolean setting that specifies whether transform outputs should be validated before proceeding with a transformation

Used By: [Transformations](#)

Usage: Select the check box to set the property to true.

Document Root Name

Description: The name to use for the [Document Root](#). This property maps to ecore:documentRoot.

Used By: [Data Models](#)

Usage: Enter the Document Root name as plain text.

Base Transform

Description: The Transformation from which this Transformation is derived

Used By: [Transformations](#)

Usage: Click in the right-hand field to open the Component dialog from where you can select a Base Transformation

Post Transform Action

Description: Java code that allows user to add functionality to the result of a transform.

Used By: [Transformations](#)

Usage: Click in the right-hand field to open an empty text pad with a predefined Java class.

Pre Transform Action

Description: Java code that allows user to add functionality to the input of a transform.

Used By: [Transformations](#)

Usage: Click in the right-hand field to open an empty text pad with a predefined Java class.

Transform Superclass

Description: Create a transform from another but with specified elements added or removed.

Used By: [Transformations](#)

Usage: Click in the right-hand field to make reference to a specified transform.

Aliases

Description: The aliases for the different sets of component names that can be applied to a data model.

Used By: [Data Models](#)

Usage: Specify a comma-delimited list of aliases. For example, specifying "[default],English,French" allows you to set up two sets of names for the components in the selected data model, with one set of names under the alias "English", and another set of names under the alias "French". This feature is useful when you wish to create internationalized models (that is, different versions of the same model in different languages). It can also be used to create different sets of names for a model relating to a particular standards library such as SWIFT, where you could choose between seeing a set of technical names and a set of business names for the same components.

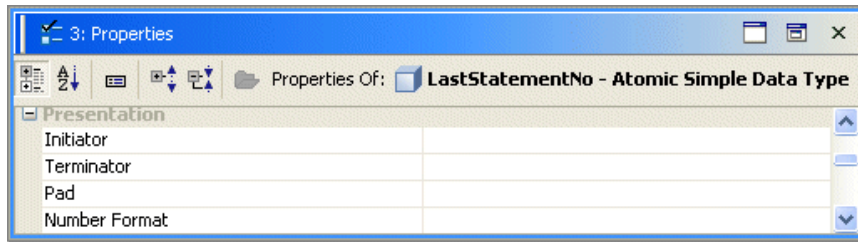
Save Layout

Description: If you change the layout of the components in a transformation, select this to ensure that the new layout is saved when you save the transformation.

Used By: [Transformations](#)

Usage: Selecting this option saves the new x and y coordinates for the transformation components in the transformation (.tfd) file. We recommend disabling this option if you plan to collaborate with other developers using a source code management system, as merge tools will view the new x and y coordinates as a change, thus making merging more difficult.

Presentation Properties



The properties available in the Presentation section of the Properties window are used to control how fields are formatted, what initiators and terminators are used, and so on. They include:


- [Initiator](#)
- [Pad](#)
- [Terminator](#)
- [Date Format](#)
- [Number Format](#)
- [True String](#)
- [Format Type](#)
- [Delimiter](#)

Note: The properties available vary depending on the currently selected component.

Initiator

Description: Specify the character string used to determine the start of this field.


Used By: All [Atomic Simple Types](#)

Usage: Click in the right-hand-cell to reveal the editable fields for the initiator. Click the  icon to open the Insert Character dialog from where you can select initiator characters. The selected initiators appear in the text fields, the field on the right shows the selected value in character format while the field on the left displays the value in hexadecimal. Non-printable characters that can't be rendered in Unicode are displayed as small square characters. Click the Plus icon to create a comma separated list of initiators, the first acts as the formatting initiator while the other initiators are recognized as valid initiators during parsing. Click inside each new line added to open the same text fields and buttons as in the Properties window.

Pad

Description: Specify a repeating character used to extend the width of a field to a certain value.

Used By: All [Atomic Simple Types](#)


Usage: Click in the right-hand cell to open a dialog where you can set a pad character. Click the  icon to open another dialog from where you can select a single pad character. Once a character has been inserted the rest of the Pad dialog becomes active. The menu to the right, allows you to decide where to position the pad character in

relation to the data for this field. The radio button allows you to select how many pad characters will be used to pad out this field. The "Pad to Minimum" size setting uses the minimum size value for this field as the number of characters to pad to. The "Pad To Size" setting activates the spinner which allows you to set the number of characters that will be used to pad this field. Having populated the dialog the details are displayed in the Pad field.

Terminator

Description: A character String used to set the end of a field.


Used By: All [Atomic Simple Types](#)

Usage: Click in the right-hand cell to reveal the editable fields for the Terminator. Click the  icon to open the Terminator character dialog from where you can select one or more characters to create a terminator. Click the plus icon on the Terminator field to open the Advanced dialog from where you can combine terminator characters to create complex terminators.

Date Format

Description: The Date Format field allows you to set a format that is used to parse incoming dates.

Used By: [Atomic Simple Types](#) with a base type of [Generic Date](#).


Usage: Click the right-hand cell to open a dialog where you can create a new date format by specifying a pattern recognizable by the `java.text.SimpleDateFormat` class. Click the  icon to open a dialog containing the characters permissible in a valid Java date format. Alternatively, you can set a date format using an off-the-shelf Java date/time format configured for the locale specified for the field. You can select a date or time, or a date and time combined format. There are four styles of format available.

Select the **Lenient** check box so that the date format will interpret invalid dates or times rather than generate an error on the date or time format. If **Lenient** is not selected, an error is generated on any date or time value that does not match the required format. In this case, the "incorrect time" means the wrong time for the current time zone and Daylight Saving Time (DST), if applicable. For more details, see <http://java.sun.com/javase/timezones> and http://en.wikipedia.org/wiki/Daylight_saving_time_around_the_world.

Number Format

Description: Specify a number format for parsing numbers.


Used By: All [Simple Data Types](#) with a numeric base type

Usage: Click the right-hand cell to open the Number Format dialog. Click the  icon to open a dialog where you can select the characters in the number format pattern. Only characters permissible for Java number formats are available. The **Localized type** radio button enables you to select from a drop-down menu of number formats for Integer, Number, Currency and Percent that will format values in a locale-sensitive manner.

True String

Description: The True String property enables you to specify a custom representation of the boolean value for 'true'

Used By: All [Atomic Simple Types](#) with a base type of [boolean](#)

Usage: Click the right-hand cell to reveal the editable fields for the True String property. Click the  icon to open a dialog from which you can insert characters into the text fields. The selected initiators appear in the text fields, the field on the right shows the selected value in character format while the field on the left displays the value in hexadecimal. Non-printable characters that can't be rendered in Unicode are displayed as small square characters.

Format Type

Description: Specify the method for interpreting a field's format


Used By: [Complex Data Types](#)

Usage: Select an option from the drop-down list.

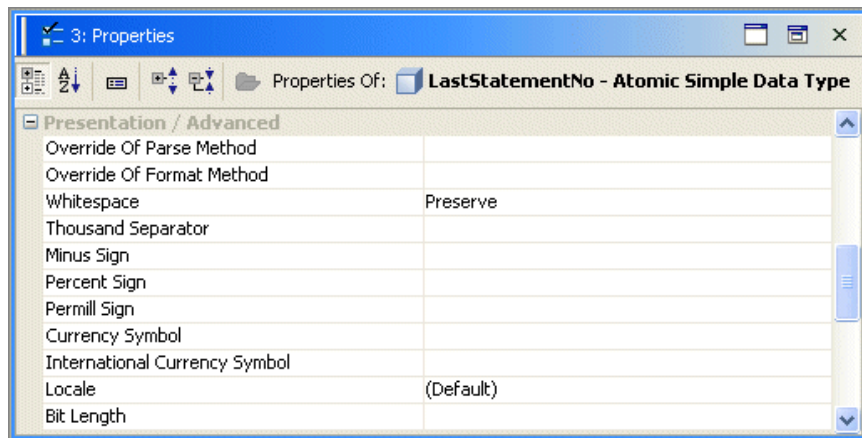
Delimiter

Description: Specify a character string used to determine the where one field ends and another begins

Used By: [Complex Data Types](#)

Usage: Click the right-hand cell to open the Delimiter dialog. Click the  icon to open a dialog from where you can select characters to make up the delimiter. Click the plus icon to open the Advanced dialog where you can combine characters to create a multi-character delimiter.

Presentation/Advanced Properties



The properties available in the Presentation/Advanced section of the Properties window include:

- [Locale](#)
- [Override of Format Method](#)
- [Override of Parse Method](#)
- [Whitespace](#)
- [Time Zone](#)
- [Currency Symbol](#)
- [International Currency Symbol](#)
- [Minus Sign](#)
- [Percent Sign](#)
- [Permill Sign](#)
- [Thousand Separator](#)
- [Decimal Separator](#)
- [Decimal Separator Always Shown](#)
- [Monetary Decimal Separator](#)
- [Collate](#)
- [Document Root Discriminator](#)
- [BitMap Content Model](#)
- [BitMap Index](#)
- [Align To Border](#)
- [Offset Within Spreadsheet](#)

Note: The properties available will vary depending on the currently selected component.

Locale

Description: The locale that is applied to this field's data for formatting purposes

Used By: All [Atomic Simple Types](#)

Usage: Select a Locale from the drop-down list of valid Java locales

Override of Format Method

Description: Write Java code for a method that over-rides the formatObject() method in biz.c24.io.api.data.SimpleDataType

Used By: All [Simple Data Types](#)

Usage: Click in the right-hand cell to open a dialog where you can enter Java code for the body of the overridden version of formatObject(). The Highlight checkbox toggles code high-lighting on and off.

Override of Parse Method

Description: Write Java code for a method that over-rides the parseObject() method in biz.c24.io.api.data.SimpleDataType

Used By: All [Simple Data Types](#)

Usage: Click in the right-hand cell to open a dialog where you can enter Java code for the body of the overridden version of parseObject().

Whitespace

Description: The behavior applied to a field's whitespace after parsing

Used By: All [Atomic Simple Types](#)

Usage: Click in the right-hand cell to display the following options:

- Preserve - keeps all whitespace as it was in the source document.
- Replace - replaces tabs (0x09), line feeds(0x0a) and carriage returns (0x0d) with spaces (0x20).
- Collapse - performs a replace first, then collapses multiple space characters into a single space.

Time Zone

Description: Specify a time zone for the current field

Used By: [Generic Date](#)

Usage: Click in the right-hand cell to open a drop-down menu of valid Java-recognized time zones

Currency Symbol

Description: Specify a character to act as the default currency symbol for this field.

Used By: All [Atomic Simple Types](#) based on numbers

Usage: Click in the right-hand cell to reveal the Currency Symbol text fields. Clicking on the down arrow icon opens the Insert Character dialog to select a character to represent the currency symbol. Having selected a character to represent the currency symbol it is displayed in the field in its Unicode and hexadecimal format

International Currency Symbol

Description: Specify a character to act as the default international currency symbol for this field.

Used By: All [Atomic Simple Types](#) based on numbers

Usage: Click in the right-hand cell to reveal the Currency Symbol text fields. Clicking the down arrow opens the Insert Character dialog to select a character to represent the currency symbol. Having selected a character to represent the currency symbol it is displayed in the field in its Unicode and hexadecimal format

Minus Sign

Description: Sets the default representation of the minus sign for this field.

Used By: All [Atomic Simple Types](#) based on numbers

Usage: Click in the right-hand cell to reveal the Minus Sign text fields. Clicking the down arrow opens the Insert Character dialog to select a character to represent the currency symbol. Having selected a character to represent the currency symbol it is displayed in the field in its Unicode and hexadecimal format.

Percent Sign

Description: Sets the default representation of the percent sign for this field.

Used By: All [Atomic Simple Types](#) based on numbers

Usage: Click in the right-hand cell to reveal the Percent Sign text fields. Clicking on the down arrow icon opens the Insert Character dialog to select a character to represent the currency symbol. Having selected a character to represent the currency symbol it is displayed in the field in its Unicode and hexadecimal format

Permill Sign

Description: Sets the default representation of the permill sign for this field.

Used By: All [Atomic Simple Types](#) based on numbers

Usage: Click to the right of the Permill Sign label to reveal the Permill Sign text fields. Clicking on the down arrow icon opens the Insert Character dialog to select a character to represent the currency symbol. Having selected a character to represent the currency symbol it is displayed in the field in its Unicode and hexadecimal format

Thousand Separator

Description: Sets the default representation of the thousand separator sign for this field.

Used By: All [Atomic Simple Types](#) based on numbers

Usage: Click in the right-hand cell to reveal the Thousand Separator text fields. Clicking on the down arrow icon opens the Insert Character dialog to select a character to represent the currency symbol. Having selected a character to represent the currency symbol it is displayed in the field in its Unicode and hexadecimal format

Decimal Separator

Description: Sets the default representation of the decimal separator sign for this field.

Used By: All [Atomic Simple Types](#) based on floating point numbers

Usage: Click to the right of the Decimal Separator label to reveal the Decimal Separator text fields. Clicking on the down arrow icon opens the Insert Character dialog to select a character to represent the currency symbol. Having selected a character to represent the currency symbol it is displayed in the field in its Unicode and hexadecimal format

Decimal Separator Always Shown

Description: A Boolean value specifying whether the decimal separator should always be shown even when its not required e.g.: true = 100. false = 100

Used By: All [Atomic Simple Types](#) based on floating point numbers

Usage: Select the check box to set the property to true.

Monetary Decimal Separator

Description: Sets the default representation of the monetary decimal separator sign for this field.

Used By: All [Atomic Simple Types](#) based on floating point numbers

Usage: Click in the right-hand cell to reveal the Monetary Decimal Separator text fields. Clicking on the down arrow icon opens the Insert Character dialog to select a character to represent the currency symbol. Having selected a character to represent the currency symbol it is displayed in the field in its Unicode and hexadecimal format

Collate

Description: A Boolean setting for specifying whether elements of a complex data type can be interpreted as a concatenated character string

Used By: [Complex Data Types](#)

Usage: Select the check box to set the property to true.

Document Root Discriminator

Description: Writes Java code deployed into a method that will be used to look up the expected element.

Used By: All [Data Models](#)

Usage: Click in the right-hand cell to open a dialog where you can enter Java code for the body of the element.

BitMap Content Model

Description: The internal structure of a bitmap which in turn defines the structure of a complex type.

Used By: [Complex Data Types](#)

Usage: Click in the right-hand cell to launch a dialog. Clicking on the + button allows you to make selections.

Bitmap Index

Description: The index of this element in its containing complex types bitmap.


Used By: [Elements](#)

Usage: Click in the right-hand cell to launch a dialog.

Align To Border

Description: Used to align the data of a message frame to a border specified in bytes.

Used By: [Complex Data Types](#)

Usage: Click in the right-hand cell to open a dialog where you can set a pad character that will be used to align the data. Click the  icon to open another dialog from where you can select a single pad character. Select the number of bytes for the border from the drop-down list.

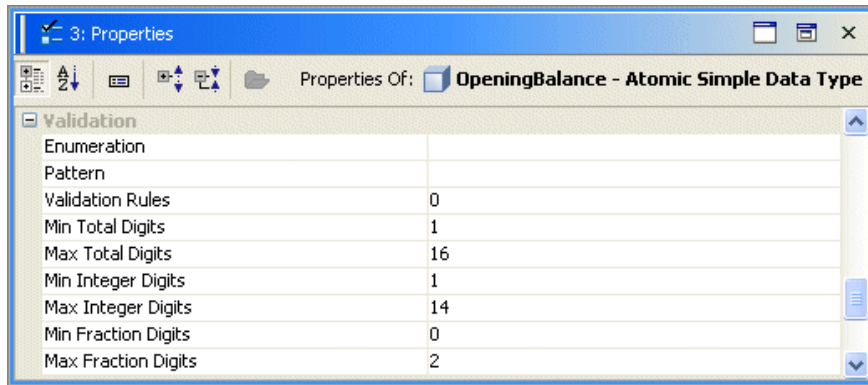
Offset Within Spreadsheet

Description: When loading data from a spreadsheet, defines the relative offset of the data for the current element in R1C1 format, with a base of R0C0, in other words offset on 0 columns and 0 rows.

Used By: [Elements](#)

Usage: Click in the right-hand cell and enter the offset value in R1C1 format.

Validation Properties



The properties available in the Validation section of the Properties window are used to set rules that will govern what data is permissible for a given field. They include:

- [Enumeration](#)
- [Pattern](#)
- [Validation Rules](#)
- [Max Length](#)
- [Min Length](#)
- [Max Total Digits](#)
- [Min Total Digits](#)
- [Max Fraction Digits](#)
- [Min Fraction Digits](#)
- [Max Integer Digits](#)
- [Min Integer Digits](#)
- [Default Value](#)
- [Fixed Value](#)
- [Presence](#)
- [Max Cardinality](#)
- [Min Cardinality](#)

Note: The properties available will vary depending on the currently selected component.



Enumeration

Description: Specify a list of values that will be used to validate data for this field

Used By: All [Atomic Simple Data Types](#) and [Complex Data Types](#)

Usage: Click in the right-hand cell to open the Select Component dialog. This contains the following options:

- Enumeration - select to create a local enumeration for this type, it will only be applied to instances of this type
- Global - select to create a global enumeration that may be reused across a number of types

Select **Enumeration** to open the New Enumeration dialog. Give is the enumeration a name and click **OK**. A tab containing the enumeration is added to the main window. Click the  and  icons in the Enumeration tab to add and remove entries to the enumeration. See [Enumerations](#) for details.

Pattern

Description: The Pattern property specifies a regular expression character string of a certain type used to validate field content



Used By: All [Atomic Simple Data Types](#) and [Complex Data Types](#)

Usage: Click in the right-hand cell to display the Pattern components drop-down menu, where you can specify a regular expression language, and a text field for writing a regular expression. The button icon is enabled when Java Regex is the selected regular expression language, when its pressed it will open a dialog containing all the characters available in the Java regular expression lexicon. double-clicking to the right of the required character class populates the pattern text field at the bottom of this dialog so user is able to view selection before clicking ok to exit.

Validation Rules

Description: The Validation Rules property displays the number of validation rules applicable to this type and allows you to add a validation rule which might provide validation of this type depending on the context in which an instance of the type is currently being used.

Used By: All [Atomic Simple Data Types](#) and [Complex Data Types](#)

Usage: Click in the right-hand cell to open a dialog. Click the  and  icons to add and remove validation rules for this type. In the Add Validation Rule dialog there are two possible options:

- Local - this allows you to create a validation rule that can only be used by this type
- Global - a global validation rule is reusable across more than one type

When the Local option is selected the Edit and New Validation Rule buttons are activated, pressing the New Validation Rule icon. Having named the new validation rule , TestValidationRule in the example below, click the rule tab shown in the main window.

By default the XPath type is displayed first, this consists of text area for editing a valid XPath expression that will be used for validating instances of this type and another field where an error message can be edited. This error message is displayed or logged when a validation error occurs for instances of this type.

The Java type, when selected allows you to add Java code by overriding the validate() method on the biz.c24.io.api.data.ComplexDataObject class.

Selecting Domain Constraint from the list allows you to either enter dynamic and/or static domain constraints. Domain constraints are powerful and useful in that they make it possible to change validation at run time.

User can choose to ignore the document node in the XPath expression by checking the ignore document node box. If this option is checked, the XPath expression should not include the name of the component whose type has this

rule applied to it. Note that this is the preferred approach since the type to which this rule is applied might be used by multiple components.

If the namespace awareness checkbox is checked, the XPath expression must use the prefixes defined in the namespaces table. If this property is not selected, the resulting XPath will behave as if the elementFormDefault and attributeFormDefault were both set to 'unqualified'.

Select **Factory Lookup** from the type list box enables to add references to [factory lookup](#) classes to which validation is delegated.

Returning to the Add Validation Rule dialog by selecting the Global option you can select a global validation rule from the Native, Referenced or Built-in categories

Once a validation rule(s) has been added the number of validation rules is incremented by 1

Max Length

Description: Specify the maximum number of characters permitted in this field

Used By: All [Atomic Simple Data Types](#) derived from Strings

Usage: Click in the right-hand cell to set the Max Length by typing an integer number

Min Length

Description: Specify the minimum number of characters permitted in this field

Used By: All [Atomic Simple Data Types](#) derived from Strings

Usage: Click in the right-hand cell to set the Min Length by typing an integer number

Max Total Digits

Description: Specify the maximum number of digits permitted in this field

Used By: All [Atomic Simple Data Types](#) derived from numbers

Usage: Click in the right-hand cell to set the Max Total Digits by typing an integer number

Min Total Digits

Description: Specify the minimum number of digits permitted in this field

Used By: All [Atomic Simple Data Types](#) derived from numbers

Usage: Click in the right-hand cell to set the Min Total Digits by typing an integer number

Max Fraction Digits

Description: Specify the maximum number of fractional digits permitted in this field

Used By: All [Atomic Simple Data Types](#) derived from floating point numbers

Usage: Click in the right-hand cell to set the Max Fraction Digits by typing an integer number

Min Fraction Digits

Description: Specify the minimum number of fractional digits permitted in this field

Used By: All [Atomic Simple Data Types](#) derived from floating point numbers

Usage: Click in the right-hand cell to set the Min Fraction Digits by typing an integer number

Max Integer Digits

Description: Specify the maximum number of integer digits permitted in this field

Used By: All [Atomic Simple Data Types](#) derived from floating point numbers

Usage: Click in the right-hand cell to set the Max Integer Digits by typing an integer number

Min Integer Digits

Description: Specify the minimum number of integer digits permitted in this field

Used By: All [Atomic Simple Data Types](#) derived from floating point numbers

Usage: Click in the right-hand cell to set the Min Integer Digits by typing an integer number

Default Value

Description: Specify a default value for an element

Used By: [Elements](#)

Usage: Click in the right-hand cell to set a String as the default value for this field

Fixed Value

Description: Specify a value that an element must have to be valid

Used By: [Elements](#)

Usage: Click in the right-hand cell to set a String as the fixed value for this field

Presence

Description: Specify the occurrence of an attribute according to a restraint indicator similar to XML Schema

Used By: [Attributes](#)

Usage: Click in the right-hand cell to select the level of presence from the drop-down menu

Max Cardinality

Description: Specify the maximum number of times that this element can appear within its enclosing context

Used By: [Elements](#)

Usage: Click in the right-hand cell to set the maximum cardinality using a valid cardinality symbol or number

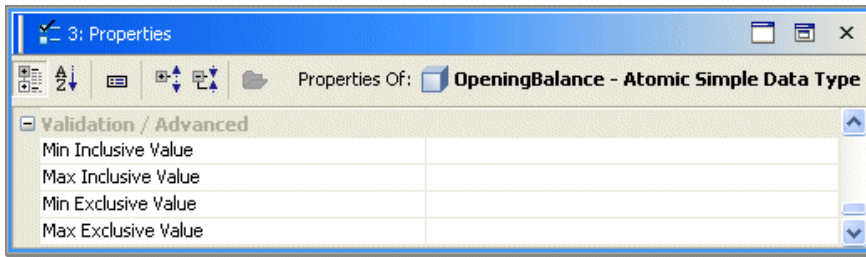
Min Cardinality

Description: Specify the minimum number of times that this element can appear within its enclosing context

Used By: [Elements](#)

Usage: Click in the right-hand cell to set the minimum cardinality using a valid cardinality symbol or number

Validation Advanced Properties



The properties available in the Validation Advanced section of the Properties window allow you to fine tune validation properties. They include:

- [Max Exclusive Value](#)
- [Min Exclusive Value](#)
- [Max Inclusive Value](#)
- [Min Inclusive Value](#)

Note: The properties available vary depending on the currently selected component.

Max Exclusive Value

Description: Specify the maximum exclusive value for a field

Used By: [Atomic Simple Data Types](#) based on Numbers and Dates

Usage: Enter the maximum exclusive value for this field as a String

Min Exclusive Value

Description: Specify the minimum exclusive value for a field

Used By: [Atomic Simple Data Types](#) based on Numbers and Dates

Usage: Enter the minimum exclusive value for this field as a String

Max Inclusive Value

Description: Specify the maximum inclusive value for a field

Used By: [Atomic Simple Data Types](#) based on Numbers and Dates

Usage: Enter the maximum inclusive value for this field as a String

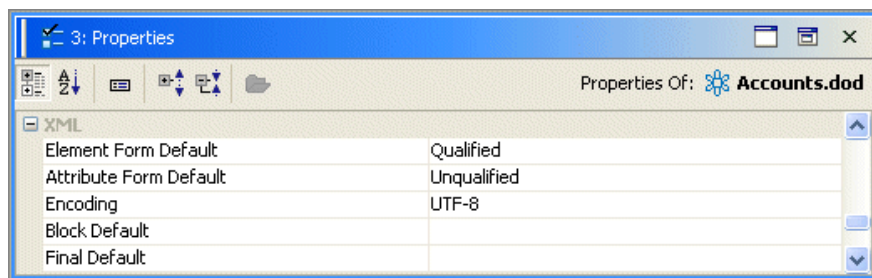
Min Inclusive Value

Description: Specify the minimum inclusive value for a field

Used By: [Atomic Simple Data Types](#) based on Numbers and Dates

Usage: Enter the minimum inclusive value for this field as a String

XML Properties



The properties available in the XML section of the Properties window have a bearing on the deployment of XML Schemas from [Data Models](#). They include:

- [Attribute Form Default](#)
- [Block Default](#)
- [Element Form Default](#)
- [Encoding](#)
- [Final Default](#)
- [Complex Type Final](#)
- [Mixed](#)
- [Type Block](#)
- [Simple Type Final](#)
- [Form](#)
- [Element Block](#)
- [Element Final](#)
- [Key](#)
- [Key Reference](#)
- [Nillable](#)
- [Substitution Group](#)
- [Unique](#)
- [Any Namespaces](#)
- [Process Contents](#)

Note: The properties available will vary depending on the currently selected component.

Attribute Form Default

Description: Set the default form for [attributes](#) either unqualified or qualified relates to XML Schema properties for the xsd:schema element

Used By: [Data Models](#)

Usage: Click in the right-hand cell to select from a drop-down menu the default form for attributes.

Block Default

Description: Set what derivations should be blocked from XML Schemas derived from this Data Model.

Used By: [Data Models](#)

Usage: Click in the right-hand cell to select from a drop-down menu what combinations of derivation are blocked in this Data Model.

Element Form Default

Description: Set the default form for elements either unqualified or qualified relates to XML Schema properties for the xsd:schema element.

Used By: [Data Models](#)

Usage: Click in the right-hand cell to select from a drop-down menu the default form for [Elements](#).

Encoding

Description: Specify the character encoding to use for the XML Schema that will be generated from this Data Model.

Used By: [Data Models](#)

Usage: Click in the right-hand cell to select from a drop-down menu the type of encoding to use.

Final Default

Description: Specify the behavior of elements that refer to this element as the head of their substitution group, All prevents all elements from doing this, Extension prevents extensions of this type and Restriction prevents restrictions of this type.

Used By: [Data Models](#)

Usage: Click in the right-hand cell to select from a drop-down menu the level of control for the Final Default.

Complex Type Final

Description: Specifies how a Complex Type can be extended in XML Schema terms.

Used By: [Complex Data Types](#)

Usage: Click in the right-hand cell to select from a drop-down menu how a complex type maybe extended.

Mixed

Description: A Boolean setting specifying whether character data can appear between elements.

Used By: [Complex Data Types](#)

Usage: Click in the right-hand cell to set this property to true or false by checking or unchecking the Mixed checkbox.

Type Block

Description: Specify which type derivations can substitute a type.

Used By: [Complex Data Types](#)

Usage: Click in the right-hand cell to select from a drop-down menu a type derivation.

Simple Type Final

Description: Specify how a type can be extended

Used By: [Atomic Simple Types](#)

Usage: Click in the right-hand cell to select from a drop-down menu how it will be extended.

Form

Description: Boolean setting to specify whether to qualify the name of attributes derived from this attribute type.

Used By: [Attributes](#) and [Elements](#)

Usage: Click in the right-hand cell to select from a drop-down menu whether the attribute is qualified.

Element Block

Description: Specify which element derivations can substitute an element.

Used By: [Elements](#)

Usage: Click in the right-hand cell to select from a drop-down menu the derivation to choose.

Element Final

Description: Specify how an element can be extended.



Used By: [Elements](#)

Usage: Click in the right-hand cell to select from a drop-down menu the extension model to use.

Key

Description: Specify a list of key identity constraints, the field when in an idle state displays the number of key constraints associated with this element





Used By: [Elements](#)

Usage: Click in the right-hand cell to open the Key Constraints dialog. Click the  and  icons to add and remove key constraints from the list. Clicking in the Fields cell opens the Fields dialog. Clicking on the Plus and Minus buttons enables you to add and remove fields to and from the dialog.

Key Reference

Description: Specify a list of key reference identity constraints, the field when in an idle state displays the number of key reference constraints associated with this element

Used By: [Elements](#)

Usage: Click in the right-hand cell to open the Key Ref Constraints dialog. Click the  and  icons to add and remove key reference constraints from the list. Click in the Fields cell to open the Fields dialog. Click  and  icons to add and remove fields to and from the dialog.

Nilable

Description: A Boolean value for specifying whether nil is a permitted value for this element

Used By: [Elements](#)

Usage: Select the check box to set to true.

Substitution Group

Description: Specify the substitution group to which an element belongs





Used By: [Elements](#)

Usage: Click in the right-hand cell to open the Select Component dialog from where you can select an element that is the substitution group for this element. From here, you can select an element from the Local category.

Unique

Description: Specify a list of unique identity constraints, the field when in an idle state displays the number of unique identity constraints associated with this element



Used By: [Elements](#)

Usage: Click in the right-hand cell to open the Unique Constraints dialog. Click the  and  icons to add and remove unique constraints from the list. Click in the Fields cell to open the Fields dialog. Click the  and  icons to add and remove fields to and from the dialog

Any Namespaces

Description: The valid list of namespaces applicable to this Any Attribute

Used By: [Any Attributes](#)

Usage: Click in the right-hand cell to open the Any Namespaces dialog. Click the  and  icons to add and remove namespaces from the list of available namespaces.

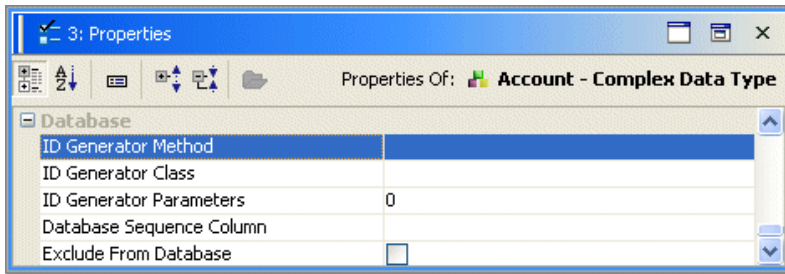
Process Contents

Description: Specifies how the contents of an any attribute should be processed

Used By: [Any Attributes](#)

Usage: Click in the right-hand cell to open the process contents menu.

Database Properties



The properties available in the Database section of the Properties window include:

- [ID Generator Method](#)
- [ID Generator Class](#)
- [ID Generator Parameters](#)
- [Database Sequence Column](#)
- [Exclude From Database](#)
- [Database Primary Key](#)

Note: The properties available vary depending on the currently selected component.

ID Generator Method

Description: Method used to determine the ID of a new record in a database.

Used By: [Data Models](#)

Usage: Click in the right-hand cell to select the preferred setting for this property from a drop-down menu.

ID Generator Class

Description: A Java class used to generate unique identifiers for new database records.

Used By: [Complex Data Types](#)

Usage: Enter the name of the ID generator class in the right-hand cell.

ID Generator Parameters

Description: Parameters passed to the Java class to configure or initialize the generator instance.

Used By: [Complex Data Types](#)

Usage: Enter the parameters required in the right-hand cell.

Database Sequence Column

Description: Specify the column in the database table that persists the sequence of the records.

Used By: [Complex Data Types](#)

Usage: Type the name of the sequence column in the right-hand cell.

Exclude From Database

Description: Boolean setting specifying whether this object exists as a valid database entity.

Used By: [Complex Data Types](#) and [SWIFT Data Types](#)

Usage: Select or clear the check box.

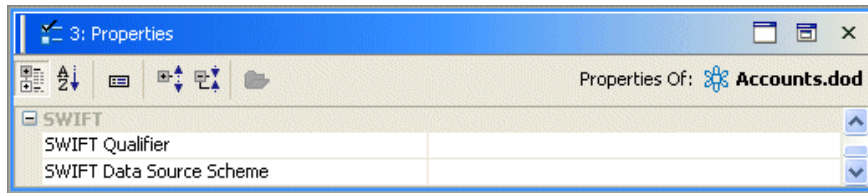
Database Primary Key

Description: Specify whether this type should be a key and if so primary or foreign.

Used By: [Attributes](#) and [Elements](#)

Usage: Select the check box to set the type as a primary key.

SWIFT Properties



The properties available in the SWIFT section of the Properties window allow you to configure certain properties pertaining to SWIFT standard specifics. They include:

- [SWIFT Data Source Scheme](#)
- [SWIFT Qualifier](#)
- [Generic](#)
- [Options Allowed](#)
- [Tag](#)

Note: The properties available vary depending on the currently selected component.

SWIFT Data Source Scheme

Description: Specify a Simple Data Type to be the Data Source Scheme for this Data Model.

Used By: [Data Models](#)

Usage: Click in the right-hand cell to open the Component dialog from where you can select a Simple Data Type to represent this Data Source Scheme.

SWIFT Qualifier

Description: Specify the type representing the SWIFT qualifier for this type.

Used By: [Data Models](#)

Usage: Click in the right-hand cell to open the Component dialog from where you can select a Simple Data Type.

Generic

Description: Boolean setting specifying whether a field of this type can contain a qualifier and a data source scheme.

Used By: [SWIFT Data Types](#)

Usage: Click in the right-hand cell to select from a drop-down menu the true or false setting for this property.

Options Allowed

Description: Boolean setting specifying whether SWIFT options are permitted or that the default option only can be used.

Used By: [SWIFT Data Types](#)

Usage: Click in the right-hand cell to select from a drop-down menu the true or false setting for this property.

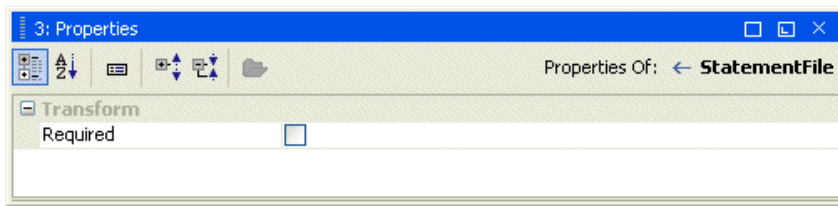
Tag

Description: Specify a numeric tag to identify this field.

Used By: [SWIFT Data Types](#)

Usage: Click in the right-hand cell to edit this field.

Transform Properties



The Transform section of the Properties window only displays when you click input or output models of transformations displayed in the Explorer window. it contains just one property:

- [Required](#)

Required

Description: Sets mandatory (where there are multiple components) inputs and outputs on a transform.


Used By: [Transformations](#)

Usage: Select the check box to set the component as mandatory.

Setting Properties using XPath

You can define some properties in ADS Designer using XPath. For example, an element's [Length Constraint](#) and [Cardinality Constraint](#) properties are both set using XPath.

You can set the following options when setting properties using XPath:

- **Ignore Document Node** - This indicates whether to ignore the document node in the XPath expression. If this property is selected, the XPath expression should not include the name of the component whose type has this rule applied to it. This is the preferred approach since the type to which this rule is applied might be used by multiple components.
- **Namespace Aware** - This indicates whether to take account of namespaces in the XPath expression. If you select this option in the Namespaces tab, the XPath expression must use the prefixes defined in the namespaces table below. To add a namespace, click the  icon. If this option is not selected, the resulting XPath behaves as if elementFormDefault and attributeFormDefault were both set to 'unqualified'.

Verifying, Building, and Running

Once you have created, or downloaded, a data model or transformation, you can do any of the following:

- [Verify](#)
- [Build](#)
- [Run](#)

Verifying Data Models and Transformations

Verifying a [data model](#), a model [component](#), or a [transformation](#) checks whether the metadata is well formed and valid within an ADS context. It identifies any problems with the metadata that will cause a [build](#) to fail.

For example, verification will fail if:

- A referenced data model is missing from the disk
- The type of components to be transformed are incompatible, for example, transforming a string to an int

To verify a data model or a transformation, right-click the file in the Project window and select **Verify File(s)**.

To verify a model component, right-click the component in the Explorer window and select **Verify Component(s)**.

Building Models, Components, and Transformations

ADS enables you to build (or deploy) the following:

- [Data models](#)
- [Model components](#)
- [Transformations](#)
- Entire [projects](#)

Building performs the following tasks:

- [Verifies](#) the data model or transformation structure
- Generates Java code from data models, components, and transformations
- Compiles the generated code
- Packages the generated code into one or more JAR files
- Creates an Ant build script containing typical Ant targets such as clean, jar, and javadoc

To build a data model or a transformation, right-click the .dod or .tfd file in the Project window and select **Build File(s)**.

To build a model component, right-click the component in the Explorer window and select **Build Component(s)**.

To build all the data models and transformations in a project, right-click the project (.iop) file in the Project window and select **Build Project**.

Running Model Components and Transformations

The Run wizard allows you to:

- Run your transformations against valid data to check that outputs are as expected
- Validate data against a data model that you know to be correct

Run Configurations

Run configurations describe how to launch the Run Wizard for a particular data model or transformation. These configurations mean you don't have to set up the run details every time you launch the Wizard for the same component.

Camel Integration


The Run wizard also allows you to integrate with Apache Camel so that you can implement routing and mediation rules for your data. You can use the wizard to combine different data formats and Camel-supported transport protocols in a variety of ways that best suit the needs of your data solutions.

Creating a Run Configuration

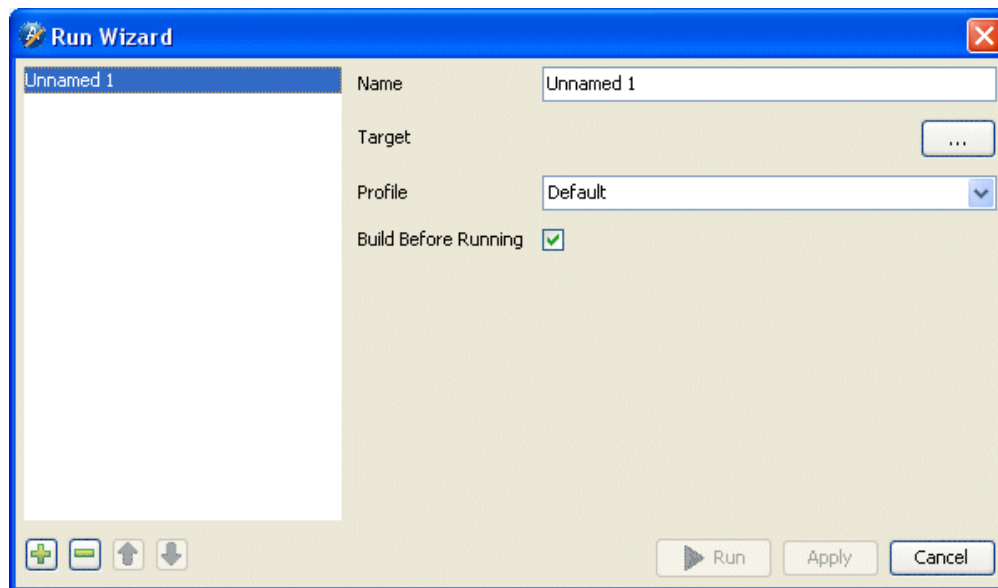
ADS Designer allows you to store a description of how to launch the Run Wizard for a particular data model or transformation in a *run configuration*.

To create a run configuration:


1. Do any of the following to open the Run Wizard:

- Select **Deploy > Run**.
- Click the  icon on the toolbar.
- Right-click a component name in the Explorer window and select Run Component.
Note: In this case, a component name can be the name of a complex type or element in a data model, or the name of a transformation.

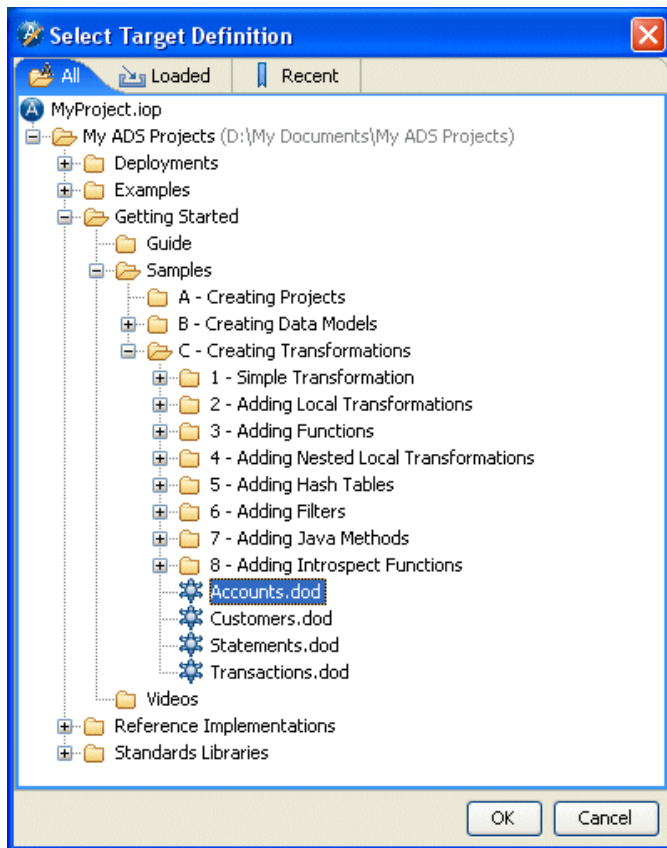
This opens the Run Wizard dialog.



Note: If you have right-clicked a component name in the Explorer window, the name of that component defaults as the configuration name automatically.

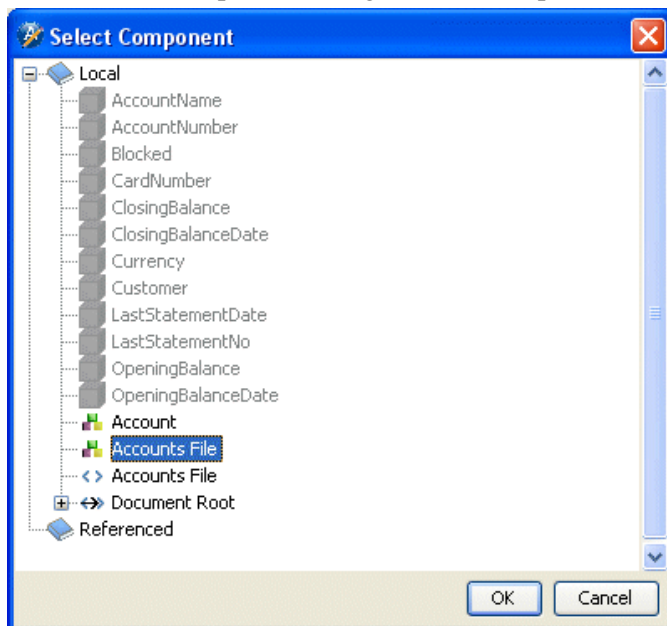
2. Type a unique name of the run configuration in the **Name** field.
3. Click the  button beside the **Target** field to select the component for which you want to set up this run configuration.

4. In the Select Target Definition dialog select a target data model or transformation.




Note: The target namespace of the selected data model or transformation is used to identify Java classes that are built (for example, `com.progress.artixdataservices.gettingstarted.account`). The root class is used to parse any data subsequently loaded into the object in the Run Wizard tab.


5. In the Select Component dialog, select a component to run and click **OK**.





Note: If you initially opened the Run Wizard dialog by right-clicking a component in the Explorer window, the value in the Target field defaults to that component automatically. You can change the selected component via the Select Target Definition panel.

6. The **Profile** field defaults to your default profile settings to determine the behavior of deployed code. If you have set up alternative profile settings, you can click the down arrow in this field to select the alternative settings.
7. Ensure that the **Build Before Running** check box is selected, if you want Designer to build instances of your object (component) before opening the Run Wizard tab. The root instance that is built will be used to parse any subsequent data you load into the object.
Clear this check box if you do not want to build instances of your object .
8. Click **Apply** to save any changes you have made on the dialog or click **Cancel** to close the dialog.

To create another run configuration, click the  icon and repeat steps 1-6 above.

To delete an existing run configuration, click the name of that configuration in the list and then click the  icon.

You can use the  and  icons to move configurations up and down the list.

When you wish to load and work with a particular configuration in a **Run** tab, click its name in the list and then click **Run**.

Working with Data Models in the Run Tab

The Run Wizard works off, and allows you to visualize, instances of classes built from data models. It enables:

- Data entry for populating an instance of a data object with manually-keyed values
- Validation
- Reading and writing of instance documents (that is, parsing and formatting of data into and out of instances of the built object)

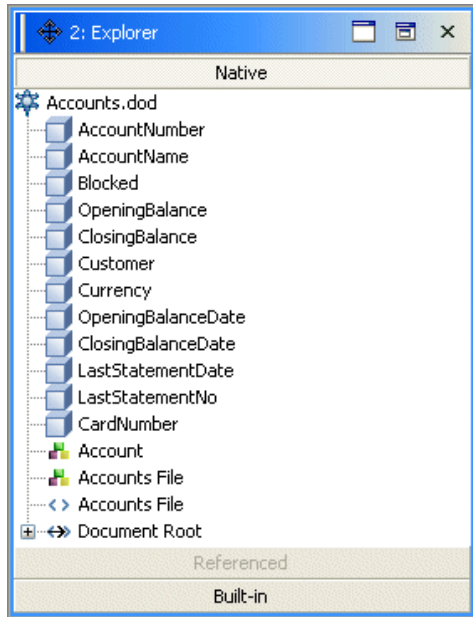
You can also use the Run Wizard to validate some data by reading it into a data model that you know to be valid. In such cases, you would not necessarily want to rebuild instances of your model.

The following topics are discussed in this section:

- [Loading Data into an Object](#)
- [Creating Object Instances](#)
- [Loading Model Changes within a Run Tab](#)
- [Setting Logging Levels](#)
- [Using the Text View](#)
- [Using Advanced Settings](#)

Loading Data into an Object

Consider the following sample data model:



The 'Accounts File' complex type has a structure as follows:

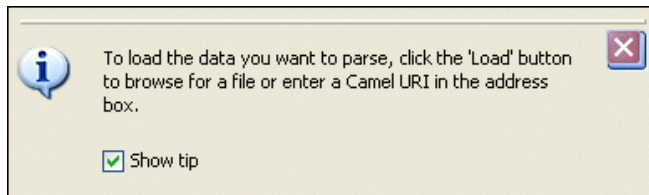
Component	Type	Cardinality	Size
Accounts File	Accounts File	77 - *	
Account	Account	1..*	77 - 107
AccountNumber	AccountNumber	1	12
AccountName	AccountName	1	20
Blocked	Blocked	1	1
OpeningBalance	OpeningBalance	1	1 - 16
ClosingBalance	ClosingBalance	1	1 - 16
Customer	Customer	1	6
Currency	Currency	1	3
OpeningBalanceDate	OpeningBalanceDate	1	1
ClosingBalanceDate	ClosingBalanceDate	1	1
LastStatementDate	LastStatementDate	1	1
LastStatementNo	LastStatementNo	1	12
CardNumber	CardNumber	1	16

This means that an 'Accounts File' object is made up of one or more 'Account' objects, each of which contains one and only one 'Account Number', 'Account Name', and so on.

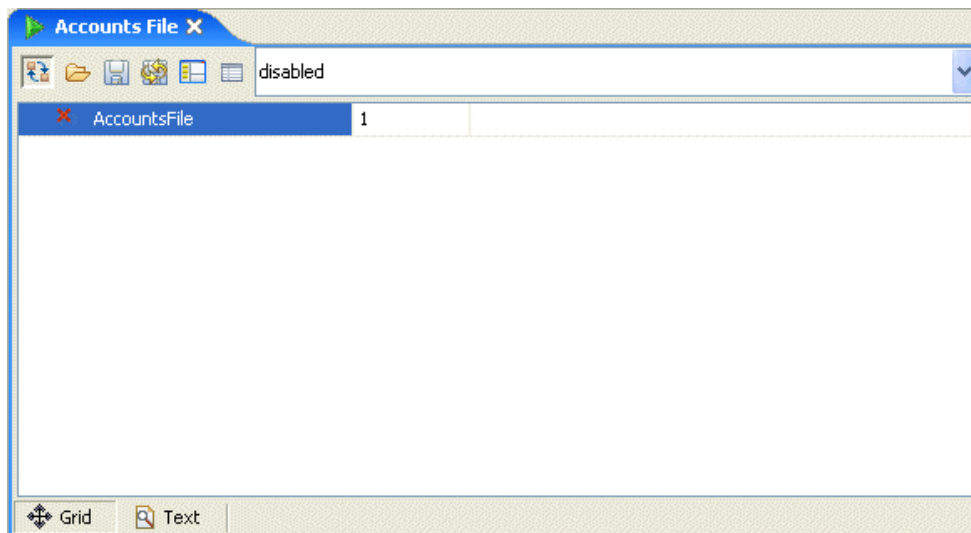
Loading an Object for the First Time

If your chosen run configuration on the Run Wizard dialog relates to a data model, clicking **Run** opens a tab for the related component. This tab will be used to show the structure of the deployed object. In this case, the tab name is based on the component name (or the name you specified for the run configuration) and also shows a ► icon to indicate that it is a Run Wizard tab.

When Designer is about to open a tab for an object that you have not loaded before, it displays the following dialog to prompt you to load some data into the object:



The class representing the deployed object is loaded into memory and displayed in its 'empty' state as shown below with a red X. This is because you have not yet loaded any data into instances of the deployed object.



In this case, the Messages window will contain a Run component name tab that is empty.

If you open the Validation window, it will display a validation error that 0 elements are set.

If you checked the **Build before Running** check box on the Run Wizard dialog, log4j messages relating to the building of Java classes for the object are displayed at the bottom of the screen during the build process.

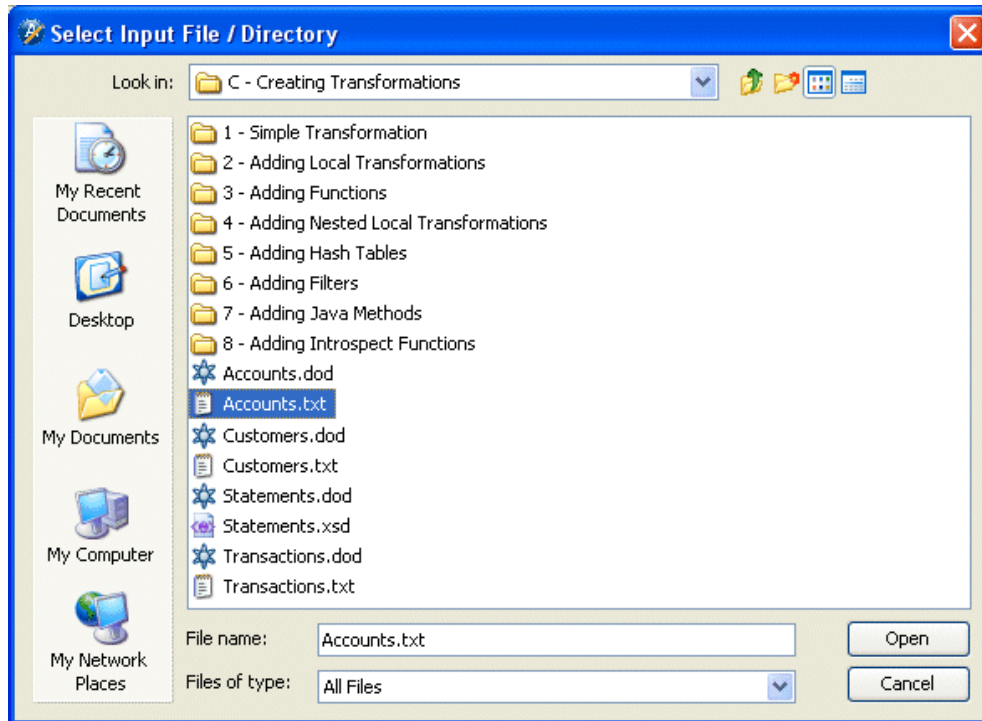
To display build messages in a tab in the Messages window, right-click the relevant component in the Explorer window and select **Build Component(s)**. This creates a Building tab in the Messages window that contains the relevant log4j messages.

Loading the Sample Data

To load data into an object:

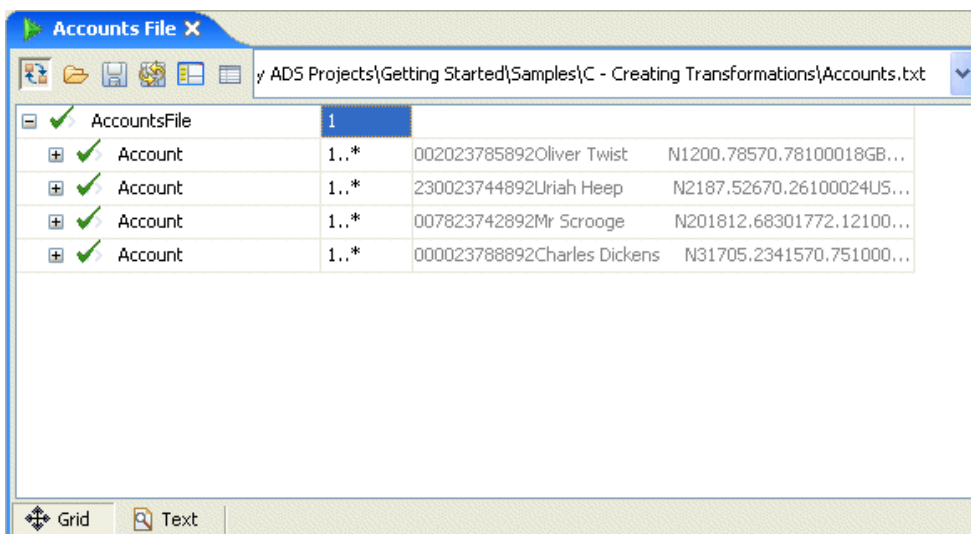
Click the  (Load) icon in a Run tab opens the Select Input File / Directory dialog.

Browse to the data file that you want to load in order to create instances of your object.



Note: You can also select a directory rather than a file. In this case, the full path to that directory is displayed in the File name field and all files in the selected directory are read into the model in turn. Files in any subdirectories are also read into the model by default, but you can disable this feature in your ADS [Preferences](#) settings.

When a selected data file is read into a model, ADS creates instances of the model (provided the model is accurate) and you can see in the tab that the data has been successfully parsed. For example:

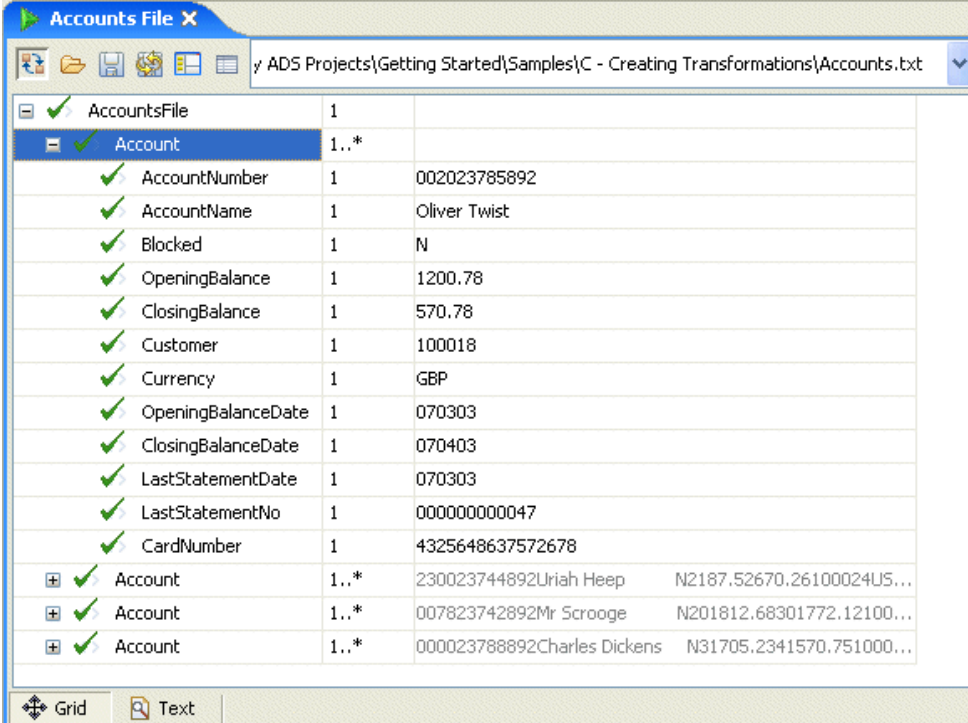


In this case:

- The Messages window indicates that validation has been successful.
- The Validation window provides statistics about the number of valid components (that is, elements, attributes, enumerations, and rules) relating to the object.
- The Properties window displays values for the number of Namespaces, Comments, Processing Instructions, and Client Properties relating to the object.

Note: If the data model is not accurate, Designer displays parsing errors. In such cases, you must fix the relevant errors and then try to load your data again.

You can expand each node within the tab to view all the record instances that have been parsed. For example:





The screenshot shows a window titled 'Accounts File' with a toolbar and a table of data. The table has columns for account details. The first row is expanded to show individual attributes for an account.

Account	Attributes	Values
AccountsFile	1	
Account	1..*	
Account	AccountNumber	002023785892
Account	AccountName	Oliver Twist
Account	Blocked	N
Account	OpeningBalance	1200.78
Account	ClosingBalance	570.78
Account	Customer	100018
Account	Currency	GBP
Account	OpeningBalanceDate	070303
Account	ClosingBalanceDate	070403
Account	LastStatementDate	070303
Account	LastStatementNo	000000000047
Account	CardNumber	4325648637572678
Account	Account	230023744892Uriah Heep N2187.52670.26100024US...
Account	Account	007823742892Mr Scrooge N201812.68301772.12100...
Account	Account	000023788892Charles Dickens N31705.2341570.751000...

Note: If you subsequently open this object again in another Run Wizard tab, the data that you previously loaded will be automatically reloaded.

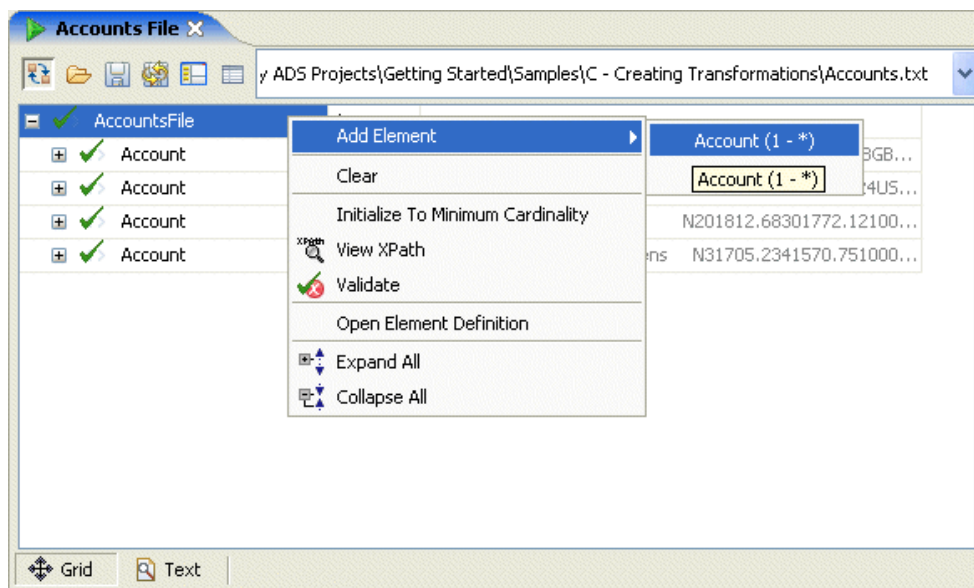
Loading Data Continuously

Enabling the  (Poll for Updates) icon in a Run tab makes the  icon act in a 'load continuously' manner. Designer automatically picks up any changes in the data file(s) that you are working with and reloads the modified data.

When you disable Poll for Updates, the  icon acts in a 'load instance' manner.

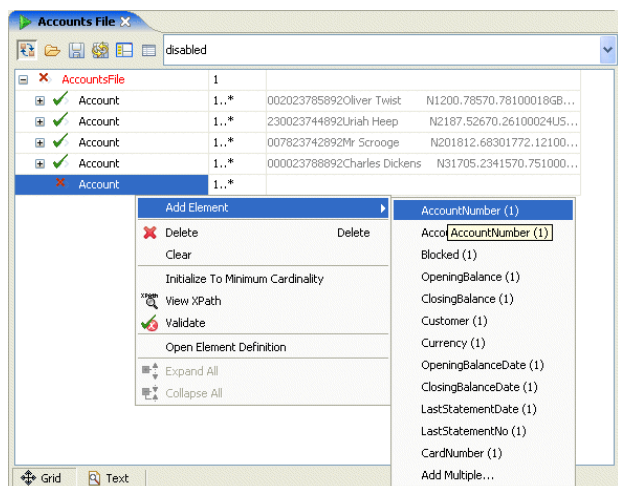
Creating Object Instances

You can add new instances of your data on-the-fly by right-clicking the object (component) name in the grid and selecting **Add Element** as shown.



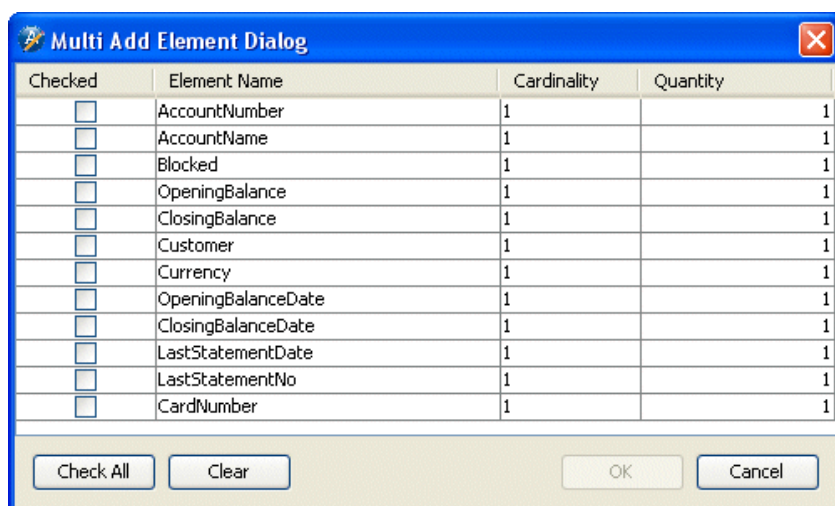
This allows you to add 'empty' child elements to the structure of the object. This section of the menu is dynamically built from the structure of the underlying element type. Therefore, based on the example shown, following the 'Add Element' arrow it reads 'Account 1 - *' because the only thing you can add to an 'AccountsFile' is an 'Account' with a 1-to-many cardinality.

Right-click the new child element name to add all the fields (types) that relate to that element and assign values to them.




Select **Add Multiple** to open the Multi Add Element dialog to add all the fields (types) simultaneously.

Click **Check All** to select all elements simultaneously and then click **OK** to add them as a new record instance in the tab.




Saving Data Instances

Click the  (Save Instance) icon in a Run tab to save to disk any data instances that are displayed in that Run tab.

Choose the file to which you want to save the new data instances in the Select Output File dialog.



Saving Instances Continuously

Click the  (Save Continuously) icon in a Run tab to have Designer automatically perform a batch save of instances from multiple different input files.


A typical use case is where you wish to load input data from multiple files and write all of those data instances to another format and/or another target transport.

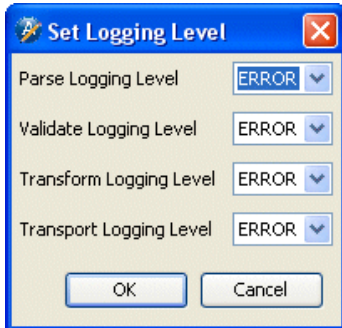
Choose the file or directory to which you want to save the new data instances in the Select Output File/Directory dialog.

Loading Model Changes

Clicking the  (Reload Active Run Configuration) icon on the toolbar allows you to reload an updated model into the currently open Run tab. A typical use case is where a model is different from a (valid) data file and you wish to update the model to match the data. If you already have a Run tab open for that model, clicking  saves you from having to open a new Run tab for the same component.

Setting Logging Levels

Clicking the  (Change Logging Levels) icon in a Run tab opens the Set Logging Level dialog:

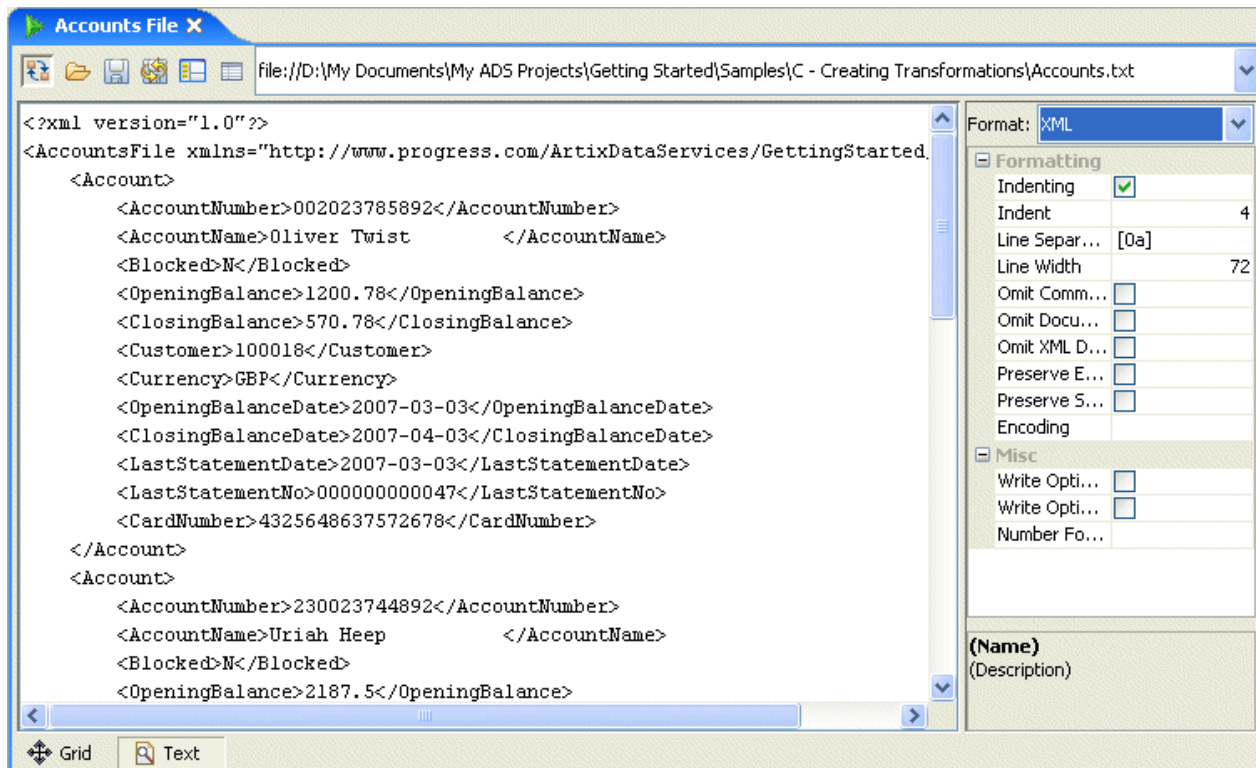


This allows you to set the logging level that you want to be output for your model parsing, model validation, transformations, and transports. The available logging levels are: Off, Fatal, Error, Warn, Info, Debug, and All.

For the Textual format the **Encoding** property allows you to select from a variety of encoding formats. If this is not set, UTF-8 will be used by default.

XML

You can view the input file in XML format. This allows you to be even more specific about the way the file is displayed. Altering the various formatting properties produces different results in the way the XML is displayed.



For the XML format you can set the following properties:

Indenting

Whether the text should be indented.

Indent

The size of the indent, if the text is to be indented.

Line Separator

Sets the line separator of characters. In the example above, the line separator is [0a] which is the Hex value for Line feed. To change this property, click and select the arrow to the right of the cell to open the Insert Character dialog.

Line Width

The maximum number of characters on a line. A value of 0 indicates that no line wrapping should occur.

Omit Comments

Whether comments should be omitted.

Omit Document Type

Whether DOCTYPE should be omitted.

Omit XML Declaration

Whether the XML declaration should be omitted.

Preserve Empty Attributes

Whether empty attributes should be preserved.

Preserve Space

Whether spaces should be preserved.

Encoding

The value of the encoding attribute that will be set in the XML header of the instance document. If this is not set, UTF-8 will be used by default.

For the XML format you can also set the following miscellaneous properties:

Write Optional Default Values

Whether the biz.c24.io.api.presentation.XMLSink class for this data model will write optional default element and attribute values.

Write Optional Fixed Values

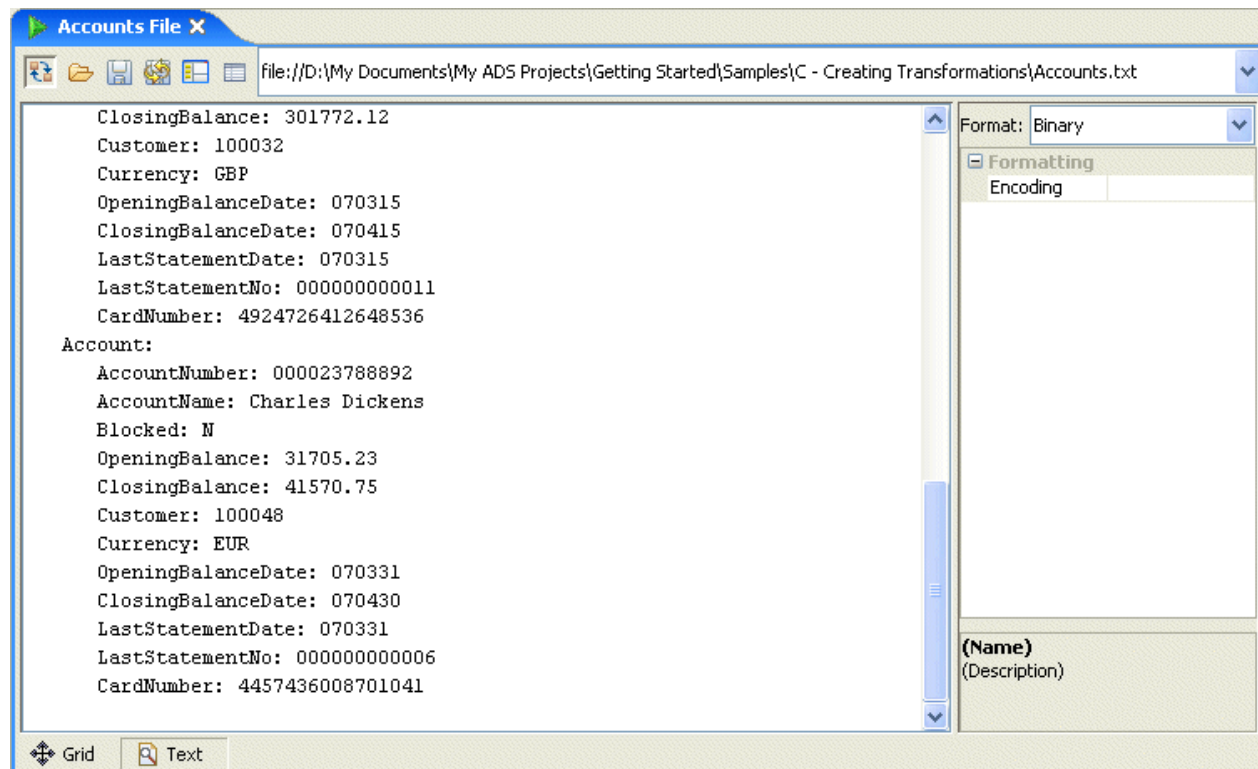
Whether the biz.c24.io.api.presentation.XMLSink class for this data model will write optional fixed values.

Number Format

Sets the decimal format object which is then used to format the numbers into strings. (This property is specified in the biz.c24.io.api.presentation.XMLSink class.)

Binary

You can view the input file in Binary format.

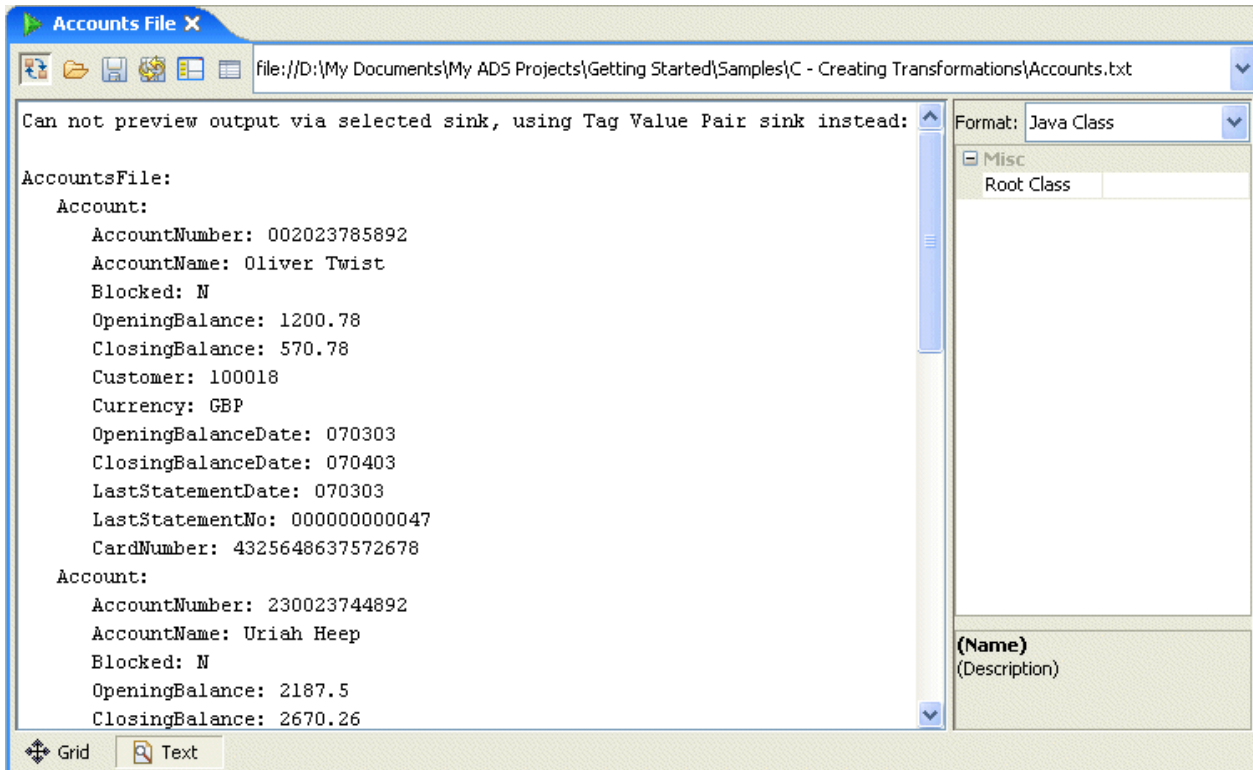


For the Binary format you can set the following properties:

- [Encoding](#)

Java Class

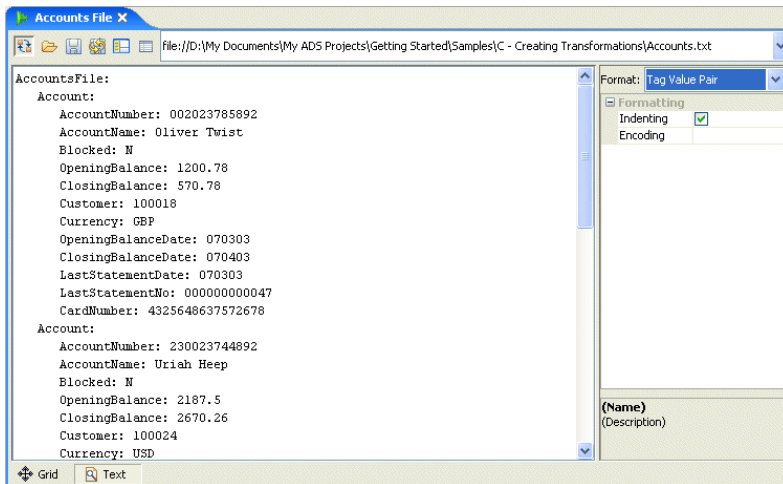
You can view the input file in Java Class format.



In the Java Class format, the **Root Class** property sets the root class to map and serialize complexDataObject information. Not all file can be viewed in this format. In this case, the Tag Value pair format is recommended.

Tag Value Pair

You can view the input file in Tag Value Pair format.

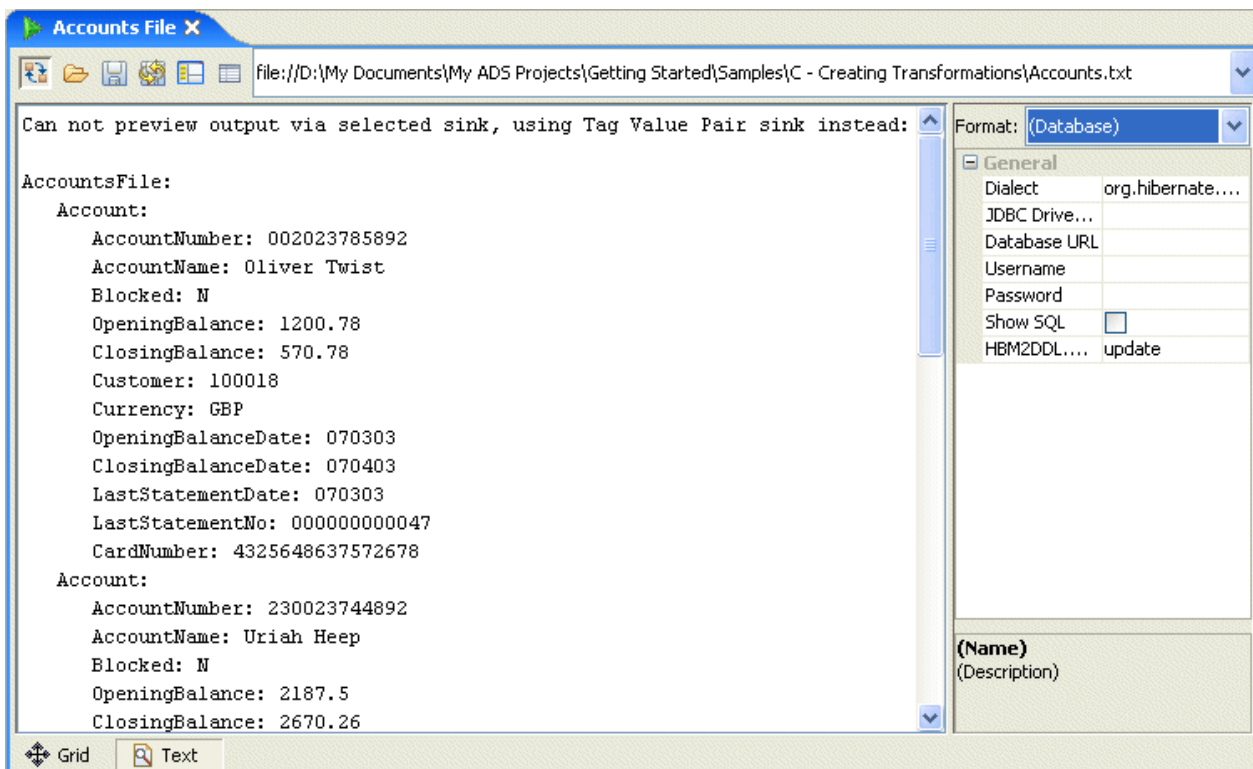


For the Tag Value Pair format you can set the following properties:

- [Indenting](#)
- [Encoding](#)

Database

You can view the input file in Database format. When using a database, Hibernate Mappings will be used to manage this functionality. Make sure that you have generated the Hibernate Mappings properly.



For the Database format you can set the following properties:

Dialect

The Hibernate dialect that will be used to communicate with the database.

JDBC Driver Class Name

The classname of the JDBC database driver.

Database URL

The URL of the database.

Username

The username with which the user will log into the database.

Password

The password that will be required to connect to the database.

Show SQL


Whether the SQL statements should be displayed.

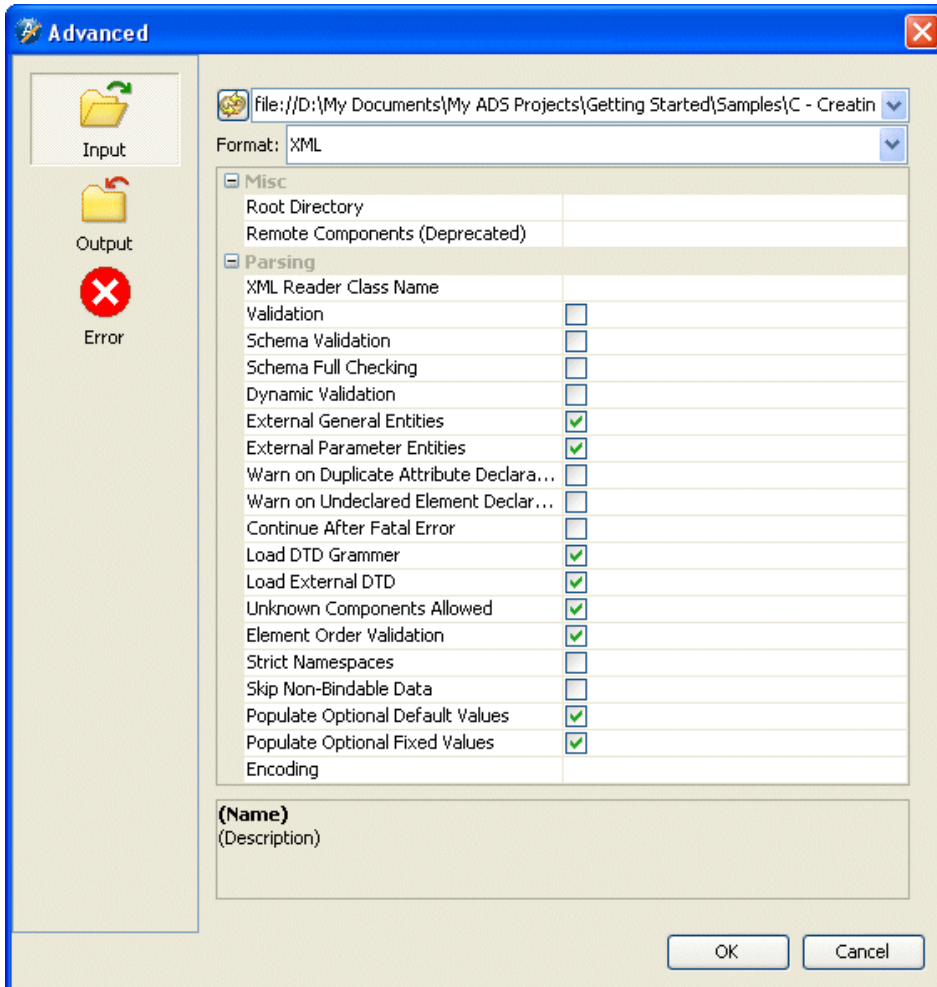
HBM2DDL.Auto

The Hibernate behavior for updating, creating, dropping, and validating table structures via DDL.

Note: An error such as 'No Persister for java.math.BigDecimal' means that your Hibernate Mappings have been incorrectly generated. Make sure that the [ID Generator Methods](#) property is properly set.

Using Advanced Settings

Clicking the  (Advanced) icon in a Run tab opens the Advanced dialog. This dialog allows you to configure settings that are used to retrieve and output your data.



Three panels are available. The Input panel relates to your input data, the Output panel relates to your valid output data, and the Error panel relates to your invalid output data. These panels allow you to select from a variety of different Camel-supported transports and data formats to work with, and a range of properties are available in each case for you to configure. Clicking on each property displays a context-sensitive description of that property at the bottom of the panel.

The top field on each panel allows you to select the Camel-supported transport protocol you want to use for your input or output data. A variety of transports are available. See <http://camel.apache.org/components.html> for more details of the supported Camel components.

Note: Support for Apache Camel integration in ADS is for use with design-time only. If you wish to use Apache Camel with ADS in a production environment, you must be using the Artix Java Message Broker or the Fuse Mediation Broker.

Click the down arrow in the Format field to select the format of your data. Possible data formats include:

- [Textual](#)
- [XML](#)
- [Binary](#)
- [Java Class](#)
- [Tag Value Pair](#) (output only)
- [SWIFT](#) (input only)
- [Crest ISO](#) (input only)
- [Database](#)

In each case, you can set various formatting properties for the format you choose. A context-sensitive description of each property is displayed at the bottom of the panel when you click that property.

Textual

The following properties relating to the Textual format are available for data inputs:

Look Ahead

The maximum number of characters to look ahead. The default is 16,384.

Look Behind

The maximum number of characters to look behind. The default is 16,384.

End of Data Required

Indicates whether parsed data can contain one and only one data instance.

Encoding

Allows you to select how characters will be encoded. If this is not set, UTF-8 is used by default.

And for output data, **Encoding** allows you to select how characters will be encoded. If this is not set, UTF-8 is used by default.

XML

The following properties relating to the XML format are available for data inputs:

Root Directory

The directory to which all relative filenames will be resolved. If this is left blank, the working directory of the JVM is used by default.

Remote Components

Class names of unknown elements, which might appear in place of an any or as part of a model that was being loaded without a defining element declaration.

XML Reader Class Name

The class name of the org.xml.sax.XMLReader implementation that should be used for XML parsing. If this is left blank, "org.apache.xerces.parsers.SAXParser" is used by default.

Validation

Check this to enable XML validation during parsing. If this is enabled, the document must specify a grammar. By default, validation is performed against the DTD.

Schema Validation

Check this to enable XML validation by XML schema during parsing. Validation errors will be reported only if the Validation check box has been checked.

Schema Full Checking

Check this to enable full schema grammar constraint checking.

Dynamic Validation

Check this to instruct the XML parser to validate the document only if a grammar is specified.

External General Entities

Check this to instruct the XML parser to include external general (text) entities.

External Parameter Entities

Check this to instruct the XML parser to include external parameter entities and the external DTD subset.

Warn on Duplicate Attribute Declaration

Check this to instruct the XML parser to raise a warning on finding duplicate attribute declarations.

Warn on Undeclared Element Declaration

Check this to instruct the XML parser to raise a warning if an element referenced in a content model is not declared.

Continue After Fatal Error

Check this to instruct the XML parser to continue processing after encountering a fatal error.

Load DTD Grammar

Check this to instruct the XML parser to load the DTD and use it to add default attributes and set attribute types when parsing.

Load External DTD

Check this to instruct the XML parser to load the external DTD completely.

Unknown Components Allowed

Indicates whether unknown components are allowed to appear in incoming XML.

Element Order Validation

Indicates whether element order validation is allowed.

Strict Namespaces

Indicates whether instance documents in namespaces (including the default namespace), other than that defined by the argument supplied to the readObject(Element) method, are accepted.

Skip Non-Bindable Data

Whether non-bindable data other than that defined in the model will be unmarshalled.

Populate Optional Default Values

Indicates whether the default values are populated to the elements/attributes if they were specified as optional.

Populate Optional Fixed Values

Indicates whether the fixed values are populated to the elements/attributes if they were specified as optional.

Encoding

Allows you to select how characters will be encoded. If this is not set, UTF-8 is used by default.

The following properties relating to the XML format are available for data outputs:

Indenting

Indicates whether output is to be indented.

Indent

The size of the indent, if the Indenting check box has been checked.

Line Separator

Allows you to select the line separator characters.

Line Width

The maximum number of characters on a line. (A value of 0 indicates that no line wrapping should occur.)

Omit Comments

Indicates whether comments are to be omitted from the output.

Omit Document Type

Indicates whether the DOCTYPE is to be omitted from the output.

Omit XML Declaration

Indicates whether the XML declaration is to be omitted from the output.

Preserve Empty Attributes

Indicates whether empty attributes are to be preserved in the output.

Preserve Space

Indicates whether spaces are to be preserved in the output.

Encoding

Allows you to select how characters will be encoded in the output. If this is not set, UTF-8 is used by default.

Write Optional Default Values

Indicates whether optional default element and attribute values will be produced.

Write Optional Fixed Values

Indicates whether optional fixed element and attribute values will be produced.

Number Format

The decimal format object that is used to format numbers into strings.

Binary

The following properties relating to the Binary format are available for data inputs:

Look Ahead

The maximum number of bytes to look ahead. The default is 8,192.

Look Behind

The maximum number of bytes to look behind. The default is 8,192.

End of Data Required

Indicates whether the stream must be emptied for the read to succeed. This check box should be left unchecked (that is, disabled) if the readObject() method is to be called more than once on the same stream or if the calling code is not concerned about additional data such as blank lines. This check box should be checked if you are expecting the stream to contain one and only one data instance.

Encoding

Allows you to select how string data (if any) will be encoded. If this is not set, UTF-8 is used by default.

And for data outputs, **Encoding** allows you to select how characters will be encoded. If this is not set, UTF-8 is used by default.

Java Class

In the Java Class format the **Root Class** property specifies the root class into which a ComplexDataObject is to be mapped and serialized.

Tag Value Pair

The following properties relating to the Tag Value Pair format are available for data outputs:

Indenting

Indicates whether output is to be indented.

Encoding

Allows you to select how characters will be encoded. If this is not set, UTF-8 is used by default.

SWIFT

In the SWIFT format **Base Package Name** specifies the SWIFT base package name.

Crest ISO

In the Crest ISO format **Base Package Name** specifies the Crest ISO base package name.

Database

The following properties relating to the Database format are available for data inputs and outputs:

Dialect

The Hibernate dialect used to communicate with the database.

JDBC Driver Class Name

The classname of the JDBC database driver.

Database URL

The URL of the database.

Username

The valid user name with which you can log into the database.

Password

The password with which you can connect to the database.

Show SQL

Whether SQL statements are to be printed to the console.

HBM2DDL.Auto

Allows you to select the Hibernate behavior for updating, creating and dropping table structures via DDL.

Working with Transformations in the Run Tab


The Run Wizard provides a way of testing that your transformations are accurate, by reading valid data into a transformation and generating Java class instances of it, thus ensuring that the relationships between data inputs and outputs are as expected.

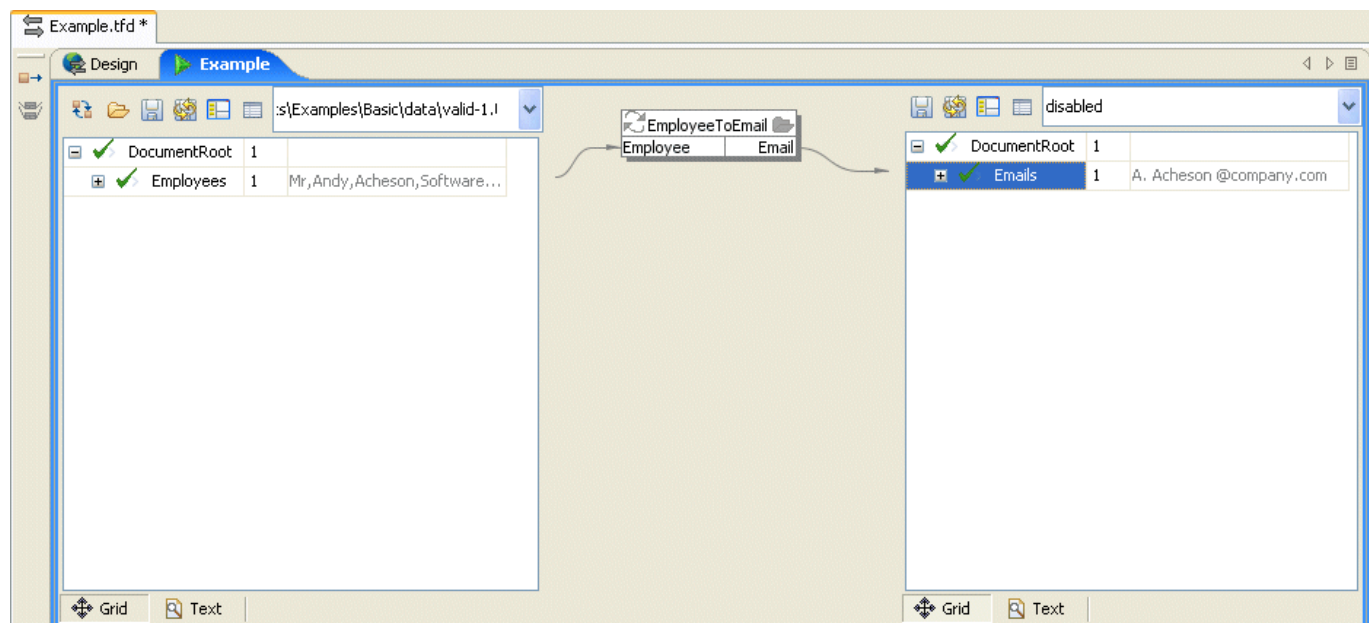
It also provides a simple yet powerful means of integrating with the Apache Camel-based Artix Router to allow you to specify various combinations of data formats and transports that best suit your data processing needs, It also allows you to set related properties for each type of format.

The following topics are discussed in this section:

- [Loading a Transformation](#)
- [Loading Input Data](#)
- [Adding and Saving New Input Instances](#)
- [Saving Instances Continuously](#)
- [Changing Logging Levels](#)
- [Setting Advanced Presentation Settings](#)
- [Selecting the Relevant Camel Transports](#)
- [Viewing Different Data Formats](#)
- [Running the Transformation](#)

Loading a Transformation

If your chosen run configuration on the Run Wizard dialog relates to a transformation, clicking Run opens a Run tab (with a  icon) within the main tab for the transformation. This tab will be divided into three sections showing your input model(s) on the left, your output model(s) on the right, and the translations between them in the middle.




The Run tab is used to show the structure of the deployed transformation. Each input and output model that comprises the transformation is shown within its own pane in the Run tab.

Loading Input Data


If you have already loaded data into the input models that comprise the transformation, that data will be automatically reloaded when the Run tab is opened. If you have not yet loaded data into one or more of the input models that comprise the transformation, the class representing that particular object will be loaded into memory and displayed in its 'empty' state with a red X.

The pane for each input model includes a  icon to allow you to [load data](#), if necessary. The pane for each input model also includes a  icon to allow you to [load data continuously](#).

Adding and Saving New Input Instances

The pane for each input and output model includes a  icon to allow you to [save instances to disk](#) after you have finished [creating instances of your data](#) within the Run tab.


Saving Instances Continuously

The pane for each input and output model includes a  icon to allow you to [automatically perform a batch save of instances](#) after loading multiple different input files. The real power of this feature can be seen in transformations where you wish to load multiple input files in a particular format and from a particular transport and, with one click, you can simultaneously convert and write them to another format and target transport.

Changing Logging Levels

The pane for each input and output model includes a  icon to allow you to [set the level of logging information](#) you want to be output for your transformations.

Setting Advanced Presentation Settings

The pane for each input and output model includes a  icon to allow you to set [advanced settings](#) for the data format relating to that model.

Selecting the Relevant Camel Transports


The pane for each input and output model allows you to select the Camel-supported transport protocol you want to use for that model. A variety of transports are available. See <http://camel.apache.org/components.html> for more details of the supported Camel components.

Note: Support for Apache Camel integration in ADS is for use with design-time only. If you wish to use Apache Camel with ADS in a production environment, you must be using the Artix Java Message Broker or the FUSE Mediation Broker.

Viewing Different Data Formats

The pane for each input and output model includes both a Grid and [Text](#) pane to allow you to view your data in different formats.

Running the Transformation

Click the  icon on the Transformations palette to run a transformation. The Messages window displays log4j messages related to deployments and to the loading and saving of data.

Packaging

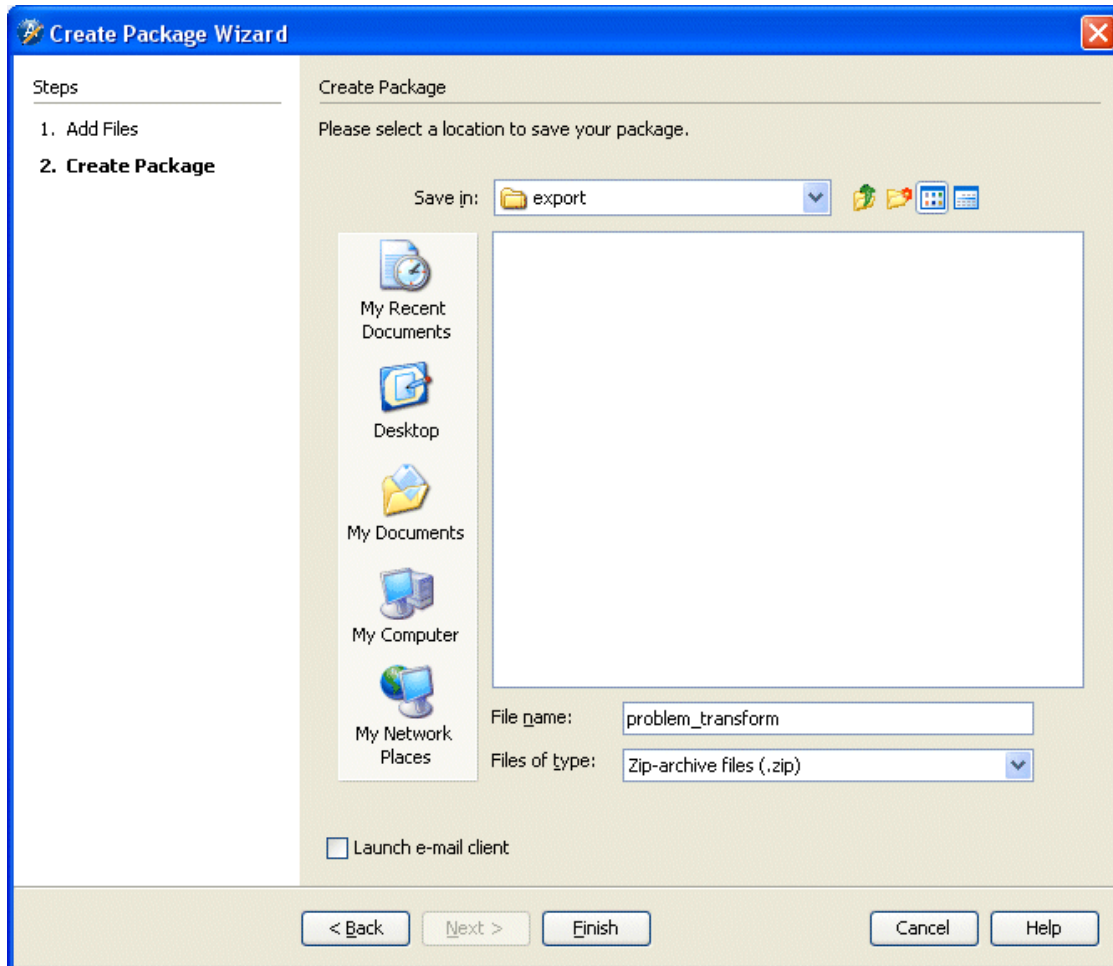
The packaging feature in ADS Designer allows you to:

- [Create a package](#) of data models and transformations that you can store or send to a third party. Packages are created in ZIP format and retain the folder structure of the source.
- [Unpack](#) a ZIP file into your My ADS Projects folder.

Creating a Package

To create a package:

1. Select **File > Packaging > Create Package**.
2. In the Create Package Wizard, use the Add Files panel to browse to the files that you want to add to the package, then click **Next**.
3. In the Create Package panel, choose a destination folder and filename for the ZIP archive.



Select the **Launch e-mail client** checkbox to launch your e-mail client after the package is created. You can then attach the generated package to an e-mail message.

4. Click **Finish**.

Unpacking a Package

To unpack a ZIP archive into your My ADS Projects directory:

1. Select **File > Packaging > Unpack**.
2. In the Unpack Wizard, use the Open Pack panel to select the ZIP archive that you want to unpack, then click **Next**.
3. In the Model Directory panel, select the folder where you want to unpack the archive.
4. Click **Finish**.

The Diff Tool

The Diff Tool is a tool for comparing different versions of data models in order to ascertain any differences between the two. It is most useful where versioned data models are being used.

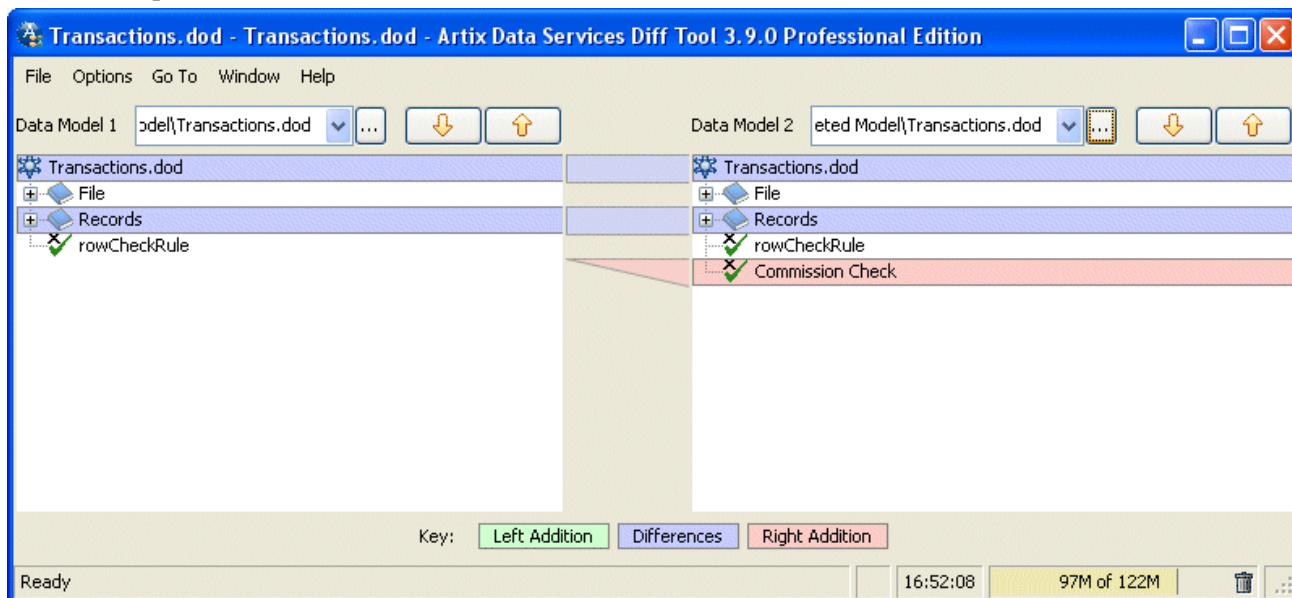
The following topics are discussed in this section:

- [Loading and Working with the Diff Tool](#)
- [File Menu](#)
- [Options Menu](#)
- [Go To Menu](#)
- [Window Menu](#)
- [Context Menu](#)

Loading and Working with The Diff Tool

To open the Diff Tool, do either of the following:

- Select **Tools > Launch Diff Tool** from the menu bar. This opens an empty version of the Diff Tool.
- Shift-click the two models that are to be compared in the Project window, right-click and select **Open In Diff Tool**. This opens the Diff Tool with the two models already loaded and compared, as shown in the following screen example.



File Menu

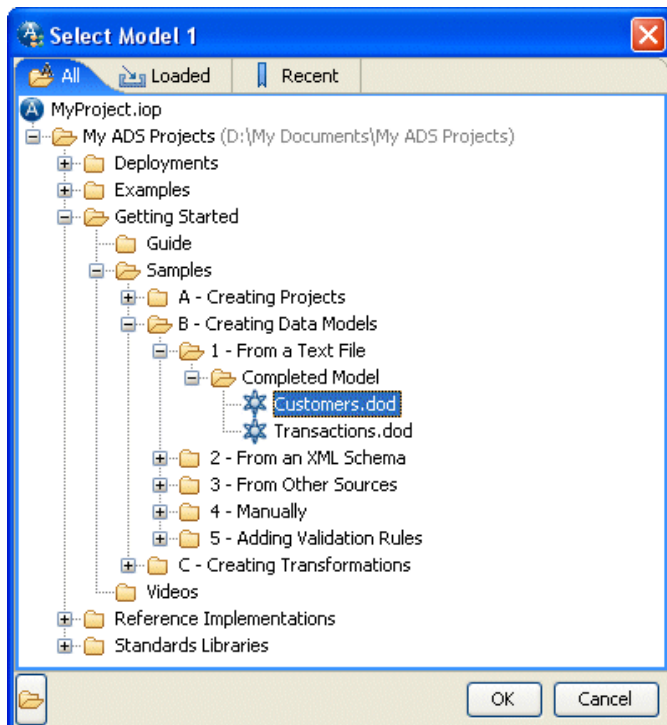
The File menu provides access to functionality for setting up the parameters to the compare operation and deciding what to merge. It contains the following items:

- [Set Model 1](#)
- [Set Model 2](#)
- [Diff Left Selection](#)
- [Diff Right Selection](#)
- [Merge Left Selection](#)
- [Merge Right Selection](#)
- [Exit](#)

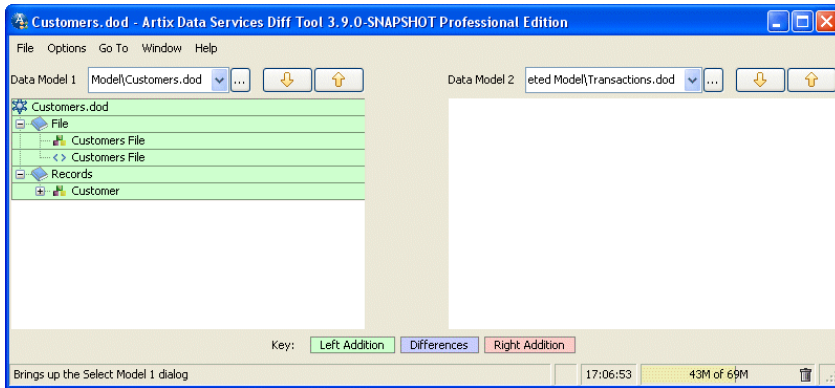
Set Model 1

This opens the Select Model 1 dialog. This dialog contains the following tabs:

- The Recent tab allows you to select a data model from a list of recently opened models.
- The Loaded tab allows you to select a data model from a list of models currently loaded in memory.
- The All tab allows you to select any data model from the path(s) for the current project.



Click the model you wish to select and then click **OK**. After selecting a model, its full path is displayed in the Data Model 1 field and the model itself is loaded into the left-hand pane of the Diff Tool window, as shown in the following example:

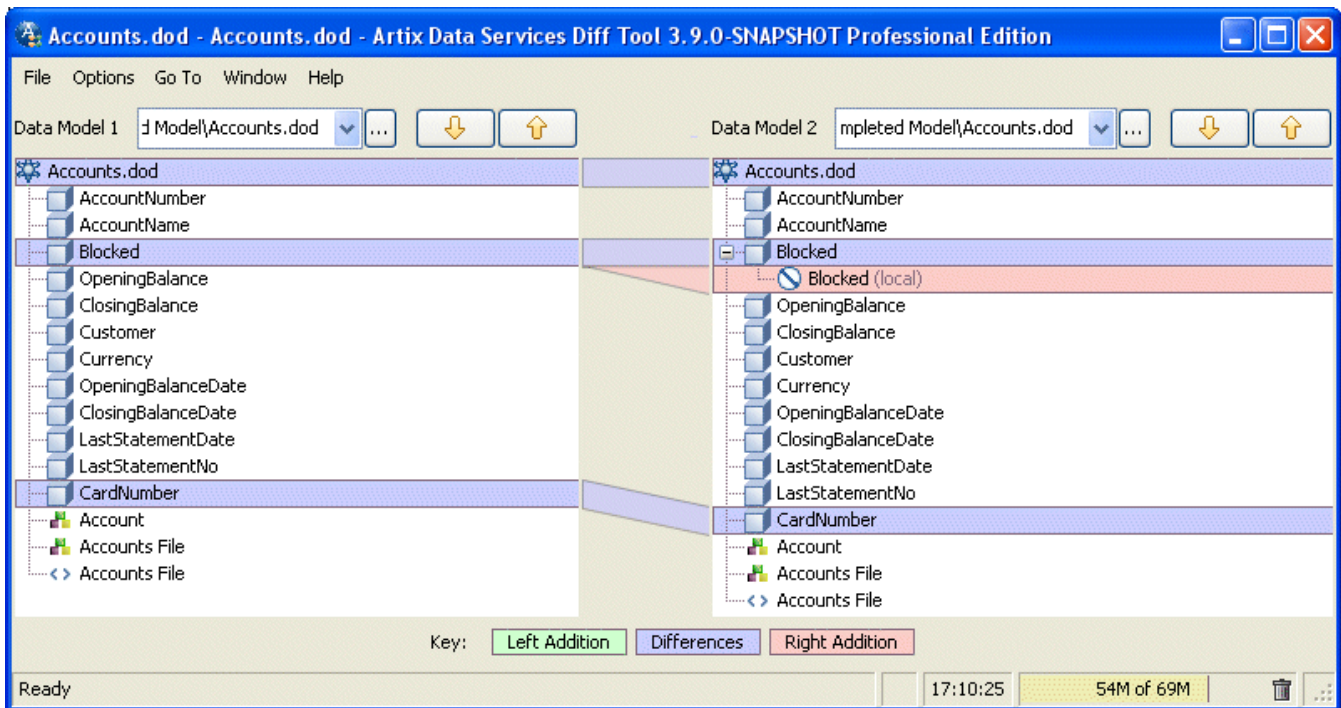


Set Model 2

This opens the Select Model 2 dialog. This dialog contains the following tabs:

- The Recent tab allows you to select a data model from a list of recently opened models.
- The Loaded tab allows you to select a data model from a list of models currently loaded in memory.
- The All tab allows you to select any data model from the path(s) for the current project.

Click the model you wish to select and then click **OK**. After selecting a model, its full path is displayed in the Data Model 2 field and the model itself is loaded into the right-hand pane of the Diff Tool window. If a model has already been loaded in the Data Model 1 pane, the models are compared as shown in the following screen example.



Diff Left Selection

This differentiates the model/component in the left-hand pane against the model/component in the right-hand pane.

Diff Right Selection

This differentiates the model/component in the right-hand pane against the model/component in the left-hand pane.

Merge Left Selection

This merges the selected node in the tree with its opposite number in the right-hand pane.

Merge Right Selection

This merges the selected node in the tree with its opposite number in the left-hand pane.

Exit

This closes the Diff Tool.

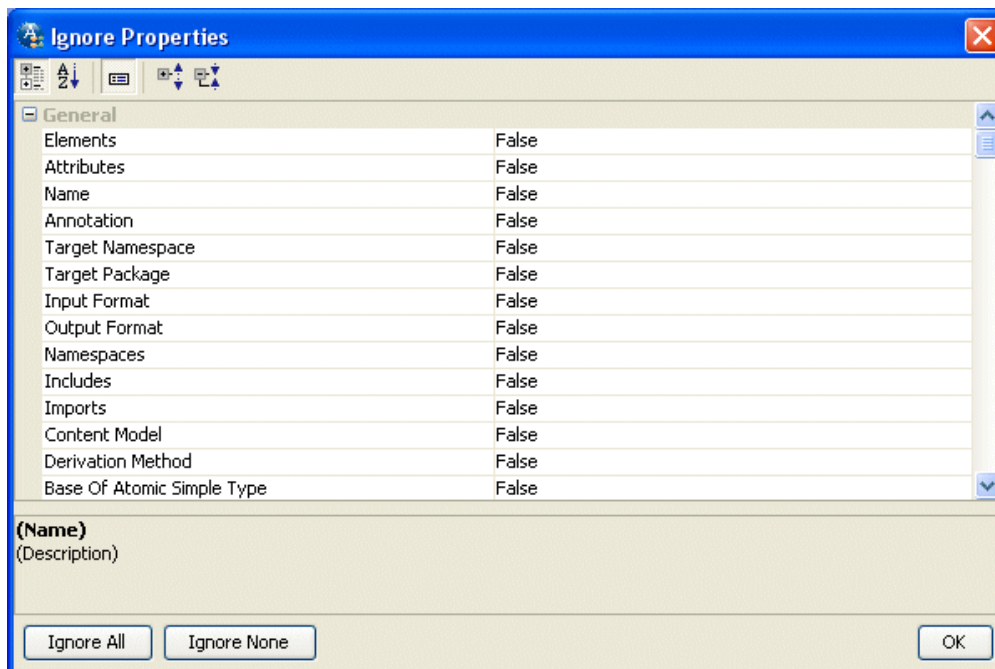
Options Menu

The Options menu allows you to specify what content should be compared and allows you to refresh the Diff Tool to reflect changes to what is to be compared in the view. It contains the following items:

- [Properties](#)
- [Refresh](#)

Properties

This opens the Ignore Properties dialog which determines what components of the two data models are to be checked during the comparison process. Each property is a boolean setting.



Refresh

This reruns the comparison analysis on the selected models in the light of any changes.

Go To Menu

The Go To menu includes options for navigating around the trees in the 2 windows. It contains the following items:

- [Next Left](#)
- [Previous Left](#)
- [Next Right](#)
- [Previous Right](#)

Next Left

This moves the focus to the next component that is to the left of the currently selected component

Previous Left

This moves the focus to the previously selected component to the left of the currently selected component

Next Right

This moves the focus to the next component that is to the right of the currently selected component

Previous Right

This moves the focus to the previously selected component to the right of the currently selected component

Window Menu

The Window menu contains functionality related to the layout of the Diff Tool. It includes the following items:

- [Save Layout](#)
- [Load layout](#)
- [Reset Layout](#)
- [Toggle Auto Hide](#)

Help Menu

The Help menu contains the following items:

- [Install Plugin](#)
- [Home Page](#)
- [User Guide](#)
- [What's This?](#)
- [Documentation](#)
- [View License Agreement](#)
- [View System Properties](#)
- [Standards Libraries](#)
- [Technical Support](#)
- [Contact Us](#)
- [About](#)

Context Menu

Within the Diff Tools you can bring up a context menu by right-clicking a node in either of the trees. It includes the following items:

- [Merge Left](#)
- [Merge Right](#)
- [Diff Component](#)

Merge Left

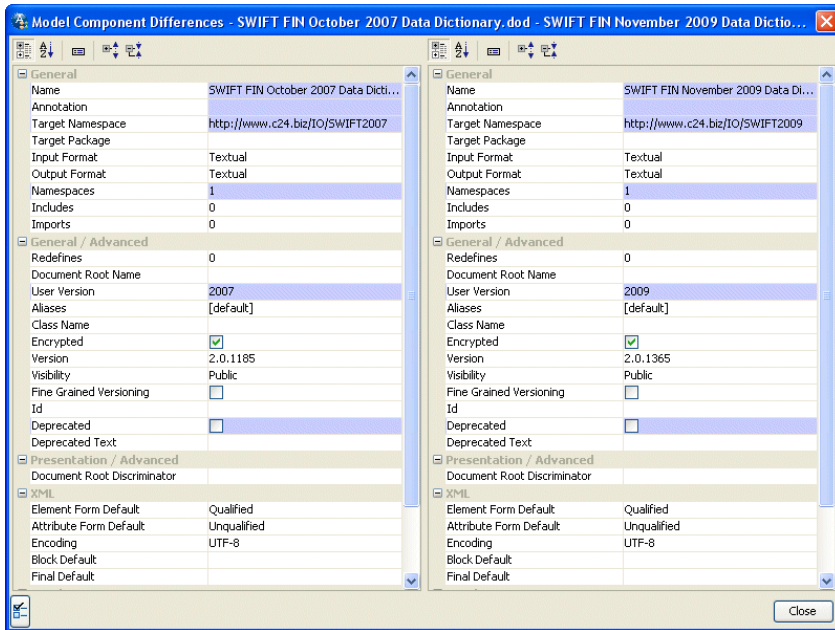
This is greyed out in the right-hand pane. It overwrites content in the right-hand model with the content in the left-hand model.

Merge Right

This is greyed out in the left-hand pane. It overwrites content in the left-hand model with the content in the right-hand model.

Diff Component

This drills down on the selected data component and compares its internal properties with those of its opposite number in a new Model Component Differences dialog.



Examples and Reference Implementations

ADS ships with a number of samples to help get you started with the product. These samples consist of:

- [Examples](#)
- [Reference Implementations](#)

You can download examples and reference implementations from the Designer [Home page](#).

We recommend that you work through at least the Basic example before starting to use ADS for your own purposes. See [Running the Samples](#) for details.

Examples

The examples are downloaded to the /My ADS Projects/examples directory.

The following examples demonstrate general ADS functionality:

- Basic - Demonstrates core ADS [functionality](#) using input models and data that are not tied to a particular financial standard.
- CXF - Demonstrates integration between the ADS runtime and Apache CXF (Artix Java)
- Camel - Demonstrates integration between the ADS runtime and Apache Camel (FUSE Mediation Router)

In addition, if you download the following standards libraries from the [Home page](#), accompanying examples, including project files and batch scripts, are added to the My ADS Projects/examples folder.

- SWIFTNet Fin
- FIX
- FpML
- ISO20022/Unifi
- SEPA

The standards library examples demonstrate the same [functionality](#) as the basic example, but they use input models and data that adhere to a particular standard.

Reference Implementations

Reference implementations demonstrate the same [functionality](#) as the examples, but can be run without having ADS Designer installed.

The reference implementations are downloaded to My ADS Projects/Reference Implementations.

Running the Samples

Apart from the Camel and CXF examples, you can run each example and reference implementation using the supplied Windows batch files or UNIX shell scripts.

Each batch file or shell script demonstrates a particular piece of ADS [functionality](#).

You can also run the samples by running Ant with the relevant target, as specified in the sample's build.xml file. (Ant is the only way to run the CXF and Camel examples.)

Note: To run the examples that feature the use of the ADS financial message packs like SWIFT, FpML and UNIFI, you need a license for these products.

Building the Example Code

Before you run an example, you must first build the source code from the data models and transformations used in each example by doing one of the following:

- Run ant deploy from the root of the relevant example folder in a command prompt.
- Open the project (.iop) file for the example under My ADS Projects/examples and [build](#) the project from within ADS Designer.

Note: Reference implementations are pre-built; you do not need to build them before running them.

Sample Directory Structure

The subdirectory under the examples and Reference Implementations folders contains the following directory structure:

- run*.bat - Windows batch files for running each example
- run*.sh - UNIX shell scripts for for running each example
- build.xml - Ant build script for building code from the example data models and transforms, and running each example
- log4j.properties - Apache log4j properties file for the configuration of the ADS API logging for the logging example
- src - a source tree of Java classes for demonstrating each of the generic functionality examples plus any Standards Library-specific classes
- model - (examples only) contains the example ADS data models and transformations used by each example. Where standards library models are used (for example, SWIFT), the models are referenced from the relevant standards library folder under My ADS Projects/Standards Libraries
- data - contains parseable/unparseable and valid/invalid data samples for use with each example set
- lib - contains all .jar files needed to compile the generated model code and the example classes
- docs - Javadoc for the source code generated from the example data models is added to this directory
- build - contains all compiled classes

Demonstrated Functionality

Each example and reference implementation is structured in a similar way to demonstrate the following generic functionality:

Example	Shows how to ...
Batching	Query a large file in batches. (Basic only)
Bean Query	Use bean methods to query data from a ComplexDataObject.
Benchmark	Benchmark the performance of the ComplexDataObject.
Clone	Make exact copies of a complex data object.
Database	Create a database schema from a ComplexDataObject using Hibernate and write the data loaded in the ComplexDataObject into a hsql database and reading the data back out again. This example uses the popular open source Java HSQLDB database which ships with ADS, there is no need to have a DBMS installed to run this example.
Domain Constraints	Validate against domain constraints. (Basic only)
Generic Iterator Query	Recursively query a ComplexDataObject to return all data associated with a regular expression.
Generic Loop Query	Recursively query a ComplexDataObject to return all data for a named element.
Java Class	Convert a random Java class into an ADS ComplexDataObject in Java code. (Basic only)
Logging	Set up logging for the parsing and validating of data into a ComplexDataObject.
Parse	Read data from a file and parse it into a ComplexDataObject. Also shows what happens if the data cannot be parsed.
To Tag Value Pair	Convert the presentation of data from the input format into Tag Value Pair format.
To XML	Convert the presentation of data from the input format into XML format.
Transform	Transform data from the structure based on the input data model to the structure based on the output data model using an ADS Transformation.

SWIFT Code validator	Use the SWIFT country, currency and BIC code validators (SWIFTNet Fin only).
SWIFT PreParser	Use the ADS API SwiftPreparser functionality to determine the type of a SWIFT message without knowing anything about it (SWIFTNet Fin only).
SWIFT Message	Use the ADS API SwiftMessage functionality to query a SWIFT message about its details, such as is outgoing, incoming, ACK or NAK, etc (SWIFTNet Fin only).
Validate	Validate data loaded into a ComplexDataObject against its validation rules and how to handle and interrogate validation exceptions for information about why data was invalid.
XPath	Use the ADS API to retrieve data from a ComplexDataObject using an XPath query.
XQuery	Use ADS to retrieve data from XML data
XSLT	Transform XML documents to other XML or human-readable documents.

Standards Libraries

ADS Standards Libraries are specialized components for specific business scenarios, such as payments processing, OTC derivatives, and corporate actions.

Each library implementation includes the industry standard message syntax, validation rules and test cases of valid and invalid data.

Progress Software maintains these libraries and issues updates to support the latest release of each standard.

Support is provided for over 100 financial messaging standards implementations across 22 standards bodies, offering customers rich, out-of-the-box support for all their financial messaging data services requirements.

Downloading Standards Libraries

You can download and update standards libraries from the Designer [Home page](#).

How Do I Configure API Logging?

The ADS API uses the Apache log4j logging system. Please see the Log4J web site for:

- A more detailed description
- The Javadoc

All deployed ADS models write messages to an `org.apache.log4j.Logger` which is named after the root element of the object hierarchy. So if you knew this was "CSVFile" for example then you could get access to the logger without using ADS by calling:

```
org.apache.log4j.Logger.getLogger("CSVFile")
```

In the general case you can get access to the Logger object by calling

```
getLog() on a biz.c24.io.api.data.ComplexDataObject or on a biz.c24.io.api.data.Element
```

When you have a pointer to the Logger you will commonly want to add an appender to receive the log messages and set the level of the logger to filter out only the messages you are interested in. The code below adds an appender which writes information out to the console and sets the level of the logger to INFO.

```
org.apache.log4j.Logger l = element.getLog( );
l.addAppender(new org.apache.log4j.ConsoleAppender(new org.apache.log4j.SimpleLayout()));
l.setLevel(org.apache.log4j.Level.INFO);
```

As another example, the code snippet below adds an appender which will write ALL messages (in HTML format) to a file called `log.html`:

```
org.apache.log4j.Logger l = element.getLog();
l.addAppender(new org.apache.log4j.WriterAppender(new org.apache.log4j.HTMLLayout(), new
FileOutputStream("log.html")));
l.setLevel(org.apache.log4j.Level.ALL);
```

It is important to realise that repeatedly creating and adding new unreferenced instances of Appender to the Logger object will eventually cause a `java.lang.OutOfMemoryError` to occur as the Logger object can only hold a finite number of appenders in its list of associated appenders. There are two ways to prevent this from happening:

If you must add many unreferenced Appender objects to a Logger then its important to remove them when your code has finished processing by calling the `removeAllAppenders` method on the Logger object, as follows:

```
l.removeAllAppenders();
```

Alternatively you could keep a reference to an Appender and re-use it, as follows:

```
org.apache.log4j.Appender appender = new org.apache.log4j.ConsoleAppender(new
org.apache.log4j.SimpleLayout());
l.addAppender(appender);
org.apache.log4j.Logger l = element.getLog();
```

//processing and logging whilst doing so

l.removeAppender(appender);

How Do I Read XML Data in the API?

If your model is to be used purely to work with XML data or you want XML to be the default presentation mask then you should set the input and/or output masks on the model to XML. To do this, highlight the model (the root node) in the Explorer window and set the [Input Mask](#) and [Output Mask](#) properties to XML.

When you build code from the model with these settings, the following will return an object formatted in XML

- `toString()` calls on a `biz.c24.io.api.data.ComplexDataObject`
- `parseObject(String)` and `formatObject(Object)` calls on `biz.c24.io.api.data`

If your data model was built with anything other than XML set as its default mask you can instantiate a new `biz.c24.io.api.presentation.XMLSource` and use it instead of the default:

```
biz.c24.io.api.data.ComplexDataObject obj = new biz.c24.io.api.presentation.XMLSource(new
java.io.StringReader(str)).readObject(MyElement.getInstance());
```

The above case reads the data from a String called "str". If you were reading from a file, lets say "in.xml", you would do the following:

```
java.io.FileReader r = new java.io.FileReader("in.xml");
biz.c24.io.api.data.ComplexDataObject obj = new
biz.c24.io.api.presentation.XMLSource(in).readObject(MyElement.getInstance());
r.close();
```

Note: the above applies to all other types of input/output masks. You can create any sink/source and use it read/write any object, regardless of what format the message originated in / is destined for and what the default masks were at build time.

See also the FAQ on [writing XML data](#).

How Do I Validate a Complex Data Object?

Data objects can be validated at any level by calling the `validate()` method on `biz.c24.io.api.data.ComplexDataObject`.

The results of the validation will be reported in one of two ways, either via an exception or through a set of `biz.c24.io.api.data.ValidationEvent` instances. Which mechanism will be used can be configured by the `setValidationMechanism(biz.c24.io.api.data.ValidationMechanismEnum)` method. The four possible values to supply to this method are:

`biz.c24.io.api.data.ValidationMechanismEnum.VALIDATION_MECHANISM_PARENT` - inherit the behavior from the parent object

`biz.c24.io.api.data.ValidationMechanismEnum.VALIDATION_MECHANISM_SYNCHRONOUS_EVENT` - use a synchronous event / listener architecture

`biz.c24.io.api.data.ValidationMechanismEnum.VALIDATION_MECHANISM_ASYNCHRONOUS_EVENT` - use an asynchronous event / listener architecture

`biz.c24.io.api.data.ValidationMechanismEnum.VALIDATION_MECHANISM_EXCEPTION` - throw `biz.c24.io.api.data.ValidationExceptions`

If the event mechanism is selected then the `validate()` method will return `true` or `false` as to whether the object passed or failed validation. Events will be delivered to all registered validation listeners (either before or after the call returns) and these can be used to give suitable feedback to the client application.

Note: since events are delivered this mechanism allows for the possibility of more than one validation failure to be reported by one `validate` call.

If the exception mechanism is selected then the `validate()` method will either return `true` or result in an exception. It will never return `false`.

It is possible to perform more fine grained validation by calling `validate(biz.c24.io.api.data.ValidationConstraints)`. The `biz.c24.io.api.data.ValidationConstraints` argument allows you to constrain the parts of the model which are to be validated.

How Do I Use ADS and XPath?

XPath is a language for addressing parts of an XML document, designed to be used by both XSLT and XPointer.

XML documents can be complex and hold a lot of information, they're of no great value though unless we can extract information out of them. Without an standard mechanism for addressing files you would not be able to read this web page, its address is defined by the web server and subsequent path plus a file name. Similarly we need XPath to be able to identify a node or nodes within an XML document. The result of an XPath expression therefore usually a node or nodes from the original XML document.

As an example take some simple XML

```
<root>
  <description lang="en">A simple XPath demo</description>
  <description lang="fr">Un demo simple de XPath</description>
  <description lang="de">Ein einfaches demo XPath</description>
  <users>
    <user key="jdavies">
      <name>John Davies</name>
      <role>administrator</role>
      <role>manager</role>
    </user>
    <user key="smiller">
      <name>Steve Miller</name>
      <role>manager</role>
    </user>
    <user key="jtreacy">
      <name>James Treacy</name>
    </user>
  </users>
  <roles>
    <role>administrator</role>
    <role>manager</role>
  </roles>
</root>
```

To extract the description in French we could use `string(root/description[@lang='fr'])`.

To count the number of defined users we can use `count(root/users/user)` although the slightly less precise `count(//user)` would also return the same result in this case.

To count the number of users with the manager role we can use `count(root/users/user[role='manager'])`, or users with the first role: `count(root/users/user[role=/root/roles/role[1]])`.

XPath can replace simple queries on databases. We could even route XML based messages depending on an XPath value.

XPath and ADS

What would you do with a message like this:

Fear, Panic, Doubt
 James, Steve, Simon
 Wayne, Geoff, John
 Pat, Les, Lisa
 Paul, Kevin, Ed

Or worse still like this:

```
{1:F01ANZBGB20AXXX0548034693}{2:I100BKTRUS33XBRDN3}{3:{108:MT100 002 OF 049}}{4:
:20:00002
:32A:000103
USD112,
:50:FRANZ HOLZAPFEL GMBH
WIEN
:52A:BANKGB2LXXX
:53D:BANK OF HONDURAS
HONDURAS
:54B:BOMBAY
:56A:INTMUS33XXX
:57D:BANK OF NEW YORK
NEW YORK
:59:/722491-987
SAM COOKE
48TH ST, 52E, N.Y
:70:REMITTANCE INFO
:72:ADDITIONAL INFO
-}{5:{MAC:4A5BACED}{CHK:B37C6D1F426C}{TNG:}}
```

ADS lets you address any message format using XPath, once defined and imported the above messages can be treated almost as if they were XML documents without the expensive and risky transformation into XML.

For example, using ADS's SWIFT library you can read in the SWIFT MT100 above and extract the currency and amount from field 32A of block 4 (in bold) using the following expression:
 Block4/Field32aDatesAndAmounts/A/CurrencyAmount or just the currency using
 Block4/Field32aDatesAndAmounts/A/CurrencyAmount/Currency. To count the number of subfields in Block 4 we can use count(Block4/*).

This simple technique means that routing, modifying, sorting etc. of complex messages can all be achieved without unnecessarily complex, time consuming and risk transformation. Using ADS the message remains in its original format and is only transformed when (and if) absolutely necessary. Your message gets from A to B quicker, safer and cheaper.

ADS also makes extensive use of XPath to encapsulate semantic validation rules inside its own message standard libraries. For example, the SWIFT standard dictates that (among other things) in an MT100, if field 56 is present, then field 57 must always also be present.

This is easily manifested as an XPath based validation rule in ADS using the following condition:

```
/Block4[Field56aIntermediary and not(Field57aAccountWithInstitution)]
```

If the condition is found to be true when the object representing the message is asked to validate itself, the appropriate validation error is fired either in an event/listener model or as a ValidationException.

How Do I Write XML Data in the API?

Start by looking at the FAQ on [reading XML data](#).

In a similar way you can use Sinks instead of Sources and Writers instead of Readers. So you can create a new `biz.c24.io.api.presentation.XMLSink` to write the object to `System.out`:

```
new biz.c24.io.api.presentation.XMLSink(System.out).writeObject(obj);
```

Or to write it to a file called `out.xml`:

```
java.io.FileWriter w = new java.io.FileWriter("out.xml");  
new biz.c24.io.api.presentation.XMLSink(w).writeObject(obj);  
w.close();
```

See also the FAQ on [reading XML data](#).

How Does the SWIFT Pre-Parser Work?

When you know what SWIFT message types your data contains, and you therefore know what to expect, life is simple. You get an instance of the relevant element(s) and parse the data in using the `readObject(Element)` method of the appropriate Source.

Where you pick up random arbitrary message types from your data source, for example listening to a JMS Queue on which messages arrive, you can use a specialized subclass of Source that ships with the SWIFT library, `biz.c24.io.api.presentation.swift.SwiftPreParser`. As an extension of `biz.c24.io.api.presentation.Source`, it can be used in the same way as `DefaultSource`, `XMLSource`, and so on.

The element you pass into the `readObject()` method should have a complex type containing elements such as `MT502i`, `MT502o`, `MT509i`, `MT509o`, `MT515i`, `MT515o` etc. (substitute the list of message types you are concerned with). These elements should refer to the incoming and outgoing types in the SWIFT model.

Alternatively you can pass null into the `readObject()` method in which case `SwiftPreParser` tries to load the element of the appropriate message type using reflection. For this method to work the messages must be built with the default package names, i.e. `biz.c24.io.swift2005.mt1nn`, `biz.c24.io.swift2005.mt2nn` etc.

If the 2005 message type could not be found, `SwiftPreParser` automatically looks for 2003 and then 2000. If your messages are built in any other package structure, you should call `setBasePackageName(String)`.

Note: Your messages still need to be in `mt1nn`, `mt2nn` etc. subpackages.

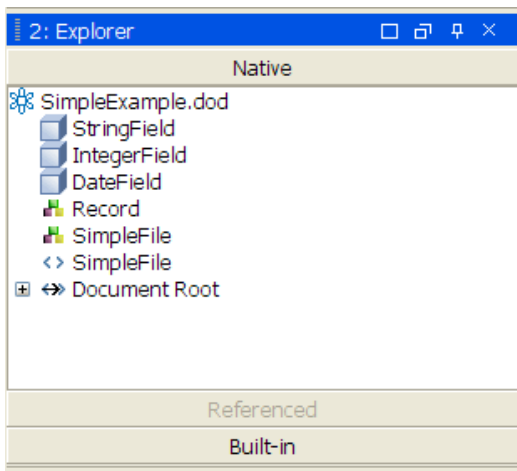
What Are Bean Classes and Interfaces?

Production of bean classes and interfaces in the generated code is a configuration option controlled by settings in Profiles.

There are two main settings to consider:

- Generate bean classes
- Generate bean interfaces

Consider the following data model, defined with a namespace of `biz.c24.io.deployed.examples.simple`:



With neither of the above settings enabled, and all other profile settings left to default, classes are generated as follows:

`biz.c24.io.deployed.examples.simple.DateFieldClass.java`

`biz.c24.io.deployed.examples.simple.IntegerFieldClass.java`

`biz.c24.io.deployed.examples.simple.RecordClass.java`

`biz.c24.io.deployed.examples.simple.SimpleExampleDataModel.java`

`biz.c24.io.deployed.examples.simple.SimpleFileClass.java`

`biz.c24.io.deployed.examples.simple.SimpleFileElement.java`

`biz.c24.io.deployed.examples.simple.StringFieldClass.java`

The code to read an instance of 'SimpleFile', and for each record print the value of 'StringField' to the console, is as follows:

```
ComplexDataObject file = new DefaultSource(new FileReader("SimpleFile.txt")).readObject(SimpleFileElement.getInstance());
for (int i=0;i<file.getElementCount("Record");i++)
{
    System.out.println(((ComplexDataObject)file.getElement("Record", i)).getElement("StringField"));
}
```

With the 'Generate bean class' option enabled, the additional classes highlighted are also generated:

biz.c24.io.deployed.examples.simple.DateFieldClass
 biz.c24.io.deployed.examples.simple.IntegerFieldClass
 biz.c24.io.deployed.examples.simple.Record
 biz.c24.io.deployed.examples.simple.RecordClass
 biz.c24.io.deployed.examples.simple.SimpleExampleDataModel
 biz.c24.io.deployed.examples.simple.SimpleFile
 biz.c24.io.deployed.examples.simple.SimpleFileClass
 biz.c24.io.deployed.examples.simple.SimpleFileElement
 biz.c24.io.deployed.examples.simple.StringFieldClass

These additional 'bean' classes, produced for the complex types in the model, contain 'get' / 'set' methods specific to their child element types. This changes the way we can use the deployed objects. The code above can be re-written as follows:

```
SimpleFile file = (SimpleFile)new DefaultSource(new FileReader("SimpleFile.txt")).readObject(SimpleFileElement.getInstance());
for (int i=0;i<file.getRecord().length;i++)
{
    System.out.println(file.getRecord()[i].getStringField());
}
```

The advantages of adopting this approach are greater type safety and ease of use.

With the 'Generate bean interface' option enabled, (note we must now specify a 'Bean Class Suffix' to prevent overwriting of bean classes by interfaces - in the example we have used 'Impl' so that the class 'Record' is now generated as 'RecordImpl') we get the following additional classes:

biz.c24.io.deployed.examples.simple.DateFieldClass.java
 biz.c24.io.deployed.examples.simple.IntegerFieldClass.java
 biz.c24.io.deployed.examples.simple.Record.java <<< This is now an interface
 biz.c24.io.deployed.examples.simple.RecordClass.java
 biz.c24.io.deployed.examples.simple.RecordImpl.java <<< This is what was generated as Record.java previously
 biz.c24.io.deployed.examples.simple.SimpleExampleDataModel.java
 biz.c24.io.deployed.examples.simple.SimpleFile.java <<< This is now an interface
 biz.c24.io.deployed.examples.simple.SimpleFileClass.java
 biz.c24.io.deployed.examples.simple.SimpleFileElement.java

biz.c24.io.deployed.examples.simple.SimpleFileImpl.java <<< This is what was generated as SimpleFile.java previously

biz.c24.io.deployed.examples.simple.StringFieldClass.java

So RecordImpl implements the Record interface, and SimpleFileImpl implements the SimpleFile interface.

You can specify a list of your own interfaces that the bean class / interface for a type should implement by editing the 'Bean Interface(s)' property of the complex type. You can also specify a class that the bean class for a type must extend by editing the property 'Bean Superclass'.

Why Do I Get An OutOfMemoryError in Log4J?

It is important to realize that repeatedly creating and adding new unreferenced instances of Appender to the Logger object will eventually cause an `java.lang.OutOfMemoryError` to occur as the Logger object can only hold a finite number of appenders in its list of associated appenders. There are two ways to prevent this from happening:

If you must add many unreferenced Appender objects to a Logger then its important to remove them when your code has finished processing by calling the `removeAllAppenders` method on the Logger object e.g.:

```
l.removeAllAppenders();
```

alternatively you could keep a reference to an Appender and re-use it e.g.:

```
org.apache.log4j.Appender appender = new org.apache.log4j.ConsoleAppender(new  
org.apache.log4j.SimpleLayout());
```

```
l.addAppender(appender);
```

```
org.apache.log4j.Logger l = element.getLog();
```

```
...
```

```
l.removeAppender(appender);
```

How Do I Edit a Model's Namespaces?

To edit a data model's namespaces select it in the Explorer window and edit the [Namespaces](#) property in the Properties window. A dialog appears containing the currently defined namespace mappings for the model.

The namespaces table always contains an entry for the target namespace of the model. When you import a model, the namespace of the imported model is added to the table, thereby allowing you to specify a prefix for it.

How Do I Set the Target Namespace of a Data Model?

Select the root data model node in the [Explorer window](#) and edit the [Target Namespace](#) property in the Properties window. The value must be a valid URI.

If the URI that does not already exist in the namespace list of the Profiles, it is added to the list.

Note: because the project can contain unloaded models which use the old value of the target namespace, that URI will NOT be removed from the lists of namespaces. It is thus quite easy to grow the list of namespaces inadvertently with regular changes to the target namespace property. In this case, you should remove old entries from all Profiles.

How Can I Build Via Apache Ant?

A number of Apache Ant tasks specific to ADS are packaged within the artix-ds-designerXXX.jar file.

These enable you to automat building and exporting data models using an Ant script.

This is useful where the build of ADS generated components are to be included within overall project builds without any requirement to manually build the components from within Designer.

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.

Why Does Windows Not Like the Length of My Generated Class Names?

Windows has a limit of 255 characters on the length of filenames. Classes generated with many packages can provoke this issue. We recommend building files with long fully-qualified path names in a directory off the root of a drive.

Where Are the Examples Located?

ADS ships a selection of example data models and transformations along with some Java classes that demonstrate the different ways deployed ADS code can be used in application development.

The examples are downloaded to your My ADS Projects folder by default.

How Do I Configure Java System Properties?

You can change Java system properties, such as heap size, in the `artix-ds.ja` file in the `ADS_HOME` directory.

Edit the file in a text editor and save your changes. These changes will take effect when you restart ADS Designer.

How Do I Configure the Designer Classpath?

Open the artix-ds.cp file in your My ADS Projects folder in a text editor.

To add your own entries to the classpath, follow the steps rules below:

- Start each entry on a new line
- Prefix each entry with a semi-colon (;)
- Do NOT add your entry to the ADS_HOME/lib/filename.jar block
- Do not include blank lines

Warning: Be sure to close the artix-ds.cp when installing or uninstalling the product.

How Do I Increase Designer's Memory?

[Configure Java system properties](#) and change the -Xmx000m property to set the maximum amount of memory you want the JVM to use.

How Is The ADS_HOME Environment Variable Set Up?

You can set the ADS_HOME environment variable in two different ways:

- Set the -DADS_HOME property in the artix-ds.ja file
- Set ADS_HOME as an operating system environment variables

Is There a Getting Started Guide for ADS?

Yes, there is a step-by-step tutorial for ADS.

Once you download the Getting Started plug-in from the [Home page](#), links to the HTML and PDF versions of the guide are stored in the My ADS projects/Getting Started/Guide folder on your machine.

This guide provides information on how to create data models and transformations and build source code from them, and how to parse valid and invalid data. It also contains example data models and transformations that you can view and extend.

How Do I Create a Data Model from an XML Schema?

See [Importing from an XML Schema](#).

How Do I Generate an XML Schema from a Data Model?

See [Exporting to XML Schema](#).

How Do I Add a New Path to a Project?

The initial set of directory location paths available to a project is defined in the project properties.

To add to or change the list of directory location paths, go to the Project properties either from the **File** menu or by right-clicking on the project root in the [Project window](#).

DataXtend SI Integration

ADS allows developers using Progress DataXtend Semantic Integrator (DataXtend SI) to make use of ADS data models—either standards libraries downloaded and installed into ADS Designer or customer-developed models.

As a DataXtend SI user, you can install an ADS DOD Importer plug-in into your DXSI workbench when you install ADS. See the *Artix Data Services Installation Guide* for details.

Once the DOD Importer plug-in is installed, proceed as follows:

In ADS Designer:

1. Download a [standards library](#) data model (.dod) file from the [Home](#) page, or [create](#) your own data model.
2. [Build](#) Java class instances of your data model.
3. If you plan to use standards library test data in DataXtend SI, first [decrypt the data](#) from the ADS Designer **Tools** menu.

In DataXtend SI:

1. Select **File > Import ... File System** to import the ADS data model, plus any required test data into your project.
2. Select **File > Import ... DataXtend SI Import > ADS DOD Model** to import the data model.
3. Add the JAR file(s) that you generated from your data model in ADS to your DataXtend SI project's Java build path.

For details of working with ADS data models in DataXtend SI, see the DataXtend SI Workbench online help.

Apache Camel Integration

You can integrate ADS with the Apache Camel routing engine. The [Run wizard](#) in ADS Designer allows you to select from a range of Camel-supported transport protocols when it comes to building code for the purposes of marshalling/unmarshalling and transforming your data.

- [Supported Transports](#)
- [Customizing Your Routing Requirements](#)
- [Examples](#)
- [Further Information](#)

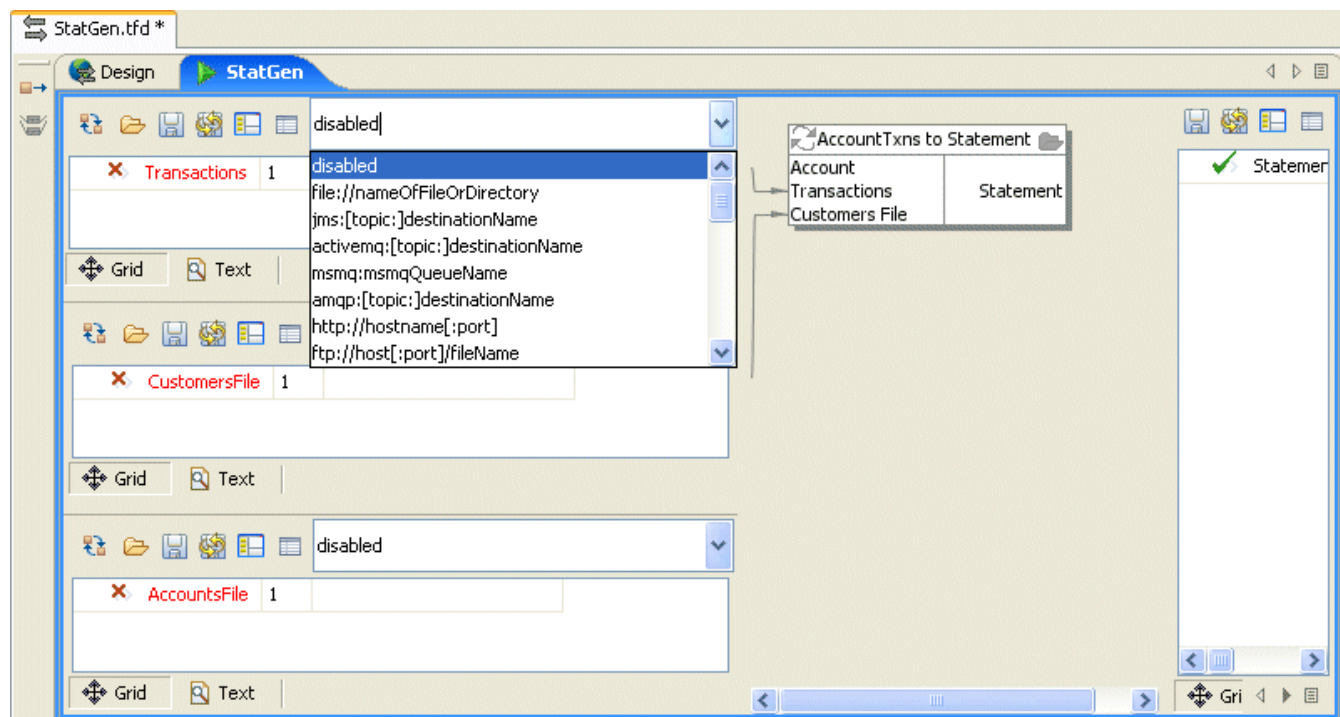
Note: Apache Camel integration is supported in ADS design-time only. If you want to use Camel with ADS in a production environment, you must use the Apache Active MQ or FUSE Message Broker.

Supported Transports

ADS supports any Camel component that is listed in <http://camel.apache.org/components.html>.

Customizing Your Routing Requirements

Customizing Camel routes for your ADS solutions in Designer is just a matter of selecting from a range of URIs that represent the various supported transports for your data inputs and outputs, and then building the required code. You can alternatively enter the URI in the relevant field. You can specify any component that is supported by Camel.



This enables you to quickly and easily implement the formatting and routing requirements for your data solutions.

Examples

Your ADS installation includes examples of how to use Camel with ADS for the purposes of marshalling, unmarshalling and transforming both textual data and FIX messages. You can download these demonstrations by clicking the **Examples** link in ADS Designer. They are downloaded by default to your My ADS Projects/Examples folder.

Further Information

Further information on how to use Camel with ADS can be found at <http://camel.apache.org/artix-data-services.html>.

Index

A

- aliases (108)
 - adding to data models (156)
- annotations (158)
- Ant Build window (47)
- any attributes (139)
- any elements (134)
- Apache Camel (357)
- atomic simple types (115)
- attribute references (242)
- attributes (135)
 - any attributes (139)
 - attribute groups (137)

B

- bean classes and interfaces (339)
 - properties (242)
- building (283)
- built-in data types (121)

C

- Camel (357)
- cardinality (153)
- Castor XML (93)
- classification groups (157)
- classpath (107)
- collate (258)
- compiling (327)
- complex data types (127)
- components (112)
 - data (129)
 - model (113)
 - validation (142)
- context menu (324)

D

- Data Model palette (55)
- data models (109)
 - adding components to (112)
 - building instances of (19)
 - creating (110)
 - exporting (190)
 - importing (162)
- data object definition (109)
- DataXtend SI (356)
- decrypting test data (74)
- Designer (11)
 - launching (10)
 - preferences (29)
- Diff Tool (315)
- document roots (204)
- DOD (109)

E

- element groups (132)
- elements (130)
 - creating from types (131)
- enumerations (143)
- examples (326)
- Excel (186)
- Explorer window (42)
- exporting (190)
 - to DTD (193)
 - to HTML (200)
 - to RelaxNG Compact (198)
 - to RelaxNG XML (196)
 - to XML Schema (191)

F

- factory lookups (145)
- File menu (57)
 - Diff Tool (317)
- finding components (26)
- Find menu (69)
- Find window (48)
- Find window (48)
- FIX (326)
- FpML (326)
- frequently asked questions (333)

G

- general/advanced properties (248)
- general properties (242)
- global elements (204)
- Go To menu (321)

H

- hashtable functions (224)
- Help menu (82)
 - Diff tool (323)
- Home page (36)

I

- Implemented/extended (242)
- importing (162)
 - from database (176)
 - from DTD (165)
 - from Excel spreadsheet (186)
 - from Java class (174)
 - from RelaxNG Compact (173)
 - from RelaxNG XML (172)
 - from text (168)
 - from WSDL (166)
 - from XML (167)
 - from XML Schema (163)
- imports (154)
- includes (154)
- Inlines (93)

instantiators (228)
introspectors (226)
ISO20022/Unifi (326)

J

Javadoc (24)
Java methods (221)
JVM (350)

L

Linkage Errors window (51)
Linux (327)
list simple types (118)
loading data (289)
loading model changes (296)
local transformations (217)
log4j (331)
 out of memory error (342)
logging (331)
logging levels (297)

M

mapping file (189)
Messages window (46)
Messages window (46)

N

namespaces (155)
 defining (105)
new features (7)

O

object instances (293)
Options menu (320)
Overview window (52)

P

preferences (29)
profile settings (92)
 classpath (107)
 namespaces (105)
 properties (93)
projects (85)
 aliases (108)
 paths (87)
 profiles (91)
 properties (88)
Project window (41)
properties (239)
 database (275)
 general (242)
 general/advanced (248)
 presentation (255)
 presentation/advanced (258)
 project (88)

SWIFT (277)
transform (279)
validation (263)
validation/advanced (268)
XML (270)

Properties window (44)

R

reference implementations (326)
router (357)
running (284)

S

SEPA (326)
simple data types (114)
spreadsheet (186)

T

text view (298)
transformations (206)
 adding filters to (218)
 adding functions to (214)
 adding hashtable functions to (224)
 adding instantiators to (228)
 adding introspectors to (226)
 adding Java methods to (221)
 adding local transforms to (217)
 adding transformation references to (216)
 automatically aligning components in (233)
 collaborating on (238)
 creating (207)
 exporting HTML from (237)
 finding components in (234)
 finding unmapped components in (231)
 globalizing local (235)
 moving mappings in (236)
 setting up translations in (229)
 using curved connectors in (230)
Transformations palette (56)

U

union simple types (120)

V

validation components (142)
validation rules (146)
verifying (282)

W

Window menu (81)
 Diff Tool (322)
WSDL (166)

X

XML (167)

XPath (335)
 setting properties (280)
 setting validation rules (146)
XQuery (147)