Oracle® Fusion Middleware

Desktop Integration Developer's Guide for Oracle Application Development Framework

11*g* Release 1 (11.1.1.5.0)

E10139-04

April 2011

Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework 11*g* Release 1 (11.1.1.5.0)

E10139-04

Primary Author: Himanshu Marathe

Contents

Preface

Welcome to the *Desktop Integration Developer's Guide for Oracle Application Development Framework*.

Audience

This manual is intended for enterprise developers who configure desktop applications to integrate with the Oracle Fusion Middleware Application Development Framework (Oracle ADF).

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/support/contact.html` or visit `http://www.oracle.com/accessibility/support.html` if you are hearing impaired.

Related Documents

For more information, see the following:

- *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*
- *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements (for example, menus and menu items, buttons, tabs, dialog controls), including options that you select.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates language and syntax elements, directory and file names, URLs, text that appears on the screen, or text that you enter.

1

Introduction to ADF Desktop Integration

This chapter provides an introduction to ADF Desktop Integration.

This chapter includes the following sections:

- Section 1.1, "Introduction to ADF Desktop Integration"
- Section 1.2, "About ADF Desktop Integration with Microsoft Excel"

1.1 Introduction to ADF Desktop Integration

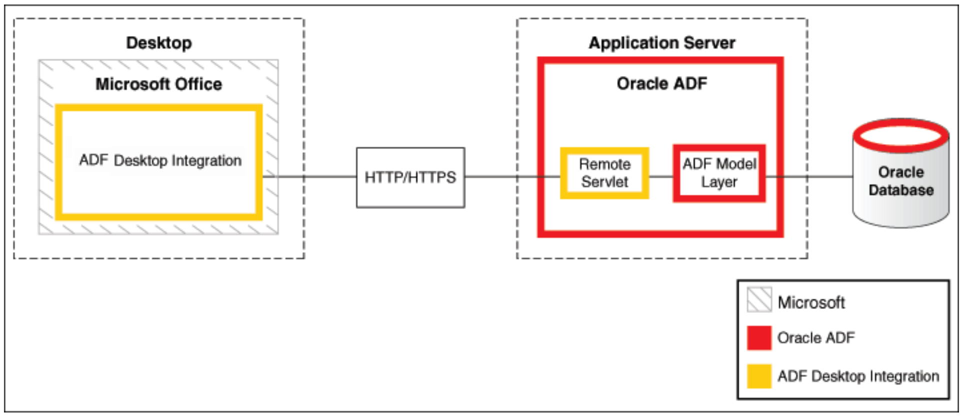
Many end users of Fusion web applications use desktop applications, such as Microsoft Excel, to manage information also used by their web application. ADF Desktop Integration provides a framework for Oracle Application Development Framework (Oracle ADF) developers to extend the functionality provided by a Fusion web application to desktop applications. It allows end users to avail themselves of Oracle ADF functionality when they are disconnected from their company network. End users may also prefer ADF Desktop Integration because it provides Excel's familiar user interface to undertake information management tasks, such as performing complex calculations or uploading a large amount of data, easily and seamlessly.

ADF Desktop Integration is a part of the Oracle ADF architecture. More information about the Oracle ADF architecture can be found in the "Oracle ADF Architecture" section of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Figure 1–1 illustrates the architecture of ADF Desktop Integration, which comprises of the following components:

- ADF Desktop Integration add-in
- ADF Desktop Integration remote servlet
- ADF Model layer

Figure 1–1 ADF Desktop Integration Architecture



For more information about ADF Desktop Integration, see the ADF Desktop Integration page on Oracle Technology Network (OTN) at:

`http://www.oracle.com/technetwork/developer-tools/adf/overview/index-085534.html`

1.2 About ADF Desktop Integration with Microsoft Excel

Currently, ADF Desktop Integration supports integration with Microsoft Excel 2007, and other higher versions of Microsoft Excel.

Note: This guide uses the term *integrated Excel workbook* to refer to Excel workbooks that you integrate with a Fusion web application and to distinguish these workbooks from workbooks that have not been integrated with a Fusion web application or configured with Oracle ADF functionality.

1.2.1 Overview of Creating an Integrated Excel Workbook

Creating an integrated Excel workbook involves the steps described in Table 1–1.

Table 1–1 Steps to Create an Integrated Excel Workbook

Use	To
JDeveloper	Create a Fusion web application. For information about creating a Fusion web application, see the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework.* Add data controls that expose the elements you require in Microsoft Excel. Create page definition files that expose the Oracle ADF bindings to use in Excel. For more information, see Section 4.3, "Working with Page Definition Files for an Integrated Excel Workbook."
Excel	Create the Excel workbooks that you intend to configure with Oracle ADF functionality. For more information, see Section 4.4, "Adding Integrated Excel Workbook to a Fusion Web Application."

Table 1–1 (Cont.) Steps to Create an Integrated Excel Workbook

Use	To
	Configure the Excel workbook using the Oracle ADF bindings that you exposed in the page definition files and the Oracle ADF components that ADF Desktop Integration provides. For more information, see the following sections and chapters: ■ Chapter 5, "Getting Started with the Development Tools" This chapter provides an overview of the tools that ADF Desktop Integration provides so that you can configure an Excel workbook with Oracle ADF functionality. ■ Chapter 6, "Working with ADF Desktop Integration Form-Type Components" This chapter describes how you insert ADF Desktop Integration form-type components into Excel worksheets and configure their properties to determine behavior at runtime. ■ Chapter 7, "Working with ADF Desktop Integration Table-Type Components" This chapter describes how you can use the ADF Table and Read-only Table components to provide end users with a means of displaying and editing data hosted by a Fusion web application. ■ Chapter 12, "Adding Validation to an Integrated Excel Workbook" This chapter describes how you provide validation for your integrated Excel workbook.
	Test your integrated Excel workbook. For more information, see Chapter 13, "Testing Your Integrated Excel Workbook."
	Once you complete the integration of your Excel workbook with your Fusion web application, you deploy it to make it available to your end users. For information about this task, see Chapter 14, "Deploying Your Integrated Excel Workbook."

1.2.2 The Advantages of Integrating Excel with a Fusion Web Application

Advantages that accrue from integrating Microsoft Excel workbooks with your Fusion web application include:

- Providing end users with access to data and functionality hosted by a Fusion web application through a desktop interface (Microsoft Excel) that may be more familiar to them.
- Users can access data hosted by a Fusion web application while not connected to the application. They must log on to the Fusion web application to download data. Once data is downloaded to an Excel workbook, they can modify it while disconnected from the Fusion web application.
- Bulk entry and update of data may be easier to accomplish through a spreadsheet-style interface.
- End users can use native Excel features such as macros and calculation.

2 Introduction to the ADF Desktop Integration Sample Application

This chapter provides an overview of the Master Price List module, which is the ADF Desktop Integration module's sample application. The Master Price List module is a module in the Fusion Order Demo application. It contains several Microsoft Excel workbooks that are integrated with a Fusion web application.

This chapter includes the following sections:

2.1 Introduction to the Master Price List Module

The Master Price List module allows end users to download information (product names, prices, and so on) about electronic devices that are sold through a storefront-type web application. End users can search the downloaded information, modify pricing information, and upload the modified information to the Fusion web application.

You must set up your development environment before you can set up and run the Master Price List module. After you set up your development environment, you can download the Fusion Order Demo application, which includes the Master Price List module.

2.2 Setting Up and Executing the Master Price List Module

Set up your development environment as described in Chapter 3, "Setting Up Your Development Environment", so that you can run the Master Price List module.

Once you have set up your development environment, download the Fusion Order Demo application, which includes the Master Price List module. For information about how to download the Fusion Order Demo application, see the "How to Download the Application Resources" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

The Fusion Order Demo application that you download includes a directory named `Infrastructure`. This directory includes scripts that create the users and data that the Fusion Order Demo application and Master Price List module require. For information about how to run these scripts, see the "How to Install the Fusion Order Demo Schema" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Note: If you have an old version of ADF Desktop Integration installed on your system, you must do the following:

1. Upgrade ADF Desktop Integration as described in Section 3.7, "Upgrading ADF Desktop Integration."
2. Refresh your Fusion Order Demo schema as described in "How to Install the Fusion Order Demo Schema" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

To run the Master Price List module:

1. Open the `MasterPriceList.jws` file in JDeveloper.

   This file is located in the `MasterPriceList` subdirectory of the directory into which you extracted the Fusion Order Demo application.

2. In the Application Navigator, click the Application Resources accordion title to expand the panel.
3. Right-click **FOD** connection and choose **Properties**.
4. In the Edit Database Connection dialog, modify the connection information shown in Table 2–1 for your environment.

Table 2–1 Database Connection Properties for the Master Price List Module

Property	Description
Host Name	The host name for your database. For example: `localhost`
JDBC Port	The port for your database. For example: `1521`
SID	The SID of your database. For example: `ORCL` or `XE`

   Do not modify the user name and password **fod/fusion**. These must remain unchanged. Click **OK**.

5. In the Application Navigator, right-click **Model** and choose **Rebuild Model.jpr**.
6. In the Application Navigator, right-click **ViewController** and choose **Rebuild ViewController.jpr**.
7. In the Application Navigator, expand the **ViewController** project, right-click **login.jspx** and choose **Run**.

   The `login.jspx` page runs and displays a login form.

8. To log on as an administrator, enter `sking` in the **User Name** field and `welcome1` in the **Password** field. To log on as a manager, enter `ahunold` in the **User Name** field and `welcome1` in the **Password** field. For more information about users, see Section 2.4.1, "Log on to the Fusion Web Application from an Integrated Excel Workbook."

You can now open and connect the integrated Excel workbooks described in Section 2.4, "Overview of the Integrated Excel Workbooks in the Master Price List Module" to the Fusion web application that the Master Price List module deploys.

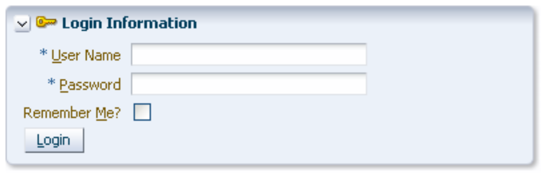
2.3 Overview of the Fusion Web Application in the Master Price List Module

The Fusion web application in the Master Price List module enables end users to edit and navigate through a list of products, search for a product, download integrated Excel workbooks, sort and reorder columns, and so on.

2.3.1 Log on to the Fusion Web Application in the Master Price List Module

When the end user runs the Master Price List Fusion web application in JDeveloper, the default browser opens the login page after the Master Price List module is deployed on Oracle WebLogic Server.

Figure 2–1 Login Dialog of Master Price List Fusion Web Application



The Master Price List module provides two user profiles to log on to the Fusion web application. Table 2–2 summarizes both user profiles.

Table 2–2 User Profiles for the Master Price List Module

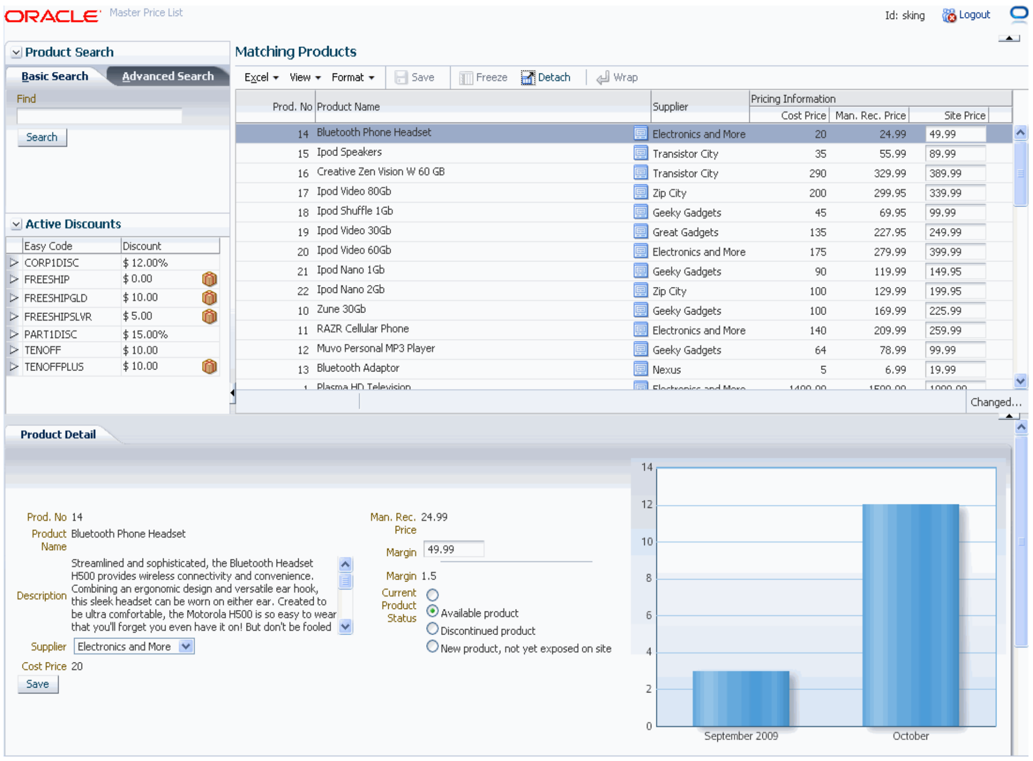
Login Name	Password	Role	Description
sking	welcome1	Administrator	Enables you to access and modify information.
ahunold	welcome1	Manager	Enables you to access information, but you cannot modify it.

After the login credentials are verified, the end user is redirected to the home page of the Master Price List Fusion web application.

2.3.2 Introduction to the Fusion Web Application in the Master Price List Module

The Master Price List Fusion web application is divided into four panes: Product Search, Matching Products, Product Detail, and Active Discounts.

Figure 2–2 Web Interface of Master Price List Fusion Web Application



Active Discounts

Easy Code	Discount
CORP1DISC	$ 12.00%
FREESHIP	$ 0.00
FREESHIPGLD	$ 10.00
FREESHIPSLVR	$ 5.00
PART1DISC	$ 15.00%
TENOFF	$ 10.00
TENOFFPLUS	$ 10.00

Prod. No	Product Name	Supplier	Pricing Information		
			Cost Price	Man. Rec. Price	Site Price
14	Bluetooth Phone Headset	Electronics and More	20	24.99	49.99
15	Ipod Speakers	Transistor City	35	55.99	89.99
16	Creative Zen Vision W 60 GB	Transistor City	290	329.99	389.99
17	Ipod Video 80Gb	Zip City	200	299.95	339.99
18	Ipod Shuffle 1Gb	Geeky Gadgets	45	69.95	99.99
19	Ipod Video 30Gb	Great Gadgets	135	227.95	249.99
20	Ipod Video 60Gb	Electronics and More	175	279.99	399.99
21	Ipod Nano 1Gb	Geeky Gadgets	90	119.99	149.95
22	Ipod Nano 2Gb	Zip City	100	129.99	199.95
10	Zune 30Gb	Geeky Gadgets	100	169.99	225.99
11	RAZR Cellular Phone	Electronics and More	140	209.99	259.99
12	Muvo Personal MP3 Player	Geeky Gadgets	64	78.99	99.99
13	Bluetooth Adaptor	Nexus	5	6.99	19.99
1	Plasma HD Television	Electronics and More	1400.00	1500.00	1900.00

Prod. No 14
Product Name Bluetooth Phone Headset
Description Streamlined and sophisticated, the Bluetooth Headset H500 provides wireless connectivity and convenience. Combining an ergonomic design and versatile ear hook, this sleek headset can be worn on either ear. Created to be ultra comfortable, the Motorola H500 is so easy to wear that you'll forget you even have it on! But don't be fooled
Supplier Electronics and More
Cost Price 20
Man. Rec. Price 24.99
Margin 49.99
Margin 1.5
Current Product Status ( ) [✓ Available product] ( ) Discontinued product ( ) New product, not yet exposed on site

The Product Search pane has tabs for searching a product in the application repository. The Product Search pane is an accordion pane and the end user can hide it, if desired. For more information about search functionality, see Section 2.3.4, "Searching a Product."

The Matching Products pane displays products in a tabular format, according to the search criteria set in the Product Search pane. By default, all products are displayed. The end user can use the Matching Products pane's toolbar to do various activities such as download integrated Excel workbooks, sort columns in ascending or descending order, hide a column, and so on. For more information about the toolbar, see Section 2.3.3, "Using the Matching Products Toolbar."

The Product Detail pane displays detailed information about the product selected in the Matching Products pane. The information includes the attributes available in the table, and additional information about the product's supplier, product's current availability status, a graphical representation of the product's sale in each month, and so on.

The Active Discounts pane is an independent pane. It displays the available discounts on products with their discount codes and discount percentages.

2.3.3 Using the Matching Products Toolbar

End users can use the toolbar in the Matching Products pane to do the following:

- Save changes
- Download integrated Excel workbooks
- Show or hide columns
- Sort columns

- Detach Products table from Matching Products pane
- Freeze or unfreeze a column
- Reorder columns
- Wrap or unwrap a column's values

2.3.3.1 How to Download Integrated Excel Workbooks

The Master Price List module provides various integrated Excel workbooks to meet different requirements. End users can download the integrated Excel workbooks from the Excel menu of the toolbar.

Figure 2–3 Excel Menu of Matching Products Toolbar

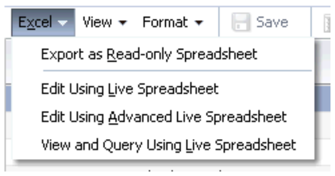

Excel
View
Format
Save
Export as Read-only Spreadsheet
Edit Using Live Spreadsheet
Edit Using Advanced Live Spreadsheet
View and Query Using Live Spreadsheet

Table 2–3 provides the description for each menu option.

Table 2–3 Download Excel Menu Options

Menu Option	Description
Export as Read-only Spreadsheet	Exports all Product table data into a spreadsheet. The spreadsheet is not an integrated Excel workbook. The exported file is saved as `read_only_pricelist.xls`.
Edit Using Live Spreadsheet	Downloads the integrated Excel workbook that enables you to edit and update common data. The downloaded file is saved as `EditPriceList.xlsx`.
Edit Using Advanced Live Spreadsheet	Downloads the integrated workbook that enables you to edit and update all data. The downloaded file is saved as `AdvEditPriceList.xlsx`.
View and Query Using Live Spreadsheet	Downloads the integrated workbook that enables you to view all products and, if desired, search for a product. This integrated Excel workbook does not allow you to edit and update data. The downloaded file is saved as `ReadOnlyPriceList.xlsx`.

2.3.3.2 How to Sort, Hide and Reorder Columns

The Master Price List module provides various operations on the Products table columns of the Matching Products pane. Using options available in the **View** menu, the end user can sort a column, show or hide a column, and reorder columns based on their requirements.

To show or hide columns:

1. In the Matching Products pane, from the toolbar, choose **View** > **Columns**.
2. In the **Columns** submenu, choose a column name to show or hide it in the Matching Products table. To show all columns, click **Show All**.

To sort a column:

1. In the Matching Products pane, select the table column to be sorted.
2. From the toolbar, choose **View** > **Sort**.
3. In the **Sort** submenu, choose **Ascending** to sort the values of the selected column in ascending order. Choose **Descending** to sort the values of the selected column in descending order.

   Tip: To sort a column in ascending or descending order, you can also click the Up or Down triangle in the column name header.

4. To sort multiple columsn by the same values at once, choose **Advanced**. In the Advanced Sort dialog, select the desired columns.

To reorder columns:

1. From the View menu, click **Reorder** to open the Reorder Columns dialog.
2. In the dialog, click the respective column name to select it, and then use the navigation buttons to reorder it.

   Tip: To order multiple columns, select their corresponding checkboxes, and then use navigation buttons. All selected columns move as a group.

2.3.3.3 Other Toolbar Operations

The Matching Products toolbar has buttons to save your changes, to detach the Matching Products table from the Matching Products pane, to freeze or unfreeze columns, and to wrap or unwrap a column's values.

- To save changes, click **Save**.
- To detach the Matching Products table from Matching Products pane, click **Detach**.
- To freeze a column, click the column header to select the column, and then click **Freeze**. Click **Freeze** again to unfreeze the column.
- To wrap a column's values, click the column header to select the column, and then click **Wrap**. Click **Wrap** again to unwrap the column values.

Note: The Master Price List Fusion web application allows edits in the following:

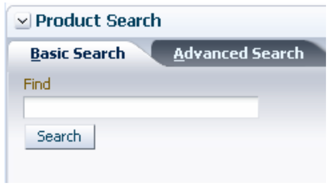
- Site Price column (Matching Products pane)
- Supplier dropdown list (Product Detail pane)
- Current Product Status option button (Product Detail pane)

2.3.4 Searching a Product

The Master Price List module provides a basic search and an advanced search facility to search for a product item. You can use the Product Search accordion panel to find an item.

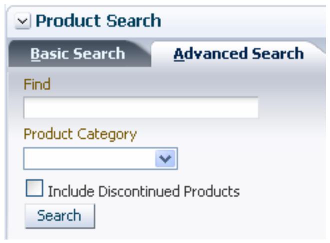
By default, the Product Search accordion panel shows the **Basic Search** tab where you can enter a search term to invoke a query on the Fusion web application and view the results.

Figure 2–4 Basic Search Tab in the Master Price List Fusion Web Application



The **Advanced Search** tab enables you to find products by category, and if required, search among discontinued products by selecting the **Include Discontinued Products** checkbox.

Figure 2–5 Advanced Search Tab in the Master Price List Fusion Web Application



2.4 Overview of the Integrated Excel Workbooks in the Master Price List Module

The Master Price List module provides the `EditPriceList.xlsx`, `AdvEditPriceList.xlsx`, and `ReadOnlyPriceList.xlsx` integrated Excel workbooks. All workbooks enable end users to:

- Log on to the Fusion web application from the workbook
- Download rows of data about product pricing
- Search the workbook for information about product pricing

In addition, the `EditPriceList.xlsx` and `AdvEditPriceList.xlsx` workbooks permit end users to:

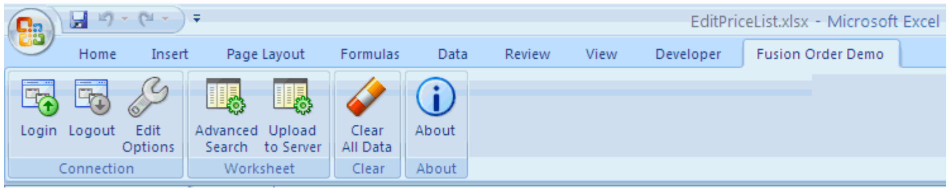
- Search the Master Price List module Fusion web application for information about products and product pricing
- Modify product pricing information in the workbook
- Use Excel formulas to perform calculations on values in an ADF Table component
- Upload modified product pricing information to the Master Price List module Fusion web application from the workbook

Subsequent sections in this chapter provide more information about the functionality in the workbooks along with cross-references to implementation details.

2.4.1 Log on to the Fusion Web Application from an Integrated Excel Workbook

At runtime, the integrated Excel workbooks in the Master Price List module render an Excel ribbon tab that allows end users to log on to the Fusion web application. Figure 2–6 shows the runtime Fusion Order Demo tab in the Ribbon of the `EditPriceList.xlsx` workbook.

Figure 2–6 Runtime Fusion Order Demo Tab



The `EditPriceList.xlsx` workbook prompts the end user to log on to the Fusion web application when the end user clicks **Login** or invokes an action that requires a connection with the Fusion web application. Because the worksheet `Startup` event in the `EditPriceList.xlsx` workbook invokes the ADF Table component `Download` action, end users are prompted to log on immediately after starting up the `EditPriceList.xlsx` workbook.

The **Login** button invokes the workbook `Login` action. For information about configuring the **Login** button (and other buttons in Figure 2–6), see Section 8.3, "Configuring the Runtime Ribbon Tab."

The workbook `Login` action invokes the Fusion web application's authentication process. For more information about implementing this functionality, see Chapter 11, "Securing Your Integrated Excel Workbook."

The Master Price List module provides two user profiles to log in to the application, as summarized in Table 2–2.

2.4.2 Download Rows of Data About Product Pricing

The `EditPriceList.xlsx` workbook uses an ADF Table component to host information downloaded from the Fusion web application about product pricing. This component allows end users to edit rows and upload modified rows to the Fusion web application.

The following sections provide information about how to implement the download functionality:

- Each worksheet that you integrate with a Fusion web application requires an associated page definition file. The Price List worksheet in the `EditPriceList.xlsx` workbook is associated with the `ExcelPriceListPageDef.xml` page definition file. In JDeveloper, expand the following nodes in the Application Navigator to view this file:

  ViewController > **Application Sources** > **oracle.foddemo.masterpricelist** > **view** > **pageDefs**

  For information about how to configure a page definition file, see Section 4.3, "Working with Page Definition Files for an Integrated Excel Workbook."
- The ADF Table component `Download` action downloads data from the Fusion web application to the worksheet. For information about how you invoke this action, see Section 7.6, "Configuring Oracle ADF Component to Download Data to an ADF Table Component."
- In the `EditPriceList.xlsx` workbook, the worksheet `Startup` event invokes an action set that includes the ADF Table component `Download` action. For information about configuring worksheet events, see Section 8.2.4, "How to Invoke an Action Set from a Worksheet Event."

The `ReadOnlyPriceList.xlsx` workbook uses an ADF Read-only Table component to download data from the Fusion web application about product pricing.

End users can view this data, but they cannot modify data or save changes to the Fusion web application.

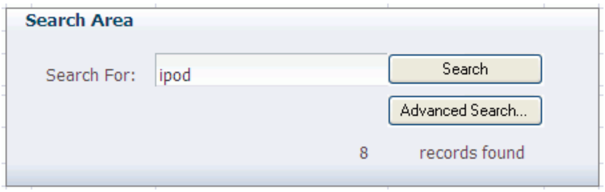
The following sections provide information about how to implement the download functionality of the `ReadOnlyPriceList.xlsx` workbook:

- For information about creating an ADF Read-only Table component, see Section 7.16, "Creating an ADF Read-Only Table Component."
- An ADF Button component is configured to invoke an action set that includes the ADF Read-only Table component `Download` action. For information about creating an ADF Button component, see Section 6.2, "Inserting an ADF Button Component."

2.4.3 Simple Search for Products in the Workbooks

The integrated Excel workbooks have ADF components configured to provide end users with a search form. End users can enter a search term in the form to invoke a query on the Fusion web application and download the results to the workbook. Figure 2–7 shows a runtime view of these components in the `EditPriceList.xlsx` workbook.

Figure 2–7 Runtime View of a Simple Search Form in the EditPriceList.xlsx Workbook



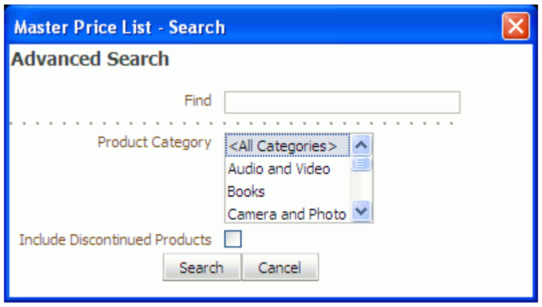
The following sections provide information about how to implement a simple search form that you can use in the `EditPriceList.xlsx` workbook:

- For information about creating a search form, see Section 8.6, "Creating ADF Databound Search Forms in an Integrated Excel Workbook."
- For information about creating a form, Section 8.7, "Adding a Form to an Integrated Excel Workbook."

2.4.4 Advanced Search for Products in the Edit Price List Workbook

The `EditPriceList.xlsx` and `AdvEditPriceList.xlsx` workbooks have search functionality configured that allow end users to invoke a page from the Fusion web application, specify search criteria, and download the results to the ADF Table component in the workbooks. Figure 2–8 shows the page from the Fusion web application that end users invoke by clicking the **Advanced Search** button.

Figure 2–8 Advanced Search Dialog in the EditPriceList.xlsx Workbook



For more information about how to implement the advanced search functionality in the `EditPriceList.xlsx` workbook, see Section 8.6, "Creating ADF Databound Search Forms in an Integrated Excel Workbook."

2.4.5 Modify Product Pricing Information in the Edit Price List Workbook

End users of the `EditPriceList.xlsx` and `AdvEditPriceList.xlsx` workbooks can edit product pricing information that the ADF Table component downloads from the Fusion web application. Columns in the runtime ADF Table component that have an `UpdateComponent` property configured permit end users to modify values and upload the changes to the Fusion web application. For example, end users can modify the values that appear in the **ProductId**, **ProductName**, and **CostPrice** columns.

End users can enter or modify the values that appear in the cells of other columns. However, the ADF Table component does not upload these changes to the Fusion web application, because some of these columns display the results of evaluating Excel formulas using values downloaded from the Fusion web application. Such columns should use a read-only style to distinguish themselves from other columns. For example, the **Difference** column displays the result of an Excel formula that subtracts the cost price from the list price and uses a read-only style, which makes it easily distinguishable from other input columns.

Other columns, such as **Status** and **Changed**, appear in the ADF Table component to provide status information about upload operations and changed columns.

The following sections provide information about how to implement this functionality:

- For information about inserting an ADF Table component, see Section 7.3, "Inserting an ADF Table Component into an Excel Worksheet."
- For information about using Excel formulas, see Section 8.10, "Using Calculated Cells in an Integrated Excel Workbook."
- For information about special columns, such as **Status** and **Changed**, see Section 7.11, "Special Columns in the ADF Table Component."

2.4.6 Upload Modified Product Information to the Fusion Web Application

The `EditPriceList.xlsx` and `AdvEditPriceList.xlsx` workbooks allow end users to upload modified data in the ADF Table component to the Fusion web application. An action set is configured for the runtime **Save Changes** button that invokes the ADF Table component's `Upload` action. For information about implementing this functionality, see Section 7.8, "Configuring an Oracle ADF Component to Upload Changes from an ADF Table Component."

Tip: You can also use the **Upload to Server** button in the Fusion Order Demo tab to upload modified data.

3

Setting Up Your Development Environment

This chapter describes how to set up your development environment to integrate an Excel workbook with a Fusion web application. The chapter concludes by describing how to upgrade and remove ADF Desktop Integration. The chapter also describes how to use ADF Desktop Integration on a system where you have multiple instances of JDeveloper. This chapter includes the following sections:

3.1 Introduction to Setting Up Your Development Environment

Setting up your development environment involves making sure that you have the correct versions of JDeveloper, Microsoft Office, and Microsoft Internet Explorer installed. You must also enable support for Microsoft .NET programmability, if it is not enabled. After you verify that you have the required software and enabled Microsoft .NET programmability, complete the setup of your development environment by:

- Allowing ADF Desktop Integration to access Microsoft Excel
- Installing ADF Desktop Integration

3.2 Required Oracle ADF Modules and Third-Party Software

Before you begin to integrate your Excel workbook with a Fusion web application, ensure that you have the required Oracle ADF modules and third-party software installed and configured:

- JDeveloper

Install the current release of JDeveloper. ADF Desktop Integration is available as a JDeveloper add-in.

- Microsoft Windows

  Microsoft Windows operating systems support the development and deployment of Excel workbooks that integrate with Fusion web applications. For more information about supported versions of Windows, see the "Oracle JDeveloper and Application Development Framework Certification Information" page on OTN at:

  `http://www.oracle.com/technetwork/developer-tools/jdev/index-091111.html`

- Microsoft Excel

  ADF Desktop Integration supports the integration of Fusion web applications with the following types of Excel workbook:

  – Excel Workbook

    The default file format for Excel workbooks is the Excel XML-based file format (`.xlsx`).

  – Excel Macro-Enabled Workbook

    Workbooks in this format (`.xlsm`) use the Excel XML-based file format and can store VBA macro code.

  ADF Desktop Integration does not support the use of other Excel file formats. For more information about supported versions of Excel, see the "Oracle JDeveloper and Application Development Framework Certification Information" page on OTN at:

  `http://www.oracle.com/technetwork/developer-tools/jdev/index-091111.html`

  Note: Microsoft Excel 2003 or older versions of Microsoft Excel are not supported.

- Internet Explorer

  Some features in ADF Desktop Integration use a web browser control from the Microsoft .NET Framework. This browser control relies on the local Internet Explorer installation to function properly.

  Note that Internet Explorer is the only web browser that supports this feature. For more information about supported versions of Internet Explorer, see the "Oracle JDeveloper and Application Development Framework Certification Information" page on OTN at:

  `http://www.oracle.com/technetwork/developer-tools/jdev/index-091111.html`

- Application server

  For information about the application servers that you can use to deploy an application developed using ADF Desktop Integration, see the "Oracle JDeveloper and Application Development Framework Certification Information" page on OTN at:

`http://www.oracle.com/technetwork/developer-tools/jdev/index-091111.html`

3.3 Enabling Microsoft .NET Programmability Support

Microsoft Excel must have Microsoft .NET programmability support enabled before you can set up ADF Desktop Integration and start development of an Excel workbook that integrates with a Fusion web application. If you enabled Microsoft .NET programmability support during installation of Microsoft Excel, no further action is required.

To enable Microsoft .NET programmability support:

1. Click the Windows **Start** button, and choose **Settings** > **Control Panel**.
2. In the Control Panel, select and open **Add or Remove Programs**.
3. Select the entry in the Add or Remove Programs dialog for Microsoft Office and click **Change**.
4. Follow the instructions in the wizard that appears to enable Microsoft .NET programmability support for Microsoft Excel.

3.4 Allowing ADF Desktop Integration to Access Microsoft Excel

You must configure Microsoft Excel settings to make it accessible from ADF Desktop Integration. You only need to perform this procedure once.

To allow Excel to run an integrated Excel workbook:

1. Open Excel.
2. Click the **Microsoft Office** button, and choose **Excel Options**.
3. In the Excel Options dialog, choose the **Trust Center** tab, and then click **Trust Center Settings**.
4. In the Trust Center dialog, choose the **Macro Settings** tab, and then click the **Trust access to the VBA project object model** checkbox.
5. Click **OK**.

For more information about securing an Excel workbook that is integrated with a Fusion web application, see Chapter 11, "Securing Your Integrated Excel Workbook."

3.5 Installing ADF Desktop Integration

When you run the ADF Desktop Integration setup tool, it verifies whether software in the following list is installed on the system where you want to install the framework. If one or more of these pieces of software is not installed, the setup tool installs it in the order specified.

1. Windows Installer 3.1
2. Microsoft .NET Framework

   The Microsoft .NET Framework 3.5 Service Pack 1 provides the runtime and associated files required to run applications developed to target the Microsoft .NET Framework.

Note: Installation of Microsoft .NET Framework may require you to restart the system where you install it. After you restart, the setup tool automatically recommences to finalize installation.

---

3. Microsoft Visual Studio Tools for Microsoft Office

   The Microsoft Visual Studio Tools for the Microsoft Office system (version 3.0 Runtime) Service Pack 1 (x86) is required to run VSTO solutions for the Microsoft Office system.

4. ADF Desktop Integration add-in

   You can install the ADF Desktop Integration add-in from JDeveloper, or from the setup tool provided in `MW_HOME\jdeveloper\adfdi`. For more information about how to set up ADF Desktop Integration, see Section 3.5.1, "How to Set Up ADF Desktop Integration."

   Note that the ADF Desktop Integration installation is specific to the current Windows user profile. If you have multiple Windows user profiles on your system, and you want to use ADF Desktop Integration integrated Excel workbooks from some specific user profiles, you must log in to each user profile and install the ADF Desktop Integration add-in. For more information, see Section 3.5.1, "How to Set Up ADF Desktop Integration."

3.5.1 How to Set Up ADF Desktop Integration

The ADF Desktop Integration add-in is available in two editions, the Designer edition and the Runtime edition. You must use the Designer edition to create and test integrated Excel workbooks, and the Runtime edition to enable end users to use ADF Desktop Integration and integrated Excel workbooks.

Note: Do not install both editions of ADF Desktop Integration on the same system.

Although you do not require administrator privileges to install the ADF Desktop Integration add-in, administrator privileges may be required to run the installers for additional software that the installer attempts to download and install. You should also ensure that the proxy settings for Internet Explorer are configured to allow access to `*.microsoft.com` because the installer attempts to automatically download missing prerequisite software from Microsoft's web site.

By default, the installer runs in English. You can change the language that appears by following the instructions in Section 3.9, "Localizing the Setup of Visual Studio Tools for Office."

To install the Designer edition of ADF Desktop Integration:

1. Open JDeveloper.
2. From the **Tools** menu, choose **Install ADF Desktop Integration**.
3. Follow the instructions that appear in the dialog boxes to successfully install the required components.

   If you encounter an error during the installation process, ensure that you have removed the previous version of ADF Desktop Integration. For more information, see Section 3.6, "Removing ADF Desktop Integration."

4. If prompted, click **Yes** to restart the system and complete the setup of ADF Desktop Integration.

   Tip: You can also install the Designer edition of ADF Desktop Integration by running the `setup.exe` tool available in the *`MW_HOME`*`\jdeveloper\adfdi\bin\excel\addin\designer` directory.

   Note: The **Install ADF Desktop Integration** menu option is available only on the Windows installation of JDeveloper.

If you use multiple instances of JDeveloper or if you have an existing instance of the ADF Desktop Integration add-in on the system where you plan to invoke the `setup.exe` tool, review the information in Section 3.8, "Using ADF Desktop Integration on a System with Multiple Instances of JDeveloper" before you perform the installation procedure.

If you want to install the Runtime edition of ADF Desktop Integration, see Section I.1, "Installing the Runtime Edition of ADF Desktop Integration."

3.6 Removing ADF Desktop Integration

You use the Microsoft Windows Control Panel to remove the ADF Desktop Integration add-in from the system where you set it up. After you remove the ADF Desktop Integration add-in, you can no longer use integrated Excel workbooks on this system unless you reinstall the add-in.

To remove the ADF Desktop Integration add-in:

1. Click the Windows **Start** button and then choose **Settings** > **Control Panel**.
2. In the Control Panel, select and open **Add or Remove Programs**.
3. In the Add or Remove Programs dialog, select the installed ADF Desktop Integration add-in edition, and then click **Remove**.

   Note: If you have installed ADF Desktop Integration on multiple user profiles, you must remove it from each user profile.

3.7 Upgrading ADF Desktop Integration

If you are using an old version of the ADF Desktop Integration add-in, you must upgrade to the current version.

To upgrade the ADF Desktop Integration add-in:

1. Uninstall the old version of the ADF Desktop Integration add-in. For more information, see Section 3.6, "Removing ADF Desktop Integration."
2. Download and install the latest version of Oracle JDeveloper.
3. Install the new version of the ADF Desktop Integration add-in. For more information, see Section 3.5, "Installing ADF Desktop Integration."

Note: If you do not uninstall the old version of ADF Desktop Integration add-in, an error occurs unless the new installer is in the exact same location as the old installer.

3.7.1 How to Migrate an Integrated Excel Workbook to the Current Version of ADF Desktop Integration

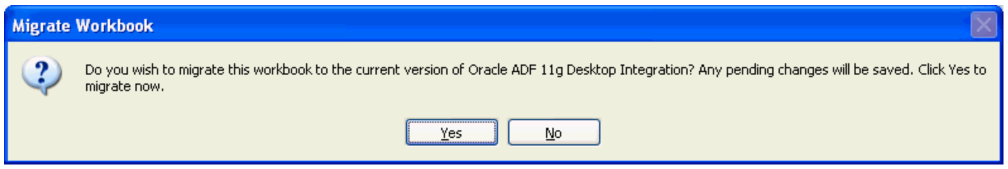
When you open the integrated Excel workbook after upgrading the ADF Desktop Integration add-in, the add-in detects and compares the ADF Desktop Integration version information of the workbook with the version installed on the client system. If required, you are asked to upgrade the metadata of the integrated workbook to the version installed on the client.

To migrate an integrated Excel workbook after upgrading:

1. Open the integrated Excel workbook.

   The Migrate Workbook dialog prompts you to migrate the workbook to the current version of ADF Desktop Integration, as shown in Figure 3–1.

   Figure 3–1 Migrate Workbook Dialog

   Migrate Workbook

   Do you wish to migrate this workbook to the current version of Oracle ADF 11g Desktop Integration? Any pending changes will be saved. Click Yes to migrate now.

   If you get one or more Microsoft Office Customization Installer error messages when you open the integrated Excel workbook, ignore the messages and continue. The error messages appear because ADF Desktop Integration cannot remove the old version information from the workbook before Excel detects it and reports the error.

2. Click **Yes** to migrate the workbook. The ADF Desktop Integration migration process closes the workbook and then reopens it, ready to be used with the current version of ADF Desktop Integration.

3.8 Using ADF Desktop Integration on a System with Multiple Instances of JDeveloper

You can have only one active installation of ADF Desktop Integration on a given system. By default, when you install JDeveloper, ADF Desktop Integration is extracted to `MW_HOME\jdeveloper\adfdi` . If you decide to move to another version of JDeveloper in a different directory, you must remove the old version of ADF Desktop Integration, as described in Section 3.6, "Removing ADF Desktop Integration." You must then set up ADF Desktop Integration with the new version of JDeveloper that you are moving to, as described in Section 3.5, "Installing ADF Desktop Integration."

Alternatively, you can set up ADF Desktop Integration in a directory that is independent of your JDeveloper installation. This approach means that you do not have to remove ADF Desktop Integration before moving to a newer version.

To set up ADF Desktop Integration in an independent directory:

1. Create a directory independent of the JDeveloper installation directory. For example, you may create the following directory:

   `D:\adfdi-excel-setup`

2. When you move to a newer version of JDeveloper, copy the contents of the following directory:

   `MW_HOME``\jdeveloper\adfdi\bin\excel\addin\designer`

   to:

   `D:\adfdi-excel-setup`

   where *`MW_HOME`* is the the Middleware Home directory.

3. Run the `setup.exe` tool located in `D:\adfdi-excel-setup`.

4. Follow the instructions that appear in the dialog boxes launched by `setup.exe` to set up the new version of ADF Desktop Integration.

5. If prompted, click **Yes** to restart the system and complete the setup of ADF Desktop Integration.

WARNING: After you install ADF Desktop Integration, do not delete the directory where you copied the setup files. You can delete the files after removing ADF Desktop Integration from the system.

3.9 Localizing the Setup of Visual Studio Tools for Office

Follow the instructions in this section to localize the setup of Visual Studio Tools for Office. By default, the installer described in Section 3.5, "Installing ADF Desktop Integration", runs in English. You can download and install a different language pack from the Microsoft Download Center for the language that you want to appear when the installer runs.

This section assumes that no instance of ADF Desktop Integration is present on your system and that your system uses a non-English version of the operating system. If ADF Desktop Integration is present, remove it as described in Section 3.6, "Removing ADF Desktop Integration."

For information about supported operating systems, see Section 3.2, "Required Oracle ADF Modules and Third-Party Software."

To localize the setup of Visual Studio Tools for Office:

1. Download the appropriate language pack (for example, French) for Microsoft Visual Studio Tools for Microsoft Office from the Microsoft Download Center at:

   `http://www.microsoft.com/downloads/`

2. Install the language pack that you downloaded in Step 1.

3. Set up ADF Desktop Integration, as described in Section 3.5, "Installing ADF Desktop Integration."

4 Preparing Your Integrated Excel Workbook

This chapter describes preparatory tasks that you must perform in developing your Fusion web application so that you can integrate an Excel workbook with the finalized application. This chapter also describes how you configure an Excel workbook before you add Oracle ADF functionality, using the tools provided in the ADF Desktop Integration module.

This chapter includes the following sections:

4.1 Introduction to Preparing Your Integrated Excel Workbooks

This chapter, and this guide as a whole, assumes that you have developed a functioning Fusion web application, as described in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Having developed your Fusion web application, you perform the tasks described in this chapter and elsewhere in this guide to configure an integrated Excel workbook with your Fusion web application. These tasks include adding the bindings that you require at runtime in the Excel workbook, preparing the Excel workbook for configuration with Oracle ADF functionality, and configuring the workbook by adding the Oracle ADF components that provide the functionality you require at runtime.

Note: Before you start, ensure that you have installed the Designer edition of ADF Desktop Integration. For more information about the ADF Desktop Integration editions, see Section 3.5, "Installing ADF Desktop Integration."

4.2 Adding ADF Desktop Integration to a Fusion Web Application

You enable Excel desktop integration for your Fusion web application by adding ADF Desktop Integration to the technology scope of the JDeveloper project where you develop the Fusion web application.

4.2.1 How to Add ADF Desktop Integration to Your JDeveloper Project

Use the Project Properties dialog in JDeveloper to add ADF Desktop Integration to the technology scope of your project.

To add ADF Desktop Integration to your project:

1. Open your project in JDeveloper.
2. In the Application Navigator, right-click the project to which you want to add ADF Desktop Integration and choose **Project Properties**.

   If your application uses the Fusion Web Application (ADF) application template, you select the **ViewController** project. If your application uses another application template, select the project that corresponds to the web application.
3. In the Project Properties dialog, select **Technology Scope** to view the list of available technologies.
4. Select the **ADF Desktop Integration** and **ADF Library Web Application Support** (optional) project technologies and add them to the **Selected Technologies** list.
5. Click **OK**.

> **Note:** You must add **ADF Library Web Application Support** to the technology scope if you plan to distribute integrated Excel workbooks by adding them to ADF library files through EAR and JAR files.

4.2.2 What Happens When You Add ADF Desktop Integration to Your JDeveloper Project

When you add the ADF Desktop Integration module to the technology scope of your project, the following events occur:

- The project adds the ADF Desktop Integration runtime library. This library references the following `.jar` files in its class path:
  - `adf-desktop-integration.jar`
  - `adf-desktop-integration-model-api.jar`
  - `resourcebundle.jar`
- The project's deployment descriptor (`web.xml`) is modified to include the following entries:
  - An ADF bindings filter (`adfBindings`)
  - A servlet named `adfdiRemote`

    > **Note:** The value for the `url-pattern` attribute of the `servlet-mapping` element for `adfdiRemote` must match the value of the `RemoteServletPath` workbook property described in Table A–18.

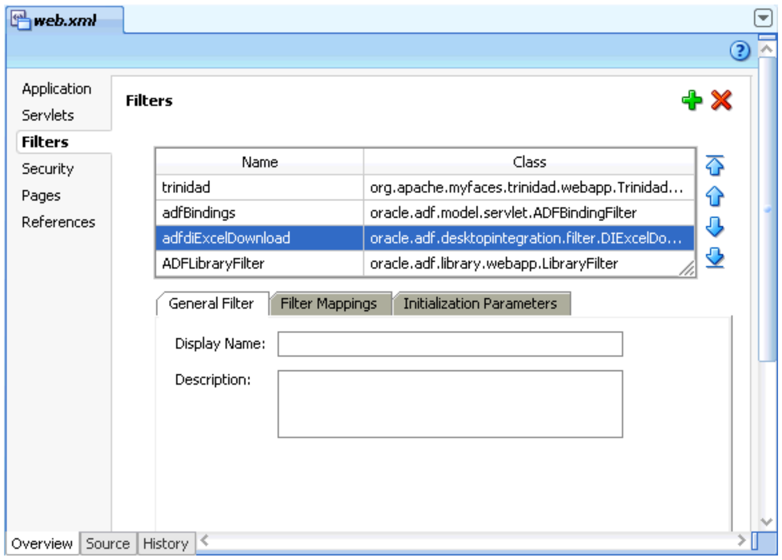
  - A filter named `adfdiExcelDownload`
  - A MIME mapping for Excel files (`.xlsx` and `.xlsm`)

The previous list is not exhaustive. Adding ADF Desktop Integration to a project makes other changes to `web.xml`. Note that some entries in `web.xml` are added only if they do not appear in the file.

While updating filter and filter mapping information in the `web.xml` file, ensure that the filter for ADF Library Web Application Support (`<filter-name>ADFLibraryFilter</filter-name>`) appears below the `adfdiExcelDownload` filter entries, so that integrated Excel workbooks can be downloaded from the Fusion web application.

Figure 4–1 shows the **Filters** tab of the `web.xml` editor in JDeveloper.

Figure 4–1 Filters tab of web.xml



Name	Class
trinidad	org.apache.myfaces.trinidad.webapp.Trinidad...
adfBindings	oracle.adf.model.servlet.ADFBindingFilter
adfdiExcelDownload	oracle.adf.desktopintegration.filter.DIExcelDo...
ADFLibraryFilter	oracle.adf.library.webapp.LibraryFilter

For more information about `web.xml`, see Appendix E, "ADF Desktop Integration Settings in the Web Application Deployment Descriptor."

4.2.3 What Happens When You Deploy ADF Desktop Integration Enabled Fusion Web Application from JDeveloper

When you deploy your ADF Desktop Integration enabled Fusion web application from JDeveloper, references to the ADF Desktop Integration shared libraries are added to the appropriate descriptor files. For any Fusion web application that contains one or more projects referencing the ADF Desktop Integration Model API library, or the ADF Desktop Integration Runtime library, a platform-dependent reference to the ADF Desktop Integration Model API shared library is added during deployment.

For any web application module (WAR) project that contains a reference to the ADF Desktop Integration Runtime library, a platform-dependent reference to the ADF Desktop Integration Runtime shared library is added during deployment.

4.2.3.1 Deploying your Fusion Web Application on Oracle WebLogic Server

When you deploy your Fusion Web application on Oracle WebLogic Server, the following happens:

- The `META-INF/weblogic-application.xml` file of the deployed application EAR contains a library reference to `oracle.adf.desktopintegration.model`.

For example:

```
<library-ref>
  <library-name>oracle.adf.desktopintegration.model</library-name>
</library-ref>
```

The shared library is delivered in `MW_HOME/oracle_common/modules/oracle.adf.desktopintegration.model_11.1.1`, in the `oracle.adf.desktopintegration.model.ear`.

- The `WEB-INF/weblogic.xml` file of the deployed web application WAR contains a library reference to `oracle.adf.desktopintegration`.

  For example:

  ```
  <library-ref>
    <library-name>oracle.adf.desktopintegration</library-name>
  </library-ref>
  ```

  The shared library is delivered in `MW_HOME/oracle_common/modules/oracle.adf.desktopintegration_11.1.1`, in the `oracle.adf.desktopintegration.war`.

4.2.3.2 Deploying your Web Application on IBM WebSphere Application Server

When you deploy your web application on IBM WebSphere Application Server, the following happens:

- For applications requiring the ADF Desktop Integration Model API library, or the ADF Desktop Integration Runtime library, the deployment procedure inserts a reference to the `com/oracle/adfdimodel` extension into the `META-INF/MANIFEST.MF` file of the application EAR file.

  For example:

  ```
  Manifest-Version: 1.0
  Extension-List: adfm adfdimodel
  adfm-Extension-Name: com/oracle/adfm
  adfm-Specification-Version: 1.0
  adfdimodel-Extension-Name: com/oracle/adfdimodel
  adfdimodel-Specification-Version: 1.0
  UseWSFEP61ScanPolicy: false
  ```

- The `deployment.xml` file for web applications with projects that refer to the ADF Desktop Integration Runtime library contains a library reference inserted during deployment.

  For example:

  ```
  <libraries xmi:id="LibraryRef_1274886542330_oracle.adf.desktopintegration_1.0_
  11.1.1.2.0" libraryName="oracle.adf.desktopintegration_1.0_11.1.1.2.0"
  sharedClassloader="true"/>
  ```

Note: For more information about system administration tasks and the specifics about shared libraries for these platforms, refer to the Oracle WebLogic Server and IBM WebSphere Application Server documentation.

4.3 Working with Page Definition Files for an Integrated Excel Workbook

Page definition files define the bindings that populate the data in the Oracle ADF components at runtime. Page definition files also reference the action bindings and method action bindings that define the operations or actions to use on this data. You must define a separate page definition file for each Excel worksheet that you are going to integrate with a Fusion web application. The integrated Excel workbook can include worksheets that do not reference page definition files.

The ADF Desktop Integration task pane displays only those bindings that ADF Desktop Integration supports in the bindings palette. If a page definition file references a binding that ADF Desktop Integration does not support (for example, a graph binding), it is not displayed.

Table 4–1 lists and describes the binding types that the ADF Desktop Integration module supports.

Table 4–1 Binding Requirements for ADF Desktop Integration Components

ADF Desktop Integration component	Supported Binding	Additional comments
ADF Input Text	Attribute binding	
ADF Output Text	Attribute binding	
ADF Label	Attribute and list bindings	This ADF Desktop Integration component uses the label property of a control binding.
ADF List of Values	List binding	
Tree Node List	Tree binding attributes and list binding	Tree binding attributes must be associated with a model-driven list.
ADF Button	Various	The ADF Button component in ADF Desktop Integration can invoke action sets. Action sets can reference action bindings, method action bindings, or actions exposed by components in ADF Desktop Integration. For more information about action sets, see Section 8.2, "Using Action Sets."
ADF Read-only Table	Tree binding	
ADF Table	Tree binding	

For information about the bindings that components in ADF Desktop Integration use, see Appendix A, "ADF Desktop Integration Component Properties and Actions."

For information about the elements and attributes in page definition files, see "*pageName*PageDef.xml" section of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

For information about ADF data binding and page definition files in a Fusion web application, see the "Using Oracle ADF Model in a Fusion Web Application" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

4.3.1 How to Create a Page Definition File for an Integrated Excel Workbook

You create and configure a page definition file that determines the Oracle ADF bindings to expose in your JDeveloper project.

To create a page definition file for an integrated Excel workbook:

1. In JDeveloper, add a new JSF page in your ADF Desktop Integration application's project.

   Tip: Add an ADF Faces `Table` component to the JSF page. JDeveloper generates the tree bindings in the JSF page that the ADF Table-type components use in the page definition file.

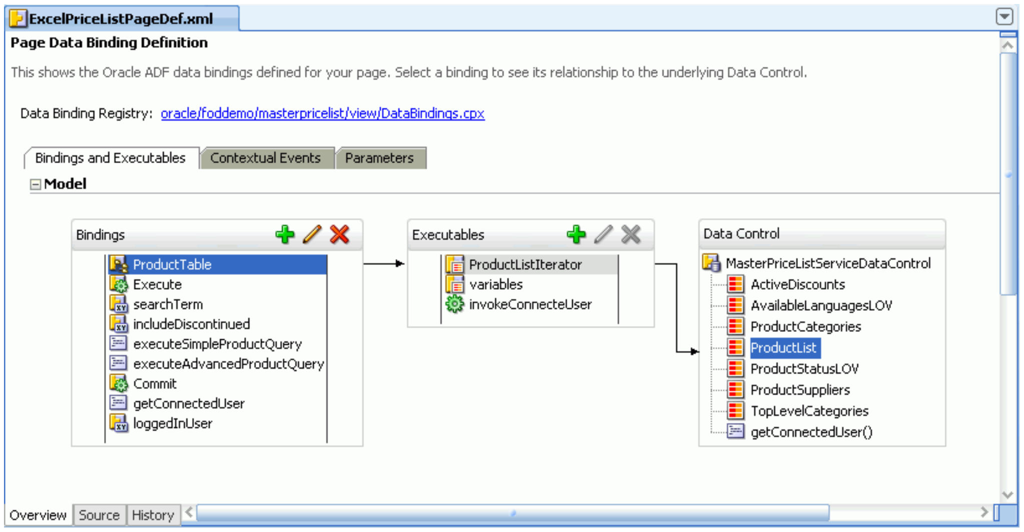
   Note: JDeveloper creates a page definition file's name based on the name of the JSF page you choose. If you want a page definition file's name to indicate an association with a particular workbook or worksheet, choose this name when creating the JSF page.

   ---

2. In the Application Navigator, select the page, right-click and select **Go to Page Definition**.
3. In the Confirm Create New Page Definition dialog, click **Yes**.
4. Add the bindings that you require for your integrated Excel workbook to the page definition file.
5. Save the page definition file.

   Figure 4–2 shows the `ExcelPriceListPageDef.xml` page definition file that the Price List worksheet in the `EditPriceList-DT.xlsx` workbook references.

   Figure 4–2 Page Definition File with Bindings for an Integrated Excel Workbook

   ExcelPriceListPageDef.xml

   Page Data Binding Definition

   This shows the Oracle ADF data bindings defined for your page. Select a binding to see its relationship to the underlying Data Control.

   Data Binding Registry: oracle/foddemo/masterpricelist/view/DataBindings.cpx

   Bindings and Executables | Contextual Events | Parameters

   Model

   Bindings: ProductTable, Execute, searchTerm, includeDiscontinued, executeSimpleProductQuery, executeAdvancedProductQuery, Commit, getConnectedUser, loggedInUser

   Executables: ProductListIterator, variables, invokeConnecteUser

   Data Control: MasterPriceListServiceDataControl, ActiveDiscounts, AvailableLanguagesLOV, ProductCategories, ProductList, ProductStatusLOV, ProductSuppliers, TopLevelCategories, getConnectedUser()

   Overview | Source | History

   For information about working with page definition files, see the "Working with Page Definition Files" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

6. Make and run your Fusion web application if you plan to run the integrated Excel workbook in test mode or publish it.

4.3.2 What Happens When You Create a Page Definition File

JDeveloper creates the `DataBindings.cpx` file the first time that you add a page definition file in your JDeveloper project using the procedure described in Section 4.3.1, "How to Create a Page Definition File for an Integrated Excel Workbook."

The `DataBindings.cpx` file defines the binding context for the Fusion web application and provides the metadata from which the Oracle ADF bindings are created at runtime. Information about working with this file can be found in the "Working with the DataBindings.cpx File" section of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. Information about the elements and attributes in the file can be found the "DataBindings.cpx" section of the same guide.

4.3.3 Reloading a Page Definition File in an Excel Workbook

If you make changes in your JDeveloper desktop integration project to a page definition file that is associated with an Excel worksheet, rebuild the JDeveloper desktop integration project and reload the page definition file in the Excel worksheet to ensure that the changes appear in the ADF Desktop Integration task pane. You associate a page definition file with an Excel worksheet when you choose the page definition file, as described in Section 4.4.3, "How to Configure a New Integrated Excel Workbook."

The **Oracle ADF** tab provides a button that reloads all page definition files in an Excel workbook.

Errors may occur when you switch an integrated Excel workbook from design mode to runtime if you do not rebuild the JDeveloper desktop integration project and restart the application after making changes to a page definition file. For example, if you:

- Remove an element in a page definition file
- Do not rebuild and restart the Fusion web application
- Or do not reload the page definition file in the integrated Excel workbook

an error message such as the following may appear when you attempt to switch a workbook to test mode:

```
[ADFDI-05530] unable to initialize worksheet: MyWorksheet
[ADFDI-05517] unable to find control MyBindingThatWasRemoved
```

To reload page definition files in an Excel workbook:

1. Ensure that you have saved the updated page definition file in JDeveloper.
2. In the Excel workbook, click the **Refresh Bindings** button in the Components group of the **Oracle ADF** tab.

   For information about the Refresh Bindings button, see Section 5.1, "Introduction to Development Tools."

4.3.4 What You May Need to Know About Page Definition Files in an Integrated Excel Workbook

Note the following points about page definition files in an ADF Desktop Integration project.

- Integrating Multiple Excel Worksheets – You can integrate multiple worksheets in an Excel workbook with a Fusion web application. You associate a page definition file with each worksheet as described in Section 4.4.4, "How to Add Additional Worksheets to an Integrated Excel Workbook."

- EL Expressions in a Page Definition File – Use the following syntax to write EL expressions in a page definition file:

  ```
  Dynamic (${})
  ```

  Do not use the syntax `Deferred (#{})` to write EL expressions. EL expressions using this syntax generate errors as they attempt to access the ADF Faces context which is not available

---

Note: EL expressions that you write for ADF Desktop Integration components in the integrated Excel workbook, such as ADF Input Text, must use the `Deferred (#{})` syntax.

4.4 Adding Integrated Excel Workbook to a Fusion Web Application

Before you start using an Excel workbook with Oracle ADF functionality, you must integrate and configure the workbook with your Fusion web application in the following way:

1. Create a new integrated Excel workbook, or enable an existing workbook to integrate it with Fusion web application. You can choose either of the two methods.
2. Configure your integrated Excel workbook by setting several properties after the workbook is enabled for ADF Desktop Integration.
3. Add additional worksheets, if required.

After you complete these steps, you can add Oracle ADF functionality using the tools provided by the ADF Desktop Integration module.

You can add an integrated Excel workbook to your Fusion web application in JDeveloper, or create an Excel workbook and enable it for ADF Desktop Integration.

4.4.1 How to Add an Integrated Excel Workbook in JDeveloper

You can add an integrated Excel workbook to a Fusion web application in JDeveloper from the New Gallery.

To add an integrated Excel workbook in JDeveloper:

1. Open your Fusion web application in JDeveloper.

   Ensure that the application is configured for ADF Desktop Integration. For more information about how to configure your application with ADF Desktop Integration, see Section 4.2, "Adding ADF Desktop Integration to a Fusion Web Application."
2. In the Application Navigator, select the project, such as ViewController, to which you want to add your new workbook.
3. From the **File** menu, choose **New**.
4. In the New Gallery dialog, expand **Client Tier**, select **ADF Desktop Integration**, and then select **Microsoft Excel Workbook**, and click **OK**.

5. In the Create ADF Desktop Integration-Enabled Excel Workbook dialog, if required, edit the file name of the workbook and its location. By default, the integrated Excel workbook is saved as `adfdi-workbook.xlsx` in the `\src\excel` directory of the selected project.

6. Click **OK**.

JDeveloper adds an integrated Excel workbook into your Fusion web application. In Application Navigator, double-click the workbook to open and configure it as desired. For more information about how to configure the integrated Excel workbook, see Section 4.4.3, "How to Configure a New Integrated Excel Workbook."

> **Tip:** If you create and add an integrated Excel workbook in JDeveloper, you may not be required to configure various workbook properties, as described in Section 4.4.3, "How to Configure a New Integrated Excel Workbook."

4.4.2 How to Enable ADF Desktop Integration in a Workbook

If you want to integrate an existing workbook with your Fusion web application, you must enable ADF Desktop Integration for the workbook. For information about the file formats of Excel workbooks that you can convert for integration with a Fusion web application, see Section 3.2, "Required Oracle ADF Modules and Third-Party Software."

To enable ADF Desktop Integration in an Excel workbook:

1. In Excel, open your workbook, or create a new blank workbook.

2. In the **Oracle ADF** tab, click **Workbook Properties**.

3. In the Enable Workbook dialog, click **Yes**.

   The ADF Desktop Integration framework prepares your workbook and initializes the ADF Desktop Integration Designer task pane.

4. Save the workbook.

Although you can store the Excel workbooks that you integrate with Fusion web applications anywhere you choose, there are several advantages to storing them with the other files of your Fusion web application. Some of these advantages are:

- Source control of the workbooks
- Facilitating the download of workbooks from web pages
- The file system folder picker that appears the first time a workbook is opened defaults to the location where you store the workbook

For example, the Master Price List module of the Fusion Order Demo stores the Excel workbooks it integrates in the following subdirectory:

```
FOD_
HOME\MasterPriceList\ViewController\src\oracle\foddemo\masterpri
celist\excel
```

where *`FOD_HOME`* is the root directory that stores the source files for the Fusion Order Demo application.

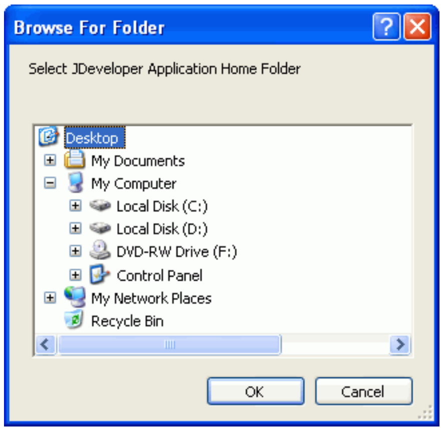
4.4.3 How to Configure a New Integrated Excel Workbook

After the workbook is integrated with ADF Desktop Integration, you must configure it.

To configure a new integrated Excel workbook:

1. When you enable ADF Desktop Integration in a workbook, the Browse for Folder dialog automatically appears, as illustrated in Figure 4–3.

Figure 4–3 Browse For Folder Dialog



   Use the Browse for Folder dialog to select the JDeveloper application home directory. In a typical JDeveloper project, the JDeveloper application home directory stores the *application_name.jws* file. The value you select is assigned to the `ApplicationHomeFolder` workbook property.

   Note: The Browse for Folder dialog does not appear if the workbook is located within the JDeveloper application workspace. In such a case, the value of `ApplicationHomeFolder` workbook property is assigned automatically.

   ---

2. In the Workbook group of the **Oracle ADF** tab, click **Workbook Properties**.
3. In the Edit Workbook Properties dialog, set or verify the values for the following properties so that you can switch between design mode and test mode as you configure your workbook:

   - `ApplicationHomeFolder`

     The value for this property corresponds to the absolute path for the root directory of the JDeveloper application workspace (`.jws`). You specified this value in Step 1.

     Note: If you are opening the Excel file after moving your application directory, ensure that the `ApplicationHomeFolder` property's value reflects the correct path.

   - `Project`

     The value for this property corresponds to the name of JDeveloper project (`.jpr`) in the JDeveloper application workspace. To change the project, click the ellipsis (...) button and choose the project from the Project dialog that lists the projects defined in the JDeveloper application workspace.

     By default, `Project` is set to the name of the project that contains the Excel document. ADF Desktop Integration loads the names of the available projects from the *application_name.jws* specified as a value for `ApplicationHomeFolder`.

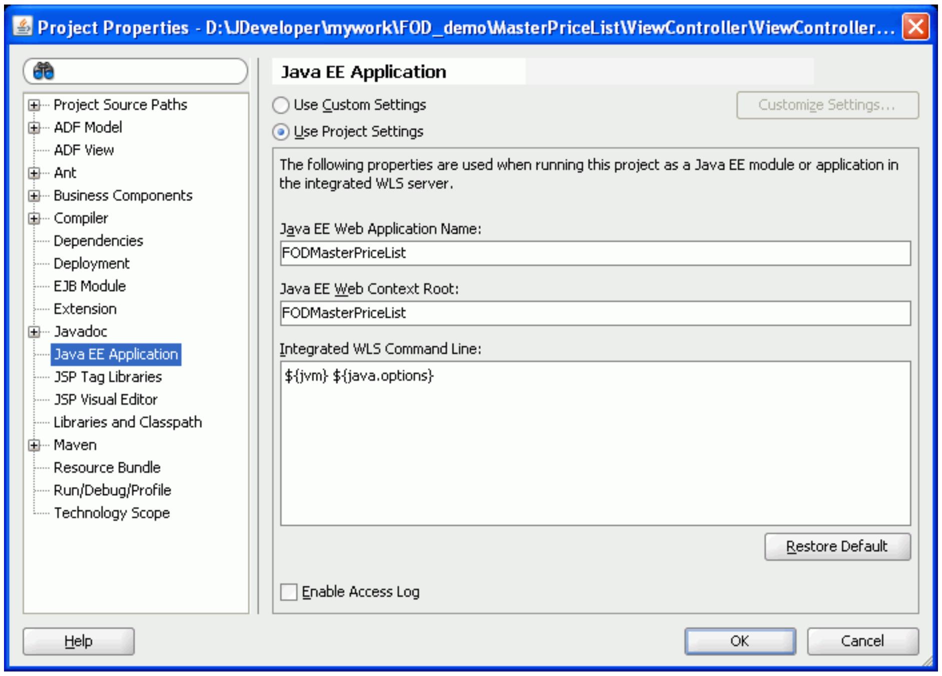
- `WebAppRoot`

  Set the value for this property to the fully qualified URL for the web context root that you want to integrate your desktop application with. The fully qualified URL has the following format:

  ```
  http://<hostname>:<portnumber>/context-root
  ```

  In JDeveloper, you specify the web context root (`context-root`) in the Java EE Application page of the Project Properties dialog. Figure 4–4 shows the web context root for the Master Price List module.

Figure 4–4 Setting Web Context Root in JDeveloper



Project Properties - D:\JDeveloper\mywork\FOD_demo\MasterPriceList\ViewController\ViewController...

Java EE Application

Use Custom Settings [✓ Use Project Settings]

The following properties are used when running this project as a Java EE module or application in the integrated WLS server.

Java EE Web Application Name: FODMasterPriceList

Java EE Web Context Root: FODMasterPriceList

Integrated WLS Command Line: ${jvm} ${java.options}

Enable Access Log

  Note that the fully qualified URL is similar to the following if you set up a test environment on your system using the Master Price List module:

  ```
  http://127.0.0.1:7101/FODMasterPriceList
  ```

  For information about how to verify that the Fusion web application is online and that it supports ADF Desktop Integration, see Section C.1, "Verifying That Your Fusion Web Application Supports ADF Desktop Integration."

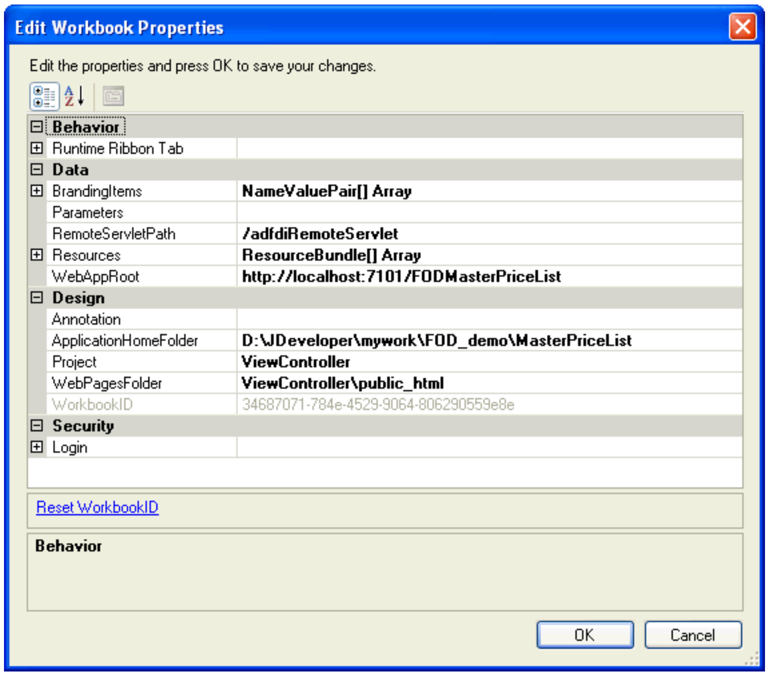
  If you are integrating an Excel file with a secure Fusion web application, it is recommended that you use the `https` protocol while entering `WebAppRoot`'s value. For more information about securing your Fusion web application, see *Oracle Fusion Middleware Programming Security for Oracle WebLogic Server*.

- `WebPagesFolder`

  Set the value for this property to the directory that contains web pages for the Fusion web application. The directory path should be relative to the value of `ApplicationHomeFolder`. For example, in the `EditPriceList-DT.xlsx` workbook, `WebPagesFolder` is set to `ViewController\public_html`.

Figure 4–5 shows an implementation of workbook properties in the Edit Workbook Properties dialog of Master Price List module's `EditPriceList-DT.xlsx` workbook.

Figure 4–5 Edit Workbook Properties Dialog



Edit Workbook Properties

Edit the properties and press OK to save your changes.

Behavior	
Runtime Ribbon Tab	
Data	
BrandingItems	NameValuePair[] Array
Parameters	
RemoteServletPath	/adfdiRemoteServlet
Resources	ResourceBundle[] Array
WebAppRoot	http://localhost:7101/FODMasterPriceList
Design	
Annotation	
ApplicationHomeFolder	D:\JDeveloper\mywork\FOD_demo\MasterPriceList
Project	ViewController
WebPagesFolder	ViewController\public_html
WorkbookID	34687071-784e-4529-9064-806290559e8e
Security	
Login	

Reset WorkbookID

Behavior

OK Cancel

4. Click **OK**.
5. In the Workbook group of the **Oracle ADF** tab, click **Worksheet Properties**.
6. In the Edit Worksheet Properties dialog, click the ellipsis button (**...**) beside the **Page Definition** input field and select a page definition file from the dialog that appears.
7. Click **OK**.

   The Excel worksheet appears with ADF Desktop Integration in Excel's task pane. The bindings of the page definition file, you selected in Step 6, appear in the **Bindings** tab.
8. Save the Excel workbook.

4.4.4 How to Add Additional Worksheets to an Integrated Excel Workbook

To use Oracle ADF functionality, you must associate each worksheet with a page definition file. You associate a page definition file with a worksheet when you add a worksheet to the integrated Excel workbook. You can integrate multiple worksheets in an integrated Excel workbook with a Fusion web application. Use a different page definition file for each worksheet in the integrated Excel workbook.

To associate a page definition file with an Excel worksheet:

1. While the Excel workbook is in design mode, click the **Home** tab in Excel ribbon, and then choose **Insert** > **Insert Sheet** in the **Cells** group.
2. In the Choose Page Definition dialog, select the page definition file.

This populates the bindings palette in the ADF Desktop Integration task pane with the bindings contained in the page definition file. You can now configure the worksheet with Oracle ADF functionality.

5

Getting Started with the Development Tools

This chapter describes how to use the development tools provided by ADF Desktop Integration. It provides an overview of the development environment that Oracle ADF exposes within Excel and goes on to describe how you display and use the different elements of this environment.

This chapter includes the following sections:

5.1 Introduction to Development Tools

ADF Desktop Integration provides several tools that you use to configure Excel workbooks so that they can access Oracle ADF functionality. The tools are available in the **Oracle ADF** tab and in the ADF Desktop Integration Designer task pane.

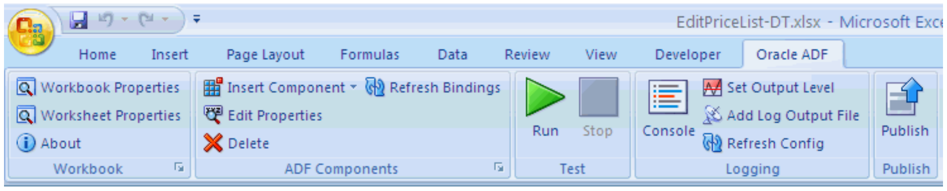
Before you start using the development tools, you must know that there are two modes in which you can work while you configure the Excel workbook that you integrate with a Fusion web application. The first mode is the design mode, and the second mode is the test mode.

In *design mode*, you use the tools provided by Oracle ADF in Excel to design and configure your integrated Excel workbook. In *test mode*, you can view and test the changes you make in design mode in the same way that the end user views the published integrated Excel workbook.

5.2 Oracle ADF Tab

The **Oracle ADF** tab, also shown in Figure 5–1, provides various buttons in design mode.

Figure 5–1 Oracle ADF Tab in Design Mode



You can use **Oracle ADF** tab buttons to invoke the actions described in Table 5–1.

Table 5–1 Oracle ADF Tab Options

In this group...	Click this button...	To...	Mode when the button is available...
Workbook	Workbook Properties	Display the Edit Workbook Properties dialog to view and edit integrated Excel workbook properties. The button is also used to enable ADF Desktop Integration in a non-integrated Excel workbook.	Design
Workbook	Worksheet Properties	Display the Edit Worksheet Properties dialog to view and edit the current worksheet properties.	Design
Workbook	About	Open the About Oracle ADF 11*g* Desktop Integration dialog that provides version and property information of integrated Excel workbook. The button is also available in non-integrated Excel workbooks after ADF Desktop Integration is installed.	Design, Test, Runtime
ADF Components	Insert Component	Display a dropdown list of Oracle ADF components that you can insert in the selected cell.	Design
ADF Components	Edit Properties	Display the property inspector window to view and edit component properties of the selected component.	Design
ADF Components	Delete	Delete the selected component from the Excel worksheet.	Design

Table 5–1 (Cont.) Oracle ADF Tab Options

In this group...	Click this button...	To...	Mode when the button is available...
ADF Components	Refresh Bindings	■ Reload the application workspace file (`.jws`) and project file (`.jpr`) referenced by the workbook properties of the integrated Excel workbook. ■ Refresh all information from the page definition files used in the active integrated Excel workbook. Any modifications that you made to the page definition files in the JDeveloper project now become available in the Excel workbook. For more information, see Section 4.3.3, "Reloading a Page Definition File in an Excel Workbook."	Design
Test	Run	Switch the Excel workbook from design mode to test mode. This button is active only when you are in design mode.	Design
Test	Stop	Switch the Excel workbook from test mode to design mode. This button is active only when you are in test mode. For more information about switching between design mode and test mode, see Section 13.3, "Testing Your Integrated Excel Workbook."	Test
Logging	Console	Display a dialog to review the client-side log entries. For more information, see Section C.3.2, "About Client-Side Logging."	Design, Test
Logging	Refresh Config	Reload the ADF Desktop Integration configuration file. For more information, see Section C.3.2, "About Client-Side Logging."	Design, Test
Logging	Set Output Level	Display the Set Output Level dialog to choose client-side log output level. For more information, see Section C.3.2, "About Client-Side Logging."	Design, Test
Logging	Add Log Output File	Create a new temporary logging listener to act as a client-side log output file. For more information, see Section C.3.2, "About Client-Side Logging."	Design, Test

Table 5–1 (Cont.) Oracle ADF Tab Options

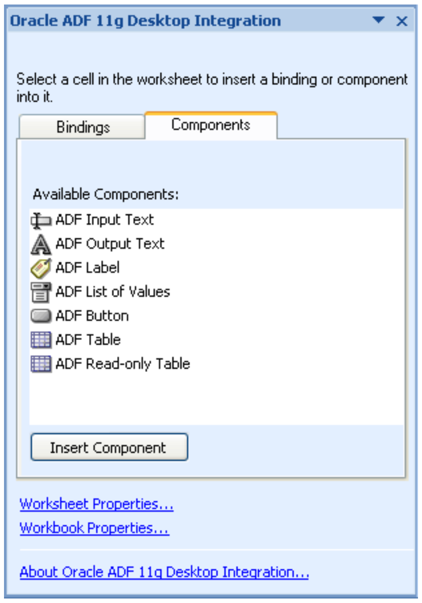
In this group...	Click this button...	To...	Mode when the button is available...
Publish		Publish the Excel workbook after you complete the integration between the Excel workbook and the Fusion web application. For more information about publishing an integrated Excel workbook, see Chapter 14, "Deploying Your Integrated Excel Workbook."	Design

Tip: For quick and easy access, you can add **Oracle ADF** tab buttons to the Excel Quick Access toolbar.

5.3 ADF Desktop Integration Designer Task Pane

Figure 5–2 displays the ADF Desktop Integration Designer task pane.

Figure 5–2 ADF Desktop Integration Designer Task Pane


Oracle ADF 11g Desktop Integration

Select a cell in the worksheet to insert a binding or component into it.

You can invoke the ADF Desktop Integration Designer task pane through launcher buttons available in the bottom-right corner of the Workbook and ADF Components group on the **Oracle ADF** tab, as illustrated in Figure 5–3.

Figure 5–3 ADF Desktop Integration Designer Task Pane Launcher Buttons

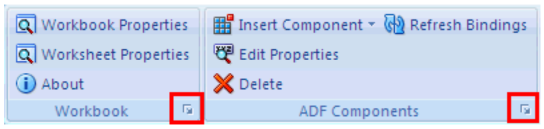


Table 5–2 lists the view tabs and links that appear in the task pane, provides a brief description of each item.

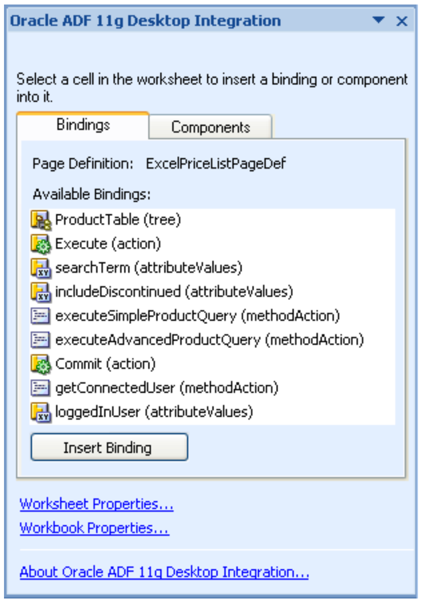
Table 5–2 Overview of ADF Desktop Integration Designer Task Pane

Task Pane UI Element	Description
Workbook Properties	Click to display the Edit Workbook Properties dialog. This dialog enables you to view and edit properties that affect the whole workbook. Examples include properties that reference the directory paths to page definition files, the URL for your Fusion web application, and so on.
Worksheet Properties	Click to display the Edit Worksheet Properties dialog. This dialog enables you to view and edit properties specific to the active worksheet. An example is the file name of the page definition file that you associate with the worksheet.
About	Click to display the About dialog. This dialog provides the version and property information that can be useful when troubleshooting an integrated Excel workbook. For example, it provides information about the underlying Microsoft .NET and Oracle ADF frameworks that support an integrated Excel workbook.

5.4 Using the Bindings Palette

The bindings palette presents the available Oracle ADF bindings that you can insert into the Excel worksheet. The page definition file for the current Excel worksheet determines what Oracle ADF bindings appear in the bindings palette. Figure 5–4 shows a bindings palette populated with Oracle ADF bindings in the ADF Desktop Integration Designer task pane. Note that the bindings palette does not display bindings that an integrated Excel workbook cannot use, so the bindings that appear may differ from those that appear in the page definition file viewed in JDeveloper.

Figure 5–4 ADF Bindings Palette in the ADF Desktop Integration Designer Task Pane


Oracle ADF 11g Desktop Integration
Bindings
Components
Insert Binding
Worksheet Properties...
Workbook Properties...
About Oracle ADF 11g Desktop Integration...

Select a cell in the worksheet to insert a binding or component into it.

Page Definition: ExcelPriceListPageDef

Available Bindings:

- ProductTable (tree)
- Execute (action)
- searchTerm (attributeValues)
- includeDiscontinued (attributeValues)
- executeSimpleProductQuery (methodAction)
- executeAdvancedProductQuery (methodAction)
- Commit (action)
- getConnectedUser (methodAction)
- loggedInUser (attributeValues)

You use the bindings palette in design mode to insert a binding. When you attempt to insert a binding, ADF Desktop Integration inserts an Oracle ADF component that references the binding you selected. ADF Desktop Integration also prepopulates the properties of the Oracle ADF component with appropriate values. For example, if you insert a binding, such as the **Commit (action)** binding illustrated in Figure 5–4, the property inspector for an Oracle ADF Button component appears. This Oracle ADF Button component has values specified for its `ClickActionSet` that include invoking the `Commit` action binding.

To insert an Oracle ADF binding, select the cell to anchor the Oracle ADF component that is going to reference the binding in the Excel worksheet, and then insert the binding in one of the following ways:

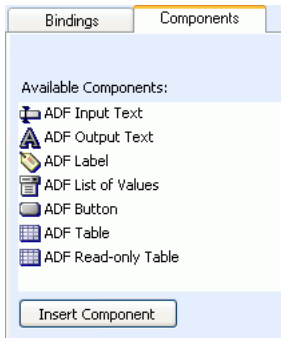
- Double-click the Oracle ADF control binding you want to insert.
- Select the control binding and click **Insert Binding** in the ADF Desktop Integration Designer task pane.

  A property inspector for the Oracle ADF component that is associated with the binding you attempt to insert appears. In some instances, you may be prompted to select one Oracle ADF component from a list of Oracle ADF components where multiple Oracle ADF components can be associated with the binding. After you select an Oracle ADF component from the list, a property inspector appears.

5.5 Using the Components Palette

The components palette displays the available ADF Desktop Integration components that you can insert into an Excel worksheet. Figure 5–5 shows the components palette as it appears in the ADF Desktop Integration Designer task pane.

Figure 5–5 Oracle ADF Components Palette in the ADF Desktop Integration Designer Task Pane



You use the components palette in design mode to insert an Oracle ADF component. First, select the cell to anchor the Oracle ADF component in the Excel worksheet, and then insert the Oracle ADF component in one of the following ways:

- Double-click the Oracle ADF component you want to insert.
- Select the component and click **Insert Component** in the ADF Desktop Integration Designer task pane.

In both cases, the Oracle ADF component’s property inspector appears. Use the property inspector to specify values for the component before you complete its insertion into the Excel worksheet.

Note: The ADF Desktop Integration components are also available in the **Insert Component** dropdown list of **Oracle ADF** tab.

5.6 Using the Property Inspector

The property inspector is a dialog that enables you to view and edit the properties of Oracle ADF bindings, Oracle ADF components, Excel worksheets, or the Excel workbook. You can open the property inspector in one of the following ways:

- Select the component or binding, and click the **Edit Properties** icon in the **Oracle ADF** tab.
- Select the component or binding, right-click and choose **Edit ADF Component Properties** from the context menu.

  Note that the ADF Button does not support the right-click action, click the button to open the property inspector dialog.

The property inspector also appears automatically after you insert an Oracle ADF binding or component into an Excel worksheet. Figure 5–6 shows a property inspector where you can view and edit the properties of an Oracle ADF Button component.

At design time, you can edit key properties of certain Oracle ADF components by editing the Excel cell where the component appears. For example, you can edit the Value property of ADF Label and ADF Input Text components by editing the value displayed in the cell.

Note: The property inspector does not validate that values you enter for a property or combinations of properties are valid. Invalid values may cause runtime errors. To avoid runtime errors, make sure you specify valid values for properties in the property inspector.

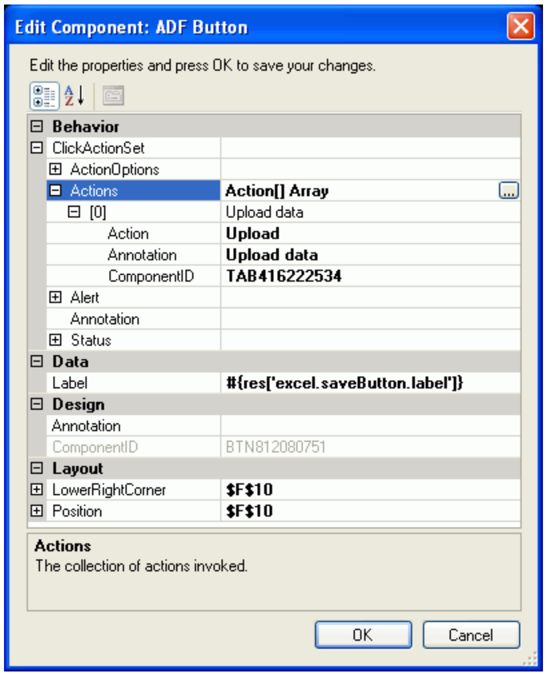
You can display the properties in an alphabetical list or in a list where the properties are grouped by categories such as Behavior, Data, and so on. Table 5–3 describes the buttons that you can use to change how properties display in the property inspector.

Table 5–3 Buttons to Configure Properties Display in Property Inspector

Button	Description
⊞	Use this button to display the properties according to category.
A↓Z	Use this button to display the properties in an alphabetical list.

In Figure 5–6, the property inspector displays the properties grouped by category.

Figure 5–6 Property Inspector Window for an Oracle ADF Component


Edit Component: ADF Button
Edit the properties and press OK to save your changes.
OK
Cancel

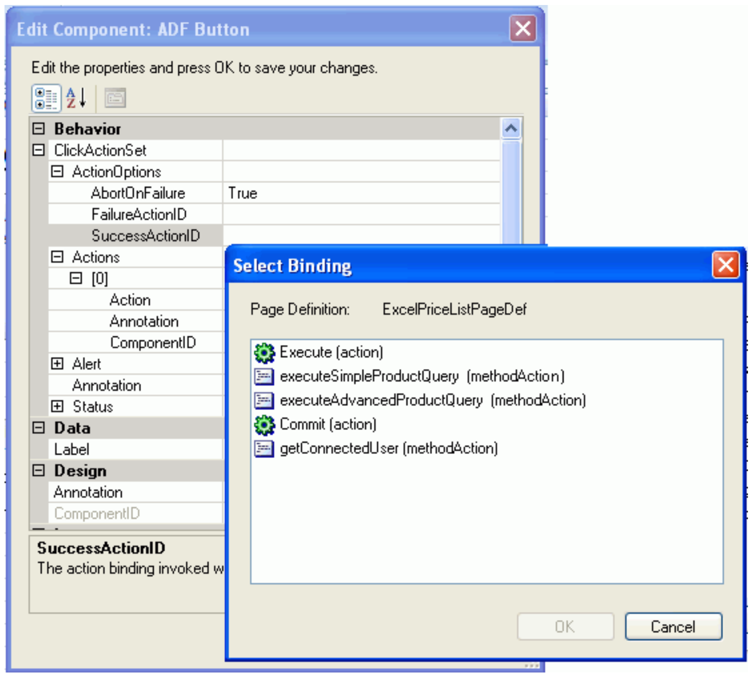
⊟ Behavior	
⊟ ClickActionSet	
⊞ ActionOptions	
⊟ Actions	Action[] Array
⊟ [0]	Upload data
Action	Upload
Annotation	Upload data
ComponentID	TAB416222534
⊞ Alert	
Annotation	
⊞ Status	
⊟ Data	
Label	#{res['excel.saveButton.label']}
⊟ Design	
Annotation	
ComponentID	BTN812080751
⊟ Layout	
⊞ LowerRightCorner	F10
⊞ Position	F10

Actions
The collection of actions invoked.

5.7 Using the Binding ID Picker

The binding ID picker is a dialog that enables you to select Oracle ADF bindings at design time to configure the behavior of Oracle ADF components at runtime. You invoke the binding ID picker from the property inspector. The binding ID picker filters the Oracle ADF bindings that appear, based on the type of binding that the Oracle ADF component property accepts. For example, the `SuccessActionID` property for an ADF Button component supports only action bindings. Therefore, the binding ID picker filters the bindings from the page definition file so that only action bindings appear, as illustrated in Figure 5–7.

Figure 5–7 Binding ID Picker



Edit Component: ADF Button

Edit the properties and press OK to save your changes.

Behavior
ClickActionSet
ActionOptions
AbortOnFailure True
FailureActionID
SuccessActionID
Actions
[0]
Action
Annotation
ComponentID
Alert
Annotation
Status
Data
Label
Design
Annotation
ComponentID

SuccessActionID
The action binding invoked w

Select Binding

Page Definition: ExcelPriceListPageDef

Execute (action)
executeSimpleProductQuery (methodAction)
executeAdvancedProductQuery (methodAction)
Commit (action)
getConnectedUser (methodAction)

For more information about ADF Desktop Integration component properties and the bindings they support, see Appendix A, "ADF Desktop Integration Component Properties and Actions."

5.8 Using the Expression Builder

You use the expression builder to write Expression Language, or EL, expressions that configure the behavior of components at runtime in the Excel workbook. You invoke the expression builder from the property inspector of component properties that support EL expressions. For example, the `Label` property in Figure 5–8 supports EL expressions and, as a result, you can invoke the expression builder to set a value for this property.

You can reference bindings in the EL expressions that you write. Note that the expression builder does not filter bindings. It displays all bindings that the page definition file exposes. See Table 4–1 to identify the types of bindings that each ADF Desktop Integration component supports.

To add an expression in the **Expression** box, select the item and click **Insert Into Expression**. You can also double-click the item to add it in the **Expression** box. Table 5–4 describes the folders available in the expression builder.

Figure 5–8 Expression Builder

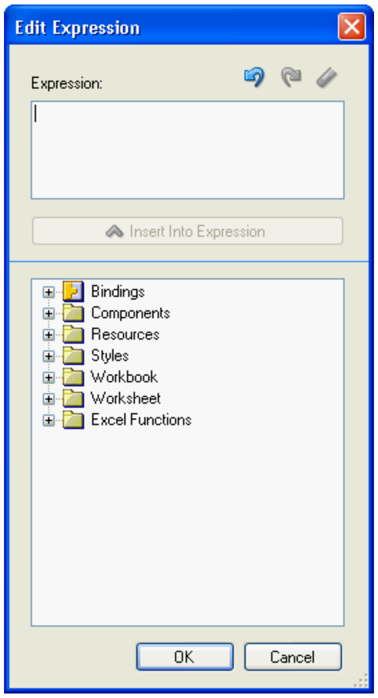


Table 5–4 Expression Builder Folders

Folder Name	Description
Bindings	Lists the bindings supported in ADF Desktop Integration from the current worksheet's page definition.
Components	Lists the ADF components available in the current worksheet.
Resources	Lists the resource bundles registered in `Workbook.Resources` along with the built-in resource bundle `_ADFDIres`.
Styles	Lists all Excel styles defined in the current workbook. For more information, see Section 9.2, "Working with Styles.".
Workbook	Lists parameters defined in `Workbook.Parameters`.
Worksheet	Lists the `errors` expression.
Excel Functions	Lists sample Excel functions that you can use with ADF Desktop Integration. For more information, see Excel's documentation.

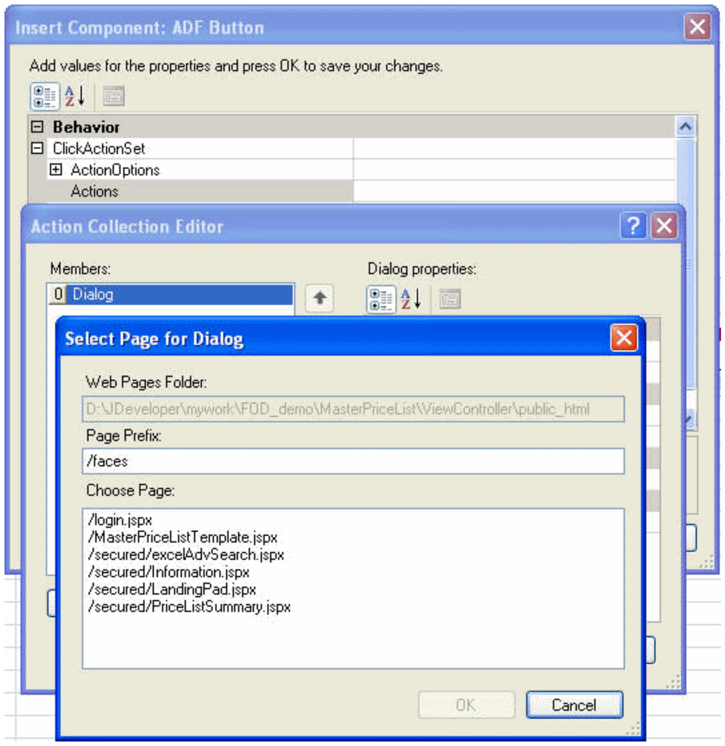
For more information about using the expression builder, see Section 9.3, "Applying Styles Dynamically Using EL Expressions." For information about the syntax of EL expressions in ADF Desktop Integration, and guidelines on how you write these expressions, see Appendix B, "ADF Desktop Integration EL Expressions."

5.9 Using the Web Page Picker

Use the web page picker to select a web page from your Fusion web application. At runtime, an Oracle ADF component, for example an Oracle ADF Button component, can invoke the web page that you associate with the Oracle ADF component.

You can invoke the web page picker when you add a `Dialog` action to an action set in the Action Collector Editor. You use the web page picker to specify a web page for the `Page` property of the `Dialog` action, as illustrated in Figure 5–9.

Figure 5–9 Web Page Picker Dialog



Web Pages Folder: D:\JDeveloper\mywork\FOD_demo\MasterPriceList\ViewController\public_html

Page Prefix: /faces

Choose Page:

```
/login.jspx
/MasterPriceListTemplate.jspx
/secured/excelAdvSearch.jspx
/secured/Information.jspx
/secured/LandingPad.jspx
/secured/PriceListSummary.jspx
```

For more information about displaying web pages in your integrated Excel workbook, see Section 8.4, "Displaying Web Pages from a Fusion Web Application."

5.10 Using the File System Folder Picker

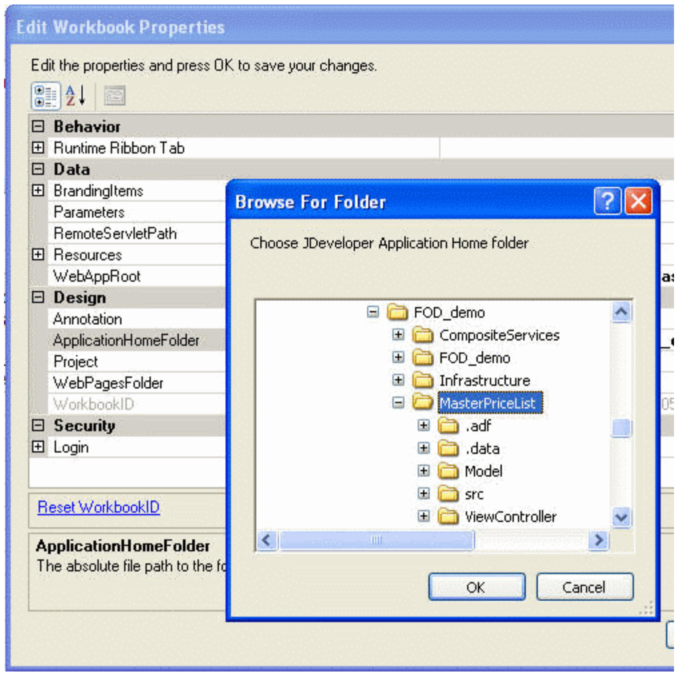
Use the file system folder picker to navigate over the Windows file system and select folders. You use this picker to specify values for the following workbook properties:

- `ApplicationHomeFolder`
- `WebPagesFolder`

The first time you open an Excel workbook the picker appears so that you can set values for the previously listed properties. For more information about opening an Excel workbook for the first time and the properties you set, see Section 4.4.3, "How to Configure a New Integrated Excel Workbook."

Figure 5–10 shows the file system folder picker selecting a value for the `ApplicationHomeFolder` workbook property.

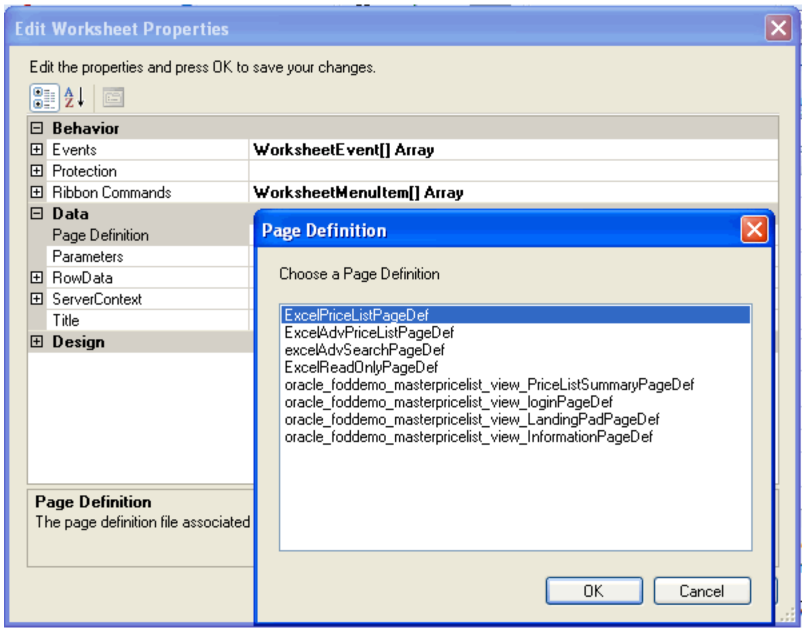
Figure 5–10 File System Folder Picker



5.11 Using the Page Definition Picker

Use the page definition picker to select the page definition ID of a page definition file and associate the file with a worksheet. The picker appears the first time that you activate a worksheet in an integrated Excel workbook. It can also be invoked when you attempt to set a value for the worksheet property, `PageDefinition`, as illustrated in Figure 5–11.

Figure 5–11 Page Definition Picker

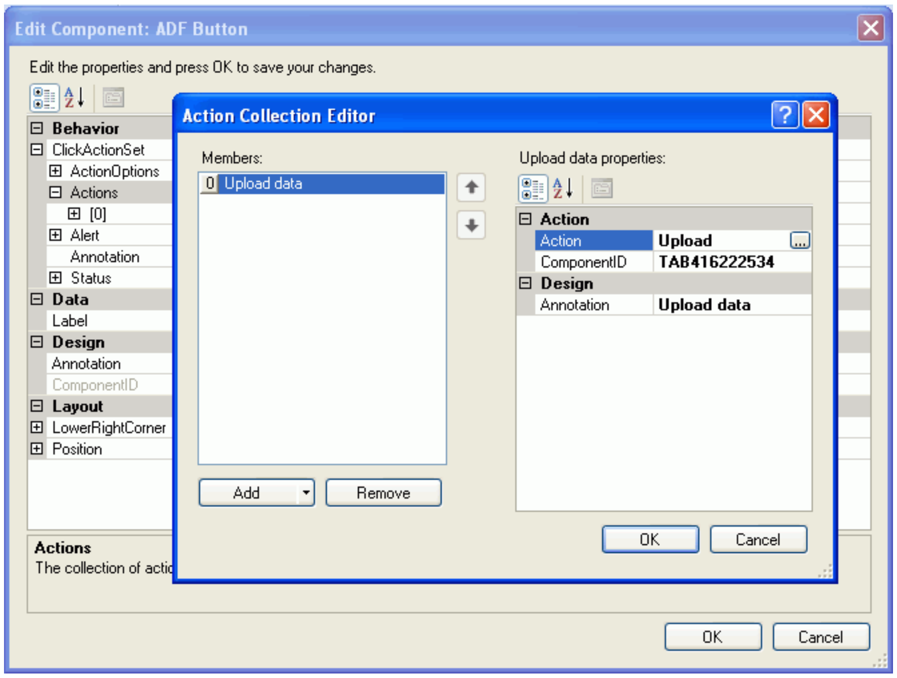

Edit Worksheet Properties
Edit the properties and press OK to save your changes.
Behavior
Events
WorksheetEvent[] Array
Protection
Ribbon Commands
WorksheetMenuItem[] Array
Data
Page Definition
Parameters
RowData
ServerContext
Title
Design
Page Definition
The page definition file associated
Page Definition
Choose a Page Definition
ExcelPriceListPageDef
ExcelAdvPriceListPageDef
excelAdvSearchPageDef
ExcelReadOnlyPageDef
oracle_foddemo_masterpricelist_view_PriceListSummaryPageDef
oracle_foddemo_masterpricelist_view_loginPageDef
oracle_foddemo_masterpricelist_view_LandingPadPageDef
oracle_foddemo_masterpricelist_view_InformationPageDef
OK
Cancel

For more information about page definition files, see Section 4.3, "Working with Page Definition Files for an Integrated Excel Workbook."

5.12 Using the Collection Editors

ADF Desktop Integration uses collection editors to manage the properties of elements in a collection. The title that appears in a collection editor's title bar describes what the collection editor enables you to configure. Examples of titles for collection editors include **CacheDataContext Collection Editor**, **TableColumn Collection Editor**, and the **Action Collection Editor**. These collection editors allow you to configure collections of cached data, table columns in the ADF Table component, and actions in an action set. Figure 5–12 shows the collection editor that configures an action set for the **Search** button that appears at runtime in the Master Price List module's `EditPriceList-DT.xlsx` workbook.

Figure 5–12 Collection Editor



Edit Component: ADF Button

Edit the properties and press OK to save your changes.

- **Behavior**
  - ClickActionSet
    - ActionOptions
    - Actions
      - [0]
    - Alert
    - Annotation
    - Status
- **Data**
  - Label
- **Design**
  - Annotation
  - ComponentID
- **Layout**
  - LowerRightCorner
  - Position

Actions
The collection of actic

Action Collection Editor

Members:

0 Upload data

Upload data properties:

Action	
Action	**Upload**
ComponentID	**TAB416222534**
Design	
Annotation	**Upload data**

Tip: Write a description in the Annotation field for each element that you add to the Action Collection Editor. The description you write appears in the **Members** list view and, depending on the description you write, may be more meaningful than the default entry that ADF Desktop Integration generates.

6 Working with ADF Desktop Integration Form-Type Components

This chapter describes how you can insert and configure components that ADF Desktop Integration provides to allow end users to manage data retrieved from a Fusion web application.

This chapter includes the following sections:

6.1 Introduction to ADF Desktop Integration Form-Type Components

Rather than expose an ADF Form component in the components palette described in Section 5.5, "Using the Components Palette," ADF Desktop Integration uses the following components to create form-type functionality in an integrated Excel workbook:

- ADF Input Text
- ADF Output Text
- ADF Label
- ADF List of Values
- ADF Button

Note: ADF Desktop Integration does not support components inserted in a merged cell.

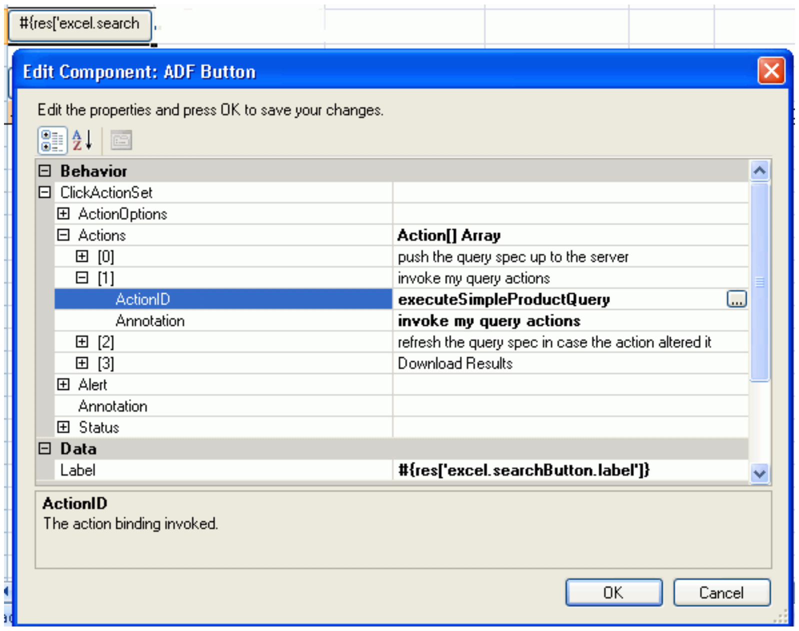
6.2 Inserting an ADF Button Component

The ADF Button component renders a button in the Excel worksheet at runtime. End users click this button to invoke one or more actions specified by the `ClickActionSet` group of properties.

The `LowerRightCorner` and `Position` properties determine the area that the button occupies on the Excel worksheet at runtime.

Figure 6–1 shows a button in an Excel worksheet in design mode. The property inspector for the button is in the foreground. When an end user clicks the button at runtime, it invokes the array of actions specified by `ClickActionSet`.

Figure 6–1 ADF Button Component



Edit the properties and press OK to save your changes.

⊟ **Behavior**	
⊟ ClickActionSet	
⊞ ActionOptions	
⊟ Actions	**Action[] Array**
⊞ [0]	push the query spec up to the server
⊟ [1]	invoke my query actions
ActionID	**executeSimpleProductQuery**
Annotation	**invoke my query actions**
⊞ [2]	refresh the query spec in case the action altered it
⊞ [3]	Download Results
⊞ Alert	
Annotation	
⊞ Status	
⊟ **Data**	
Label	**#{res['excel.searchButton.label']}**

ActionID
The action binding invoked.

For more information about the properties of the ADF Button component, see Section A.8, "ADF Button Component Properties."

To insert an ADF Button component:

1. Open the integrated Excel workbook.
2. Select the cell in the Excel worksheet where you want to anchor the component.
3. In the components palette, select **ADF Button** and click **Insert Component**. Alternatively, in the **Oracle ADF** tab, select **ADF Button** from the **Insert Component** dropdown list.
4. Configure properties in the property inspector to determine the actions the component invokes at runtime in addition to the appearance, design, and layout of the component. Table 6–1 outlines some properties you must specify values for, and provides links to additional information.

Table 6–1 ADF Button component properties

For this property...	Specify...
`Label`	A string or an EL expression that resolves to a label at runtime to indicate the purpose of the ADF Button component. For example, the `Label` property for the **Advanced Search** button in the runtime `EditPriceList-DT.xlsx` workbook of the Master Price List module has the following value in design mode: `#{res['excel.advSearchButton.label']}` This EL expression references a string key in the `res` resource bundle. For more information about resource bundles, see Section 10.2, "Using Resource Bundles in an Integrated Excel Workbook." For more information about using labels in integrated Excel workbooks, see Section 9.4, "Using Labels in an Integrated Excel Workbook." If you want to include the ampersand (`&`) character in the label, you must use `&&`. A single `&` character acts as a special character and is not displayed in the label.
`ClickActionSet`	Specify one or more actions in the `Actions` array of the `ClickActionSet` that the end user invokes when he or she clicks the ADF Button component. For more information about action sets, see Section 8.2, "Using Action Sets."

5. Click **OK**.

Notes:

- If you change the view mode of the Excel worksheet to the Page Layout or Page Break mode, the ADF Button components may be rendered in an unexpected position. You must return back to Normal mode without saving the workbook, and then Run and stop the integrated Excel workbook to render the buttons to their original positions.
- You can modify the properties of the component at a later time by selecting the cell in the worksheet that anchors the component and then displaying the property inspector.
- The ADF Button components are active at 100% zoom only, and are disabled when the end user zooms in or out on an integrated Excel worksheet.

Tip: In design mode, you can click the button, or press the spacebar when the button is in focus, to open the property inspector. The right-click context menu is disabled for a button.

If you want to add navigation buttons in your integrated Excel workbook to navigate to previous or next record, see Section 6.9, "Using Navigation Buttons."

6.3 Inserting an ADF Label Component

The ADF Label component is a component that you can insert into the active worksheet to display a static string value. You specify a value in the input field for `Label` in the property inspector or alternatively you invoke the expression builder to write an EL expression that resolves to a string at runtime. The retrieved string can be defined in a resource bundle or in an attribute control hint for an entity or view object. For example, the following EL expression resolves to the value of a string key defined in a resource bundle at runtime:

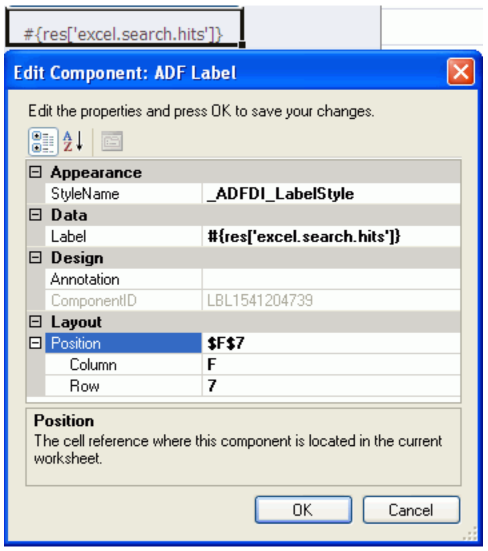
```
#{bindings.ProductList.label}
```

The value that you specify for the `Label` property in an ADF Label component or other Oracle ADF components is evaluated after the worksheet that hosts the Oracle ADF component is initialized (opened for the first time).

You can configure a number of properties for the component, such as style and position, in the worksheet using the property inspector.

Figure 6–2 shows an ADF Label component with its property inspector in the foreground. The ADF Label component references an EL expression that resolves to the value of a string key defined in the `res` resource bundle at runtime.

Figure 6–2 ADF Label Component



#{res['excel.search.hits']}

Edit the properties and press OK to save your changes.

Appearance	
StyleName	_ADFDI_LabelStyle
Data	
Label	#{res['excel.search.hits']}
Design	
Annotation	
ComponentID	LBL1541204739
Layout	
Position	F7
Column	F
Row	7

Position
The cell reference where this component is located in the current worksheet.

To insert an ADF Label component:

1. Open the integrated Excel workbook.
2. Select the cell in the Excel worksheet where you want to anchor the component.
3. In the components palette, select **ADF Label** and click **Insert Component**. Alternatively, in the **Oracle ADF** tab, select **ADF Label** from the **Insert Component** dropdown list
4. Configure properties in the property inspector to determine the appearance, design, and layout of the component.
5. Click **OK**.

Note: You can modify the properties of the component at a later time by selecting the cell in the worksheet that anchors the component and then displaying the property inspector. You can also right-click in the cell and choose **Edit Component** from the context menu to open the property inspector.

For more information about using labels in an integrated Excel workbook, see Section 9.4, "Using Labels in an Integrated Excel Workbook."

6.4 Inserting an ADF Input Text Component

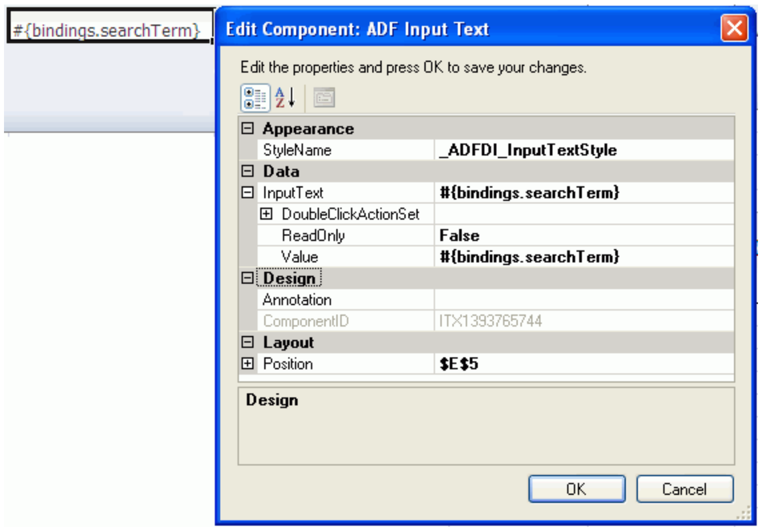
The ADF Input Text component is a component that you can insert into the active worksheet using the components palette. The active cell in the worksheet when you insert the component displays the current value from the component's binding after the worksheet `DownSync` action is invoked. End users can edit this value at runtime. You configure the worksheet `UpSync` action to transfer changes end users make to the value back to the Fusion web application and a `Commit` action binding to commit the changes in the Fusion web application.

You can configure a number of properties for the component, such as its position, style and behavior when a user double-clicks the cell (`DoubleClickActionSet` properties), in the worksheet using the property inspector. For more information about `DoubleClickActionSet`, see Section 8.2, "Using Action Sets."

The ADF Table component can invoke this component as a subcomponent when you set values for the ADF Table component column's `InsertComponent` or `UpdateComponent` properties. In this context, the ADF Input Text component enables the end user to input data into the ADF Table component. For more information, see Section 7.5, "Configuring an ADF Table Component to Insert Data."

Figure 6–3 shows an ADF Input Text component with its property inspector in the foreground. The ADF Input Text component binds to the `searchTerm` attribute binding in the Master Price List module of the Fusion Order Demo application. The end user enters a search term in this component and then uses an ADF Button component to invoke a search.

Figure 6–3 ADF Input Text Component



Edit the properties and press OK to save your changes.

Appearance	
StyleName	**_ADFDI_InputTextStyle**
Data	
InputText	**#{bindings.searchTerm}**
DoubleClickActionSet	
ReadOnly	**False**
Value	**#{bindings.searchTerm}**
Design	
Annotation	
ComponentID	ITX1393765744
Layout	
Position	**E5**

Design

To insert an ADF Input Text component:

1. Open the integrated Excel workbook.
2. Select the cell in the Excel worksheet where you want to anchor the component.
3. In the components palette, select **ADF Input Text** and click **Insert Component**. Alternatively, in the **Oracle ADF** tab, select **ADF InputText** from the **Insert Component** dropdown list

4. Configure properties in the property inspector to determine the appearance, layout, and behavior of the component. Table 6–2 outlines some properties that you must specify values for. For information about the component's other properties, see Section A.2, "ADF Input Text Component Properties."

Table 6–2 ADF Input Text component properties

For this property...	Specify...
`InputText.Value`	An EL expression for the `Value` property to determine what binding the component references.
`InputText.ReadOnly`	An EL expression that resolves to `False` so that changes the end user makes are uploaded. Write an EL expression that resolves to `True` if you want the component to ignore changes. `False` is the default value.

5. Click **OK**.

Note: You can modify the properties of the component at a later time by selecting the cell in the worksheet that anchors the component and then displaying the property inspector. You can also right-click in the cell and choose **Edit Component** from the context menu to open the property inspector.

6.5 Inserting an ADF Output Text Component

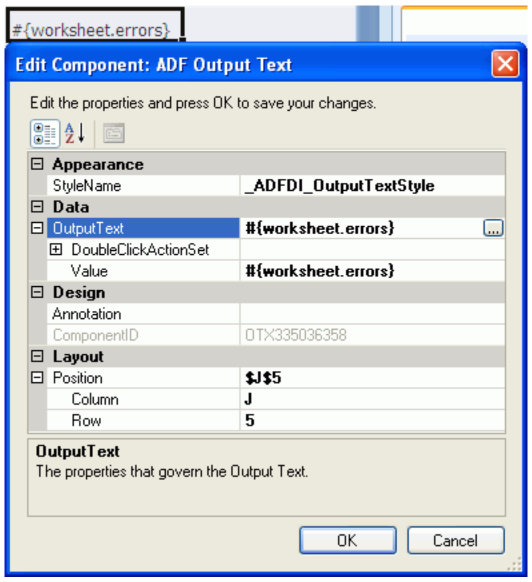
The ADF Output Text component is a component that you can insert into the active worksheet using the components palette. The active cell in the worksheet when you insert the component displays the current value from the component's binding after you invoke the worksheet `DownSync` action. The value the component displays is read-only. Changes that the end user makes to the value in the cell that anchors the component are ignored when changes are sent to the Fusion web application.

This component can also serve as a subcomponent for the ADF Table and ADF Read-only Table components. Columns in the ADF Table and ADF Read-only Table components can be configured to use the ADF Output Text component.

You can configure a number of properties for the component such as style, behavior when a user double-clicks the cell (`DoubleClickActionSet` properties), and position, in the worksheet using the property inspector.

Figure 6–4 shows an ADF Output Text component with its property inspector in the foreground. The ADF Output Text component references an ADF Table component in the Master Price List module of the Fusion Order Demo application. At runtime, the cell that anchors the ADF Output Text component displays any errors returned by the ADF Table component.

Figure 6–4 ADF Output Text Component


#{worksheet.errors}

Edit Component: ADF Output Text

Edit the properties and press OK to save your changes.

Appearance	
StyleName	_ADFDI_OutputTextStyle
Data	
OutputText	#{worksheet.errors}
DoubleClickActionSet	
Value	#{worksheet.errors}
Design	
Annotation	
ComponentID	OTX335036358
Layout	
Position	J5
Column	J
Row	5

OutputText
The properties that govern the Output Text.

To insert an ADF Output Text component:

1. Open the integrated Excel workbook.
2. Select the cell in the Excel worksheet where you want to anchor the component.
3. In the components palette, select **ADF Output Text,** and click **Insert Component**. Alternatively, in the **Oracle ADF** tab, select **ADF OutputText** from the **Insert Component** dropdown list
4. Configure properties in the property inspector to determine the appearance, layout, and behavior of the component.

   For example, you must write or specify an EL expression for the `Value` property to determine what binding the ADF Output Text component references. For more information about the values that you specify for the properties of the ADF Output Text component, see Section A.3, "ADF Output Text Component Properties."
5. Click **OK**.

Note: You can modify the properties of the component at a later time by selecting the cell in the worksheet that anchors the component and then displaying the property inspector. You can also right-click in the cell and choose **Edit Component** from the context menu to open the property inspector.

6.6 Inserting an ADF List of Values Component

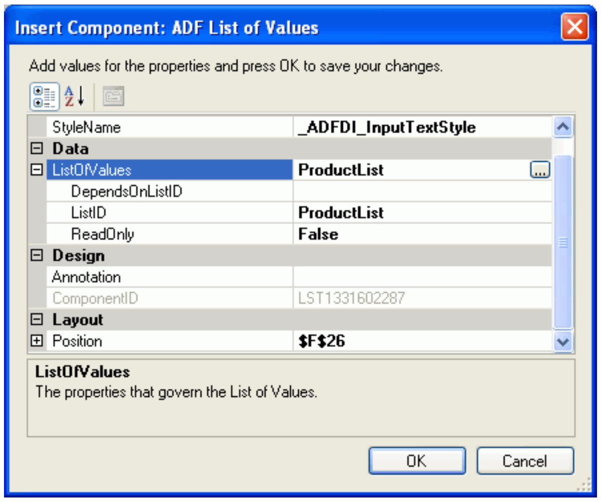
The ADF List of Values component is a component that displays a dropdown menu in the Excel worksheet cell at runtime. It displays a maximum of 250 values at runtime. You can insert the List of Values component into a cell in the Excel worksheet.

You must specify a value for the `ListID` property. The `ListID` property references the list binding which populates the dropdown menu with a list of values at runtime after you invoke the worksheet `DownSync` action.

Figure 6–5 shows an ADF List of Values component with its property inspector in the foreground. The ADF List of Values component references a list binding (`ProductList`) that populates a dropdown menu in the Excel worksheet at runtime.

Note: You can display a dropdown menu in an ADF Table component's column by selecting `TreeNodeList` or `ModelDrivenColumnComponent` as the subcomponent to create when you specify a value for the `TableColumn` array's `InsertComponent` property. For more information, see Section 7.13, "Creating a List of Values in an ADF Table Component Column."

Figure 6–5 ADF List of Values Component



Add values for the properties and press OK to save your changes.

StyleName	_ADFDI_InputTextStyle
⊟ Data	
⊟ ListOfValues	ProductList
DependsOnListID	
ListID	ProductList
ReadOnly	False
⊟ Design	
Annotation	
ComponentID	LST1331602287
⊟ Layout	
⊞ Position	F26

ListOfValues
The properties that govern the List of Values.

To insert an ADF List of Values component:

1. Open the integrated Excel workbook.
2. Select the cell in the Excel worksheet where you want to anchor the component.
3. In the components palette, select **ADF List of Values** and click **Insert Component**. Alternatively, in the **Oracle ADF** tab, select **ADF List of Values** from the **Insert Component** dropdown list
4. Invoke the binding ID picker by clicking the ellipsis button (**...**) beside the input field for the **ListID** property and select a list binding that the page definition file exposes.
5. Configure other properties in the property inspector to determine the appearance, design, and layout of the component. For information about ADF List of Values component properties, see Section A.5, "ADF List of Values Component Properties."
6. Click **OK**.

Note: You can modify the properties of the component at a later time by selecting the cell in the worksheet that anchors the component and then displaying the property inspector. You can also right-click in the cell and choose **Edit Component** from the context menu to open the property inspector.

6.7 Displaying Output from a Managed Bean in an ADF Component

You can configure an ADF component to display output from a managed bean in your Fusion web application. Information about how to use managed beans in a Fusion web application can be found in the "Using a Managed Bean in a Fusion Web Application" section of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. You reference a managed bean in an integrated Excel workbook through an EL expression. Add a method action binding to the page definition file you associate with the Excel worksheet to retrieve the value of the managed bean and assign it to an attribute binding. Use an EL expression to retrieve the value of the attribute binding at runtime.

6.7.1 How to Display Output from a Managed Bean

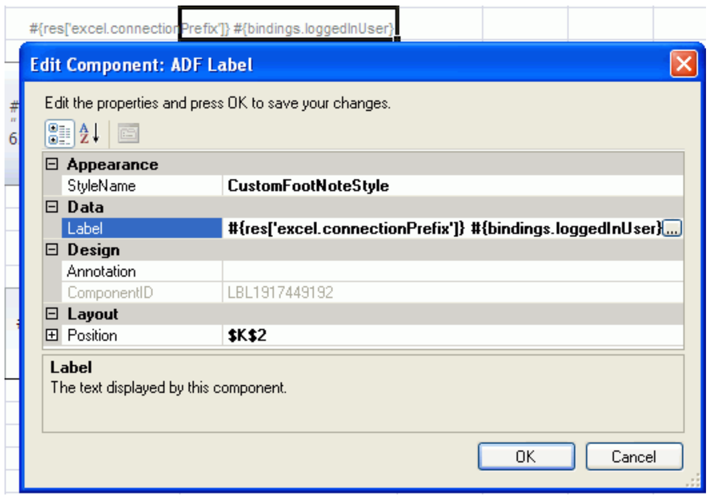
You write an EL expression for a property that supports EL expressions (for example, the `Label` property).

To display output from a managed bean:

1. Open the integrated Excel workbook.
2. Select the ADF component to display the output from the managed bean, and open its property inspector.

   Figure 6–6 shows an example from the `EditPriceList-DT.xlsx` workbook in the Master Price List module where an ADF Label component is configured to display the output from an attribute binding that has its value populated by an action binding.

Figure 6–6 ADF Label Component That Displays Output from a Managed Bean at Runtime



#{res['excel.connectionPrefix']} #{bindings.loggedInUser}

Edit Component: ADF Label

Edit the properties and press OK to save your changes.

Appearance	
StyleName	CustomFootNoteStyle
Data	
Label	#{res['excel.connectionPrefix']} #{bindings.loggedInUser}
Design	
Annotation	
ComponentID	LBL1917449192
Layout	
Position	K2

Label
The text displayed by this component.

3. Write an EL expression that gets the output from a managed bean at runtime.

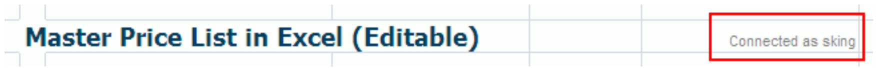
   The example in Figure 6–6 shows an EL expression that retrieves the value of a string key (`excel.connectionPrefix`) from the `res` resource bundle and the value of the `loggedInUser` attribute binding. This attribute binding references the output from the managed bean.
4. Click **OK**.

6.7.2 What Happens at Runtime When an ADF Component Displays Output from a Managed Bean

The method action binding retrieves values from the managed bean and populates the attribute binding. The EL expression that you write retrieves the value from the attribute binding and displays it to the end user through the ADF component that you configured to display output. For example, the ADF Label component shown in design mode in Figure 6–7 displays a string similar to the following at runtime:

```
Connected as sking
```

Figure 6–7 Output from a Managed Bean at Runtime



Master Price List in Excel (Editable)

Connected as sking

In Figure 6–7, `sking` is the user name of the user that is logged on to the Fusion web application through the integrated Excel workbook.

6.8 Displaying Concatenated or Calculated Data in Components

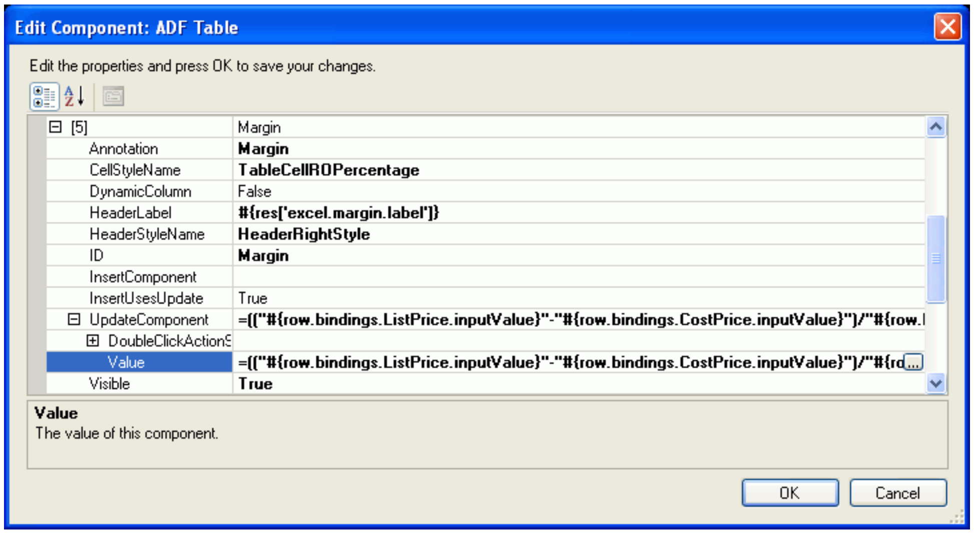
The ADF Desktop Integration module supports EL expressions within components that allow a single component to display data that is based on a calculation or concatenation of multiple binding expressions.

6.8.1 How to Configure a Component to Display Calculated Data

You write an EL expression for the `Value` property of an Input Text or Output Text component.

Figure 6–8 shows an EL expression example from the `EditPriceList-DT.xlsx` workbook in the Master Price List module where an ADF Output Text component of a column is configured to display the margin between the List Price and Cost Price columns.

Figure 6–8 ADF Output Text Component That Displays Calculated Data



Edit Component: ADF Table

Edit the properties and press OK to save your changes.

⊟ [5]	Margin
Annotation	**Margin**
CellStyleName	**TableCellROPercentage**
DynamicColumn	False
HeaderLabel	**#{res['excel.margin.label']}**
HeaderStyleName	**HeaderRightStyle**
ID	**Margin**
InsertComponent	
InsertUsesUpdate	True
⊟ UpdateComponent	=(("#{row.bindings.ListPrice.inputValue}"-"#{row.bindings.CostPrice.inputValue}")/"#{row.l
⊞ DoubleClickActionS	
Value	=(("#{row.bindings.ListPrice.inputValue}"-"#{row.bindings.CostPrice.inputValue}")/"#{ro
Visible	**True**

Value
The value of this component.

To create an EL expression to display calculated data

1. Open the integrated Excel workbook.
2. Select the ADF Input Text or ADF Output Text component to display calculated data.
3. Open the property inspector and click the ellipses button (...) of the `Value` property.
4. Write an EL expression that gets the output from two, or more, expressions.

   Example 6–1 shows an EL expression that calculates the difference between the values of List Price and Cost Price columns of an item, and then divides it with value of Cost Price column to generate a margin.

Example 6–1 An EL Expression for Calculated Data

```
=(("#{row.bindings.ListPrice.inputValue}"-"#{row.bindings.CostPrice.inputValue}")/
"#{row.bindings.CostPrice.inputValue}")
```

5. Click **OK**.

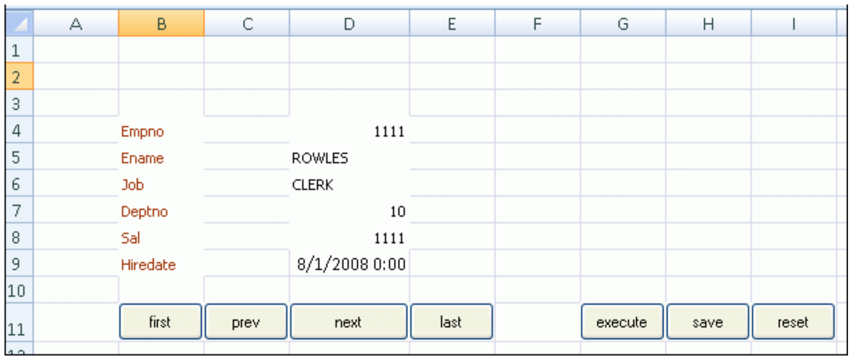
For more information about EL expressions, see Appendix B, "ADF Desktop Integration EL Expressions."

Note: If the `Value` property of an ADF Input Text component contains a formula, the ADF Input Text component becomes read-only at runtime regardless of the value of the `ReadOnly` property.

6.9 Using Navigation Buttons

You can create navigation buttons (Next, Previous, First, and Last) to navigate from one record to another as shown in Figure 6–9. If the end user changes a record's data before navigating to another record, you can choose to save those changes or ignore them.

Figure 6–9 Navigation Buttons in an Integrated Excel Workbook



Empno	1111
Ename	ROWLES
Job	CLERK
Deptno	10
Sal	1111
Hiredate	8/1/2008 0:00

To save changes before navigating to another record, define the action sets of the button in the following order:

1. `Worksheet.UpSync`
2. `Commit`
3. Navigation action (for example, Next)

4. `Worksheet.DownSync`

> **Note:** If you omit the `Commit` action from the action set, any pending changes to multiple records are lost when the end user's web application session ends.

To ignore changes before navigating to another record, define the action sets of the button in the following order:

1. Navigation action (for example, Next)
2. `Worksheet.DownSync`

> **Note:** If you define button actions to ignore changes, then it is the end user’s responsibility to save changes before navigating to another record.

7 Working with ADF Desktop Integration Table-Type Components

This chapter describes how you work with the table-type components that ADF Desktop Integration provides.

This chapter includes the following sections:

7.1 Introduction to ADF Desktop Integration Table-Type Components

ADF Desktop Integration provides the following table-type components to display structured data:

- ADF Table component
- ADF Read-only Table component

The ADF Table and ADF Read-only Table components provide end users the functionality to download rows of data from the Fusion web application. The ADF Table component provides additional functionality that allows end users to edit or delete the downloaded data, or to insert new rows of data. The ADF Table component's `Upload` action is used to upload the resulting data.

The number of rows that an ADF Table or ADF Read-only Table component contains expands or contracts based on the number of rows to download from a Fusion web application. You should not place anything to the left or right of a table-type component unless you want to replicate it when Excel inserts rows to accommodate the data that one of the table-type components downloads. You can place other components above or below a table-type component as they maintain their position relative to the table-type component at runtime. End users who want to insert new rows of data into an ADF Table component at runtime must insert full rows into the Excel worksheet that hosts the ADF Table component.

Each ADF Table component contains a **Key** column. Do not remove the **Key** column as it contains important information that is used by ADF Desktop Integration for proper functioning of the table. Removal of the **Key** column, or any modification in the **Key** column cell, results in errors and data corruption. For more information about the **Key** column, see Section 7.12, "Configuring ADF Table Component Key Column."

The other ADF Desktop Integration components that you can use with these table-type components are described in Chapter 6, "Working with ADF Desktop Integration Form-Type Components."

7.2 Page Definition Requirements for an ADF Table Component

The ADF Table component is one of the Oracle ADF components that ADF Desktop Integration exposes. It appears in the components palette of the ADF Desktop Integration Designer task pane and, after inserted into an Excel worksheet, allows the following operations:

- Read-only
- Insert-only
- Update-only
- Insert and update

Review the following sections for information about page definition file requirements specific to an ADF Table component.

Before you can configure an ADF Table component to provide data-entry functionality to your end users, you must configure the underlying page definition file for the Excel worksheet with ADF bindings. For general information about the page definition file requirements for an integrated Excel workbook, see Section 4.3, "Working with Page Definition Files for an Integrated Excel Workbook."

Expose the following control bindings when you create a page definition file for authoring an ADF Table component:

- Tree binding that exposes desired attribute bindings and a tree binding attribute that uniquely identifies each row in the table.
- Method action bindings and action bindings if you intend to configure values for the ADF Table component's `RowActions` and `BatchOptions` groups of properties. Examples of procedures where you set values for these groups of properties include:
  – Section 7.3, "Inserting an ADF Table Component into an Excel Worksheet"
  – Section 7.5, "Configuring an ADF Table Component to Insert Data"
  – Section 7.7, "Configuring a Worksheet to Download Pre-Insert Data to an ADF Table Component"

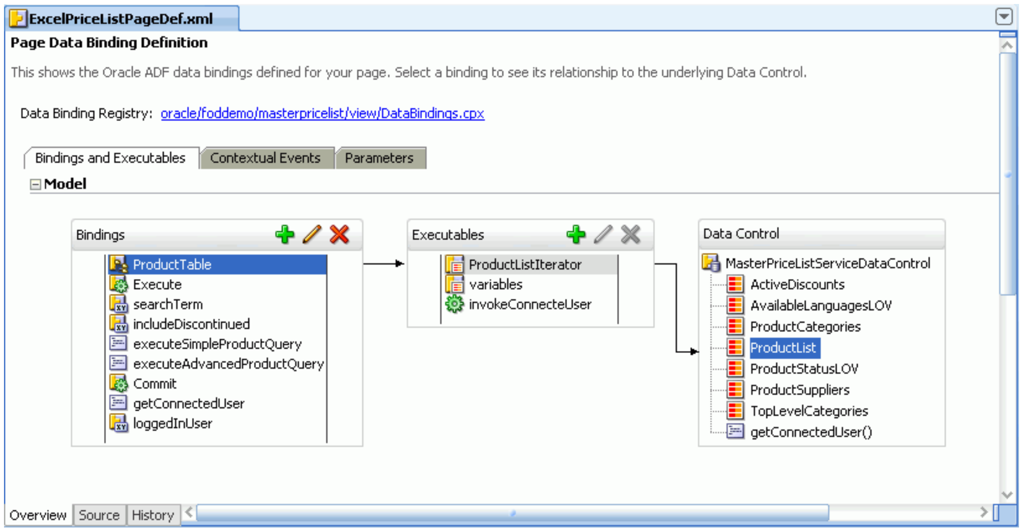
  Note: The previous list is not exhaustive.

- (Optional) Update record action binding.

  Note: Use descriptive names for the attributes of different iterators. Excel displays a flat list of bindings, so iterators are not displayed.

Figure 7–1 shows the bindings that the `ExcelPriceListPageDef.xml` page definition file includes. This page definition file can support the use of an ADF Table component in the Excel worksheet that it is associated with.

Figure 7–1 ADF Bindings Supporting Use of an ADF Table Component


ExcelPriceListPageDef.xml

Page Data Binding Definition

This shows the Oracle ADF data bindings defined for your page. Select a binding to see its relationship to the underlying Data Control.

Data Binding Registry: oracle/foddemo/masterpricelist/view/DataBindings.cpx

7.3 Inserting an ADF Table Component into an Excel Worksheet

After you have configured a page definition file correctly, you can insert the ADF Table component into the worksheet and configure its properties to achieve the functionality you want.

To insert an ADF Table component into an Excel worksheet:

1. Open the integrated Excel workbook.

2. Select the cell in the Excel worksheet where you want to insert the ADF Table component. When inserting an ADF Table component, you must ensure that the data of two tables do not overlap at runtime.
3. In the bindings palette of the ADF Desktop Integration Designer task pane, select the tree binding to use and click **Insert Binding**. Based on your selection, the Select Component dialog or the Insert Component dialog appears.
4. In the dialog that appears, select **ADF Table** and click **OK**.

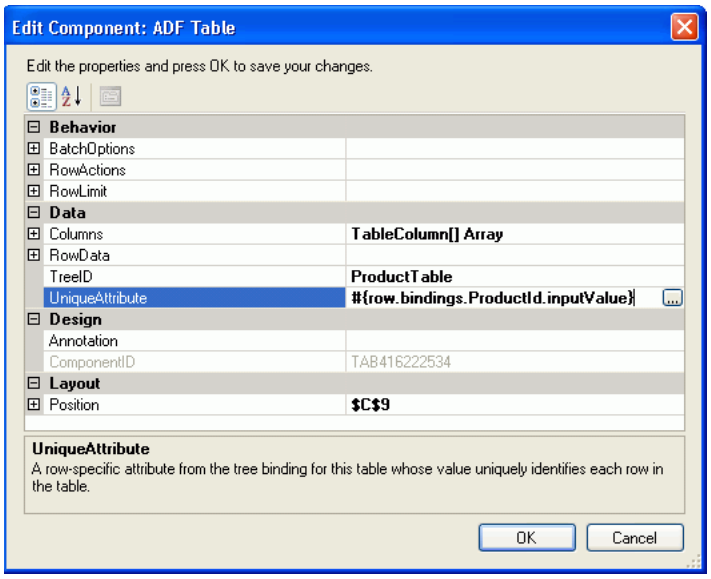
Notes:

- You can also insert an ADF Table component by using the components palette or the **Oracle ADF** tab. Select **ADF Table** and click **Insert Component**. Alternatively, in the **Oracle ADF** tab, select **ADF Table** from the **Insert Component** dropdown list. If you use either the components palette or the **Oracle ADF** tab to create the table component, you would have to add each column to appear in the component at runtime.
- When you insert an ADF Table component using Insert Binding, then by default, `InputText` is defined as the subcomponent type for all columns. If you want a column to have a list subcomponent (`TreeNodeList` or `ModelDrivenColumnComponent`), then delete the old column and reinsert it with your desired subcomponent type.

---

5. Configure properties for the ADF Table component using the property inspector shown in Figure 7–2.

Figure 7–2 ADF Table Property Inspector


Edit Component: ADF Table

Edit the properties and press OK to save your changes.

Behavior	
BatchOptions	
RowActions	
RowLimit	
Data	
Columns	TableColumn[] Array
RowData	
TreeID	ProductTable
UniqueAttribute	#{row.bindings.ProductId.inputValue}
Design	
Annotation	
ComponentID	TAB416222534
Layout	
Position	C9

UniqueAttribute
A row-specific attribute from the tree binding for this table whose value uniquely identifies each row in the table.

6. Specify a binding expression for the attribute that uniquely identifies each row in the iterator associated with the tree binding. The `UniqueAttribute` property may be left blank if the binding's iterator supports row keys.
7. Configure the `BatchOptions` properties of the ADF Table component as described in Table 7–1.

Table 7–1 BatchOptions Properties of the ADF Table Component

Set this property to...	This value...
`CommitBatchActionID`	The `Commit` action binding that the page definition file exposes.

8. Optionally, configure the `RowLimit` group of properties to determine what number of rows the ADF Table component can download.

   For more information, see Section 7.17, "Limiting the Number of Rows Your Table-Type Component Downloads."

9. Click **OK**.

   For more information about the properties that you can set for the ADF Table component, see Section A.9, "ADF Table Component Properties and Actions."

7.3.1 How to Add a Column in an ADF Table Component

After inserting an ADF Table component in the worksheet of your integrated Excel workbook, you may want to add a column that is not available in the tree binding. For example, you may want to add a column that displays values calculated by an Excel formula.

To add a column in an ADF Table component:

1. Open the integrated Excel workbook.
2. Select the cell in the Excel worksheet that references the ADF Table component and click the **Edit Component** button in the **Oracle ADF** tab. You can also right-click and select **Edit ADF Component Properties** from the context menu.
3. In the the Edit Component: ADF Table dialog, click ellipses button (...) of the `Columns` property to open the TableColumn Collection Editor dialog. The dialog lists all columns of the selected ADF Table component.
4. Click **Add** to add a new column. The new column is inserted at the end of the Members list. To move the column to a specific position, select the column and use **Up** and **Down** arrow keys.
5. Configure the new column’s properties in the right pane of the dialog. For information about ADF Table component properties, see Section A.9, "ADF Table Component Properties and Actions."
6. Click **OK**.

ADF Desktop Integration does not limit the number of columns you can add in an ADF Table component, you can add as many columns as your version of Excel supports. However, it has been observed that a very wide table gives slow performance and poor user experience. If you experience the same, try reducing the number of columns of the table before diagnosing other reasons for slow performance.

7.4 Configuring an ADF Table Component to Update Existing Data

When you add the ADF Table component, by default, it allows end users to edit the existing data, but it does not allow them to add new data rows or delete existing data rows.

7.4.1 How to Configure an ADF Table Component to Update Data

If you want the end user to be able to edit existing data, but would like to restrict addition or deletion of data rows, no additional configuration is required. Ensure that the ADF Table component `RowAction` properties are set, as described in Table 7–2.

Table 7–2 RowAction properties of ADF Table Component

Property	Value
`InsertRowEnabled`	`False`
`DeleteRowEnabled`	`False`
`UpdateRowEnabled`	`True`

7.4.2 What Happens at Runtime When an ADF Table Component Updates Data

When the end user changes data in a row, ADF Desktop Integration marks the row and an upward pointing triangle appears in a row of the `_ADF_ChangedColumn` column. After updating the existing data, the end user initiates upload process to save the changes. For more information about the ADF Table component's upload process, see Section 7.8, "Configuring an Oracle ADF Component to Upload Changes from an ADF Table Component."

Excel uploads modified rows from the integrated workbook in batches rather than row by row. You can configure the size of batches and the actions an ADF Table component invokes when it uploads a batch. For more information about batch processing, see Section 7.10, "Batch Processing in an ADF Table Component."

For more information about the properties that you can set for the ADF Table component, see Section A.9, "ADF Table Component Properties and Actions."

7.5 Configuring an ADF Table Component to Insert Data

The primary purpose of an ADF Table component is to provide end users with an interface where they can input or edit data which can then be uploaded to the database that serves your Fusion web application. For this to happen, you must expose methods on data controls, create action bindings in your page definition file, and set properties for the ADF Table component that an Excel worksheet hosts. Note that a full Excel row must be inserted for this functionality to work correctly.

7.5.1 How to Configure an ADF Table Component to Insert Data Using a View Object's Operations

If you want the changes that the end user makes in an ADF Table component to be committed invoking the ADF Table component's `Upload` action, you must configure some of the ADF Table component's properties.

To configure an ADF Table component to insert data using a view object's operations:

1. Open the project in JDeveloper.
2. If not present, add a `CreateInsert` and a `Commit` action binding to the page definition file that is associated with the Excel worksheet that hosts the ADF Table component.

For more information, see Section 4.3, "Working with Page Definition Files for an Integrated Excel Workbook" and Section 7.2, "Page Definition Requirements for an ADF Table Component."

3. Open the integrated Excel workbook.
4. Select the cell in the Excel worksheet that references the ADF Table component and click the **Edit Component** button in the **Oracle ADF** tab.
5. In the Edit Component: ADF Table dialog, configure the `RowActions` properties of the ADF Table component as described in the Table 7–3:

Table 7–3 RowActions properties of ADF Table component

Set this property to...	This value...
`InsertRowEnabled`	`True`
`InsertBeforeRowActionID`	The `CreateInsert` action binding that the page definition file exposes.
`InsertRowsAfterUploadEnabled`	`True`, to upload the inserted rows again regardless of whether they have been previously uploaded. By default, this property is set to `False`. The property is ignored if `InsertRowEnabled` is set to `False`.

6. Configure the `BatchOptions` properties of the ADF Table component as described in the following table:

Table 7–4 BatchOptions Properties of the ADF Table Component

Set this property to...	This value...
`CommitBatchActionID`	The `Commit` action binding that the page definition file exposes.

7. Configure the `Columns` property of the ADF Table component as described in the following table:

Table 7–5 Columns property of ADF Table component

Set this property to...	This value...
`InsertUsesUpdate`	`True`
`UpdateComponent`	■ Set the **Value** field of the `UpdateComponent` property to the update attribute from the page definition file. For example, `#{row.bindings.ProductId.inputValue}`. ■ Verify that `ReadOnly` property of `UpdateComponent` is set appropriately. This property only appears if you selected `InputText` or `TreeNodeList` as the subcomponent to associate with the column. Set `ReadOnly` to `False` if you do want users to edit the values in the column, set to `True` otherwise. For more information about the components that you can use as a subcomponent, see Chapter 6, "Working with ADF Desktop Integration Form-Type Components."
`ID`	Set a value in this field that uniquely identifies the column in the ADF Table component's list of columns. A value for this property is required. The ADF Table component generates an initial value that you need not modify.

Table 7–5 (Cont.) Columns property of ADF Table component

Set this property to...	This value...
`CellStyleName`	Set this property to a style defined in the workbook or to an EL expression that applies a style to the cells in the column at runtime. For more information about styles, see Chapter 9, "Configuring the Appearance of an Integrated Excel Workbook."
`HeaderLabel`	Set this property to a label or to an EL expression that evaluates to a label which is rendered in the column header at runtime. For more information about labels, see Section 9.4, "Using Labels in an Integrated Excel Workbook."
`HeaderStyleName`	Set this property to a style defined in the workbook or to an EL expression that applies a style to the column's header cell at runtime. For more information about styles, see Chapter 9, "Configuring the Appearance of an Integrated Excel Workbook."

8. Repeat Step 7 for each column that contains data to commit during invocation of the `Upload` action.

   For information about ADF Table component properties, see Section A.9, "ADF Table Component Properties and Actions."

> **Note:** If the `InsertRowsAfterUploadEnabled` property is set to `False` and the end user tries to upload the inserted rows again, an error message in the status column is displayed indicating that the row cannot be inserted more than once.

7.5.2 How to Insert a New Row in a Polymorphic View Object

If you are using a polymorphic view object and want to insert a new row, you should create a custom method to set the discriminator value to the newly inserted row. The default `CreateInsert` action binding does not support polymorphic view objects.

Example 7–1 shows the sample code of a custom method used to insert a new row.

Example 7–1 Insert a New Row in a Polymorphic View Object

```
public void createInsertWithDiscriminator()
{
    AttributeList attrs = new NameValuePairs();
    attrs.setAttribute("<Discriminator_Attribute>", <Discriminator_Value>);
    ViewRowImpl row = (ViewRowImpl)<View_Object>.createAndInitRow(attrs);
    <View_Object>.insertRow (row);
}
```

Before implementing the sample code of Example 7–1, replace `<Discriminator_Attribute>` with the attribute name, `<Discriminator_Value>` with the discriminator value, and `<View_Object>` with the view object name.

7.6 Configuring Oracle ADF Component to Download Data to an ADF Table Component

After you add an ADF Table component to a worksheet, you configure it and the worksheet that hosts it, so that the ADF Table component downloads data from the Fusion web application. To achieve this, you configure an Oracle ADF component, such as ADF Button, a worksheet ribbon button, or a worksheet event to invoke an action set. The action set that is invoked must include the ADF Table component `Download` action among the actions that it invokes.

7.6.1 How to Configure an Oracle ADF Component to Download Data to an ADF Table Component

Configure an Oracle ADF component, a worksheet ribbon button, or a worksheet event to invoke an action set that, in turn, invokes the ADF Table component `Download` action.

To configure an Oracle ADF component to download data to an ADF Table component:

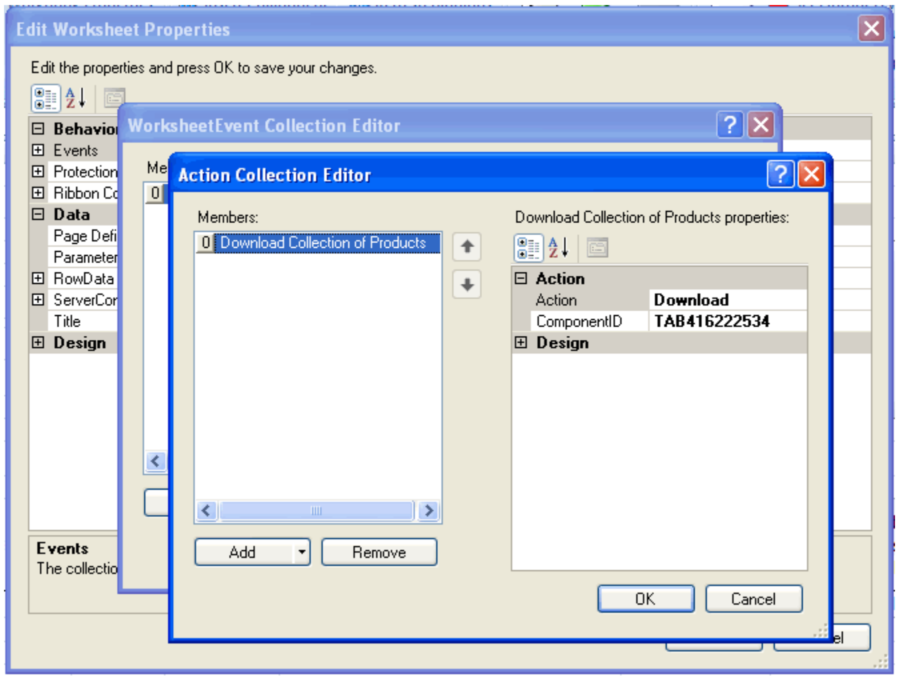
1. Open the integrated Excel workbook.
2. Open the Action Collection Editor to configure an action set for the worksheet event, worksheet ribbon button, or Oracle ADF component (a button, for example) that is going to invoke the action set at runtime.

   For more information about invoking action sets, see Section 8.2, "Using Action Sets."
3. Add the ADF Table component `Download` action to the list of actions that the action set invokes at runtime.

   The ADF Table component `Download` action downloads the current state of the binding referenced by the ADF Table component `TreeID` property. To ensure that the state of this binding is up to date before download, add a query action that refreshes the binding before the action set invokes the ADF Table component `Download` action.

   Figure 7–3 shows the Action Collection Editor in the `EditPriceList-DT.xlsx` workbook where the action set invoked by the worksheet event `Startup` is configured.

Figure 7–3 Action Set Downloading Data to an ADF Table Component


Edit Worksheet Properties
Edit the properties and press OK to save your changes.
Behavior
Events
Protection
Ribbon Co
Data
Page Defi
Parameter
RowData
ServerCor
Title
Design
Events
The collectio
WorksheetEvent Collection Editor
Action Collection Editor
Members:
Download Collection of Products properties:
Add
Remove
OK
Cancel

0 Download Collection of Products

Action	
Action	**Download**
ComponentID	**TAB416222534**
Design	

4. Click **OK**.

7.6.2 What Happens at Runtime When an ADF Table Component Downloads Data

The end user invokes the action set that you configured. The action set invokes the list of actions specified in order. These include an action that invokes the `Download` action of the ADF Table component. This action downloads the current state of the binding referenced by the ADF Table component `TreeID` property. If the tree binding referenced by the `TreeID` property contains data with a master-detail relationship (for example, a product category with multiple products), the ADF Table component shows the first record in the detail result set (for example, the first product). How you configured the tree binding in the Fusion web application determines which of the detail records is defined as the first record. For more information about using tree bindings to display master-detail data, see the "Using Trees to Display Master-Detail Objects" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

The number of rows that the action downloads depends on the values set for the `RowLimit` group of properties in the ADF Table component. For more information, see Section 7.17, "Limiting the Number of Rows Your Table-Type Component Downloads."

7.7 Configuring a Worksheet to Download Pre-Insert Data to an ADF Table Component

Pre-insert data is data contained in one or more rows of data that you configure an iterator in a Fusion web application to reference. These rows of data have not yet been committed to the Fusion web application's database. You can configure the iterator to populate values for some or all of its attributes.

At design time in the integrated Excel workbook, you can configure an ADF Table component and the worksheet that hosts it so that the ADF Table component downloads pre-insert data from the Fusion web application. To achieve this, you configure an Oracle ADF component, such as an ADF Button component, a worksheet ribbon button, or a worksheet event to invoke an action set. The action set that is invoked must include the ADF Table component `DownloadForInsert` action among the actions that it invokes.

The `DownloadForInsert` action differs from the `Download` action as follows:

- `DownloadForInsert` populates table cell data with the value of the EL expression for the insert component that is associated with each column in the ADF Table component. `Download` populates the table cell data with the EL expression for the update component that is associated with each column in the ADF Table component.
- The EL expression `#{components.`*`componentID`*`.currentRowMode}` returns `Insert` when evaluated by the `DownloadForInsert` action. In contrast, the same EL expression evaluated by the `Download` action returns `Update`. The *`componentID`* part of the EL expression references the ID of the ADF Table component.

Note the following points to invoke the `DownloadForInsert` action:

- Use the action with data rows that are in the `STATUS_INITIALIZED` state, as these data rows are ignored when the transaction is committed.

- An action set that includes the `DownloadForInsert` action does not execute this action if an ADF Table component's `RowActions.InsertRowEnabled` property is set to `False`.
- It serves no purpose to include both the `DownloadForInsert` and `Download` actions in the same action set, as the last executed action determines what data appears in the ADF Table component.

7.7.1 How to Configure a Worksheet to Download Pre-Insert Data to an ADF Table Component

Configure an Oracle ADF component, a worksheet ribbon button, or a worksheet event to invoke an action set that, in turn, invokes the ADF Table component `DownloadForInsert` action.

To configure a worksheet to download pre-insert data to an ADF Table component:

1. Open the integrated Excel workbook.
2. Open the Action Collection Editor to configure an action set for the worksheet event, worksheet ribbon button, or Oracle ADF component that is going to invoke the action set at runtime.

   For more information about invoking action sets, see Section 8.2, "Using Action Sets."
3. Add the ADF Table component `DownloadForInsert` action to the list of actions that the action set invokes at runtime.
4. Click **OK**.

7.7.2 What Happens at Runtime When an ADF Table Component Downloads Pre-Insert Data

The end user invokes the action set that you configured. The action set invokes the list of actions specified in order. These include an action that invokes the `DownloadForInsert` action of the ADF Table component. This action downloads pre-insert data from the Fusion web application and inserts it in rows of the ADF Table component in the Excel worksheet. The `InsertComponent` property is configured for the ADF Table component columns associated with the rows inserted to host the pre-insert data. End users can invoke the ADF Table component's `Upload` action to commit the pre-insert data to the Fusion web application's database.

7.8 Configuring an Oracle ADF Component to Upload Changes from an ADF Table Component

You configure the ADF Table component and the worksheet that hosts it so the end user can upload changes they make to data in the ADF Table component to the Fusion web application. To configure this functionality, you decide what user gesture or worksheet event invokes the action set that invokes the ADF Table component's `Upload` action.

To provide upload options to end users in a web page from the Fusion web application that differ from the default upload dialog, you must specify a `Dialog` action in the action set before the action that invokes the ADF Table Component's `Upload` action. For more information, see Section 7.8.5, "How to Create a Custom Upload Dialog."

Note: In a master-detail relationship, ADF Desktop Integration does not support editing of the `ViewLink` attribute, as it changes the selections in the child view object. To prevent any accidental editing, define the `ViewLink` attributes to be read-only, or use a model configuration that does not include a view link between master and detail.

7.8.1 How to Configure an Oracle ADF Component to Upload Data from an ADF Table Component

Configure an Oracle ADF component, a worksheet ribbon button, a component (a button, for example), or a worksheet event to invoke an action set that, in turn, invokes the ADF Table component `Upload` action.

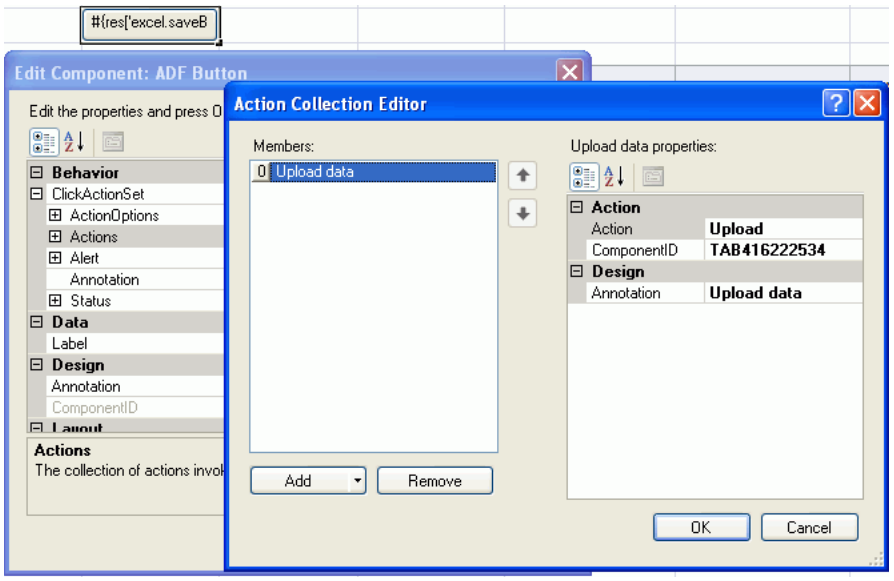
To configure an Oracle ADF component to upload changed data from an ADF Table component:

1. Open the integrated Excel workbook.
2. Open the Action Collection Editor to configure the action set that invokes the ADF Table component `Upload` action.

   For more information about action sets, see Section 8.2, "Using Action Sets."
3. Add the ADF Table component `Upload` action to the list of actions that the action set invokes at runtime.

   Figure 7–4 shows the Action Collection Editor in the `EditPriceList-DT.xlsx` workbook where the action set invoked by the ADF Button labeled **Save Changes** at runtime is configured.

Figure 7–4 Action Set Uploading Data from an ADF Table Component



Action	
Action	Upload
ComponentID	TAB416222534
Design	
Annotation	Upload data

4. Click **OK**.

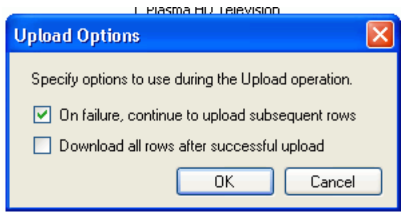
Note: The action set does not include a call to a commit-type action as the ADF Table component's batch options already include calls to `Commit`. For more information, see Section 7.10.1, "Configuring Batch Options for an ADF Table Component."

7.8.2 What Happens at Runtime When an ADF Table Component Uploads Data

At runtime, the end user invokes the action set through whatever mechanism you configured (ADF component, worksheet ribbon button, worksheet event). This triggers the following sequence of events:

1. If the ADF Table component contains dynamic columns, ADF Desktop Integration verifies whether the dynamic columns that were expanded the last time the ADF Table component's `Download` action was invoked are still present in the Fusion web application. If the columns are not present, ADF Desktop Integration prompts the end user to determine whether to continue upload process. If the end user decides not to continue, ADF Desktop Integration returns an abort code to the executing action set.
2. If the ADF Table component contains no pending changes to upload, the ADF Table component's `Upload` action returns a success code to the executing action set.
3. If you did not configure a custom upload dialog for the action set, as described in Section 7.8.5, "How to Create a Custom Upload Dialog," ADF Desktop Integration presents the default upload dialog shown in Figure 7–5.

   Figure 7–5 Default Upload Dialog

   
   Upload Options

   Specify options to use during the Upload operation.

   ☑ On failure, continue to upload subsequent rows

   ☐ Download all rows after successful upload

   OK Cancel

   If the end user clicks **Cancel**, ADF Desktop Integration returns an abort code to the executing action set. If the end user clicks **OK**, the action set continues executing with the options specified in the dialog for the upload operation.
4. The ADF Table component uploads modified rows in batches, rather than row by row. You can configure the batch options using the `BatchOptions` group of properties. For more information about batch options for the ADF Table component, see Section 7.10, "Batch Processing in an ADF Table Component."

   Each row of a batch is processed in the following way, and the process continues until all changed rows of each batch are processed:

   a. For inserted rows, invoke the `InsertBeforeRowActionID` action, if specified.

   b. Set attributes from the worksheet into the model, including any cached row attribute values.

   c. For edited rows, invoke the `UpdateRowActionID` action; and for inserted rows, invoke the `InsertAfterRowActionID` action, if specified.

   d. For each uploaded row, displays a status message in the Status column. For more information, see Section 8.2.5, "How to Display a Status Message While an Action Set Executes."

   e. For any row failure, it verifies the value of `AbortOnFail`. If `AbortOnFail` is set to `False`, it continues upload process, otherwise it stops uploading data and invokes the commit action.

5. While uploading data, the ADF Table component returns a success or failure code to the executing action set based on the following:

   - If the ADF Table component uploads all batches successfully, it returns the success status to the executing action set. If the end user has selected the **Download all rows after successful upload** option in Step 3, the ADF Table component then downloads all rows from the Fusion web application.
   - If the ADF Table component did not upload all batches successfully, the action set invokes the action specified by the `RowActions.FailureActionID` property, if an action is specified for this property. ADF Desktop Integration returns a failure code to the action set.

If you selected **On failure, continue to upload subsequent rows** in the Upload Options dialog of Step 3, the Upload action returns a success code to the action set even if some individual rows encountered validation failures.

Note: If an ADF Table component column's `ReadOnly` property evaluates to `True`, the ADF Table component's `Upload` action ignores changes in the column's cells.

For more information about an ADF Table component column's properties, see Table A–10.

7.8.3 What Happens at Runtime When a ReadOnly EL Expression is Evaluated During Upload

At runtime, if an ADF Table component column's `ReadOnly` property evaluates to `True`, the ADF Table component's `Upload` action ignores all changes in the column's cells.

It is recommended that you avoid `ReadOnly` EL Expressions that specifies row value binding expressions as part of the expression. If a row value binding must be used, you must understand how the EL expression is evaluated during `Upload`.

Currently, all EL expression evaluation is performed on the client. Therefore, an extra round trip to the server would be needed to first evaluate a `ReadOnly` EL expression containing a row value binding before the row value can be updated. In order to avoid the high cost of making an extra call to the server, `ReadOnly` EL expression evaluation during upload is performed the same as during table change tracking (as if the user were offline).

For more information about change tracking, see Section 7.19, "Tracking Changes in an ADF Table Component."

7.8.4 What Happens at Runtime When an Upload Fails

When the ADF Table component starts uploading data, ADF Desktop Integration creates a savepoint before initiating the upload process. In case of any failure, ADF Desktop Integration reverts back to the same savepoint, ensuring the integrity of the server-side state of the Fusion web application.

For each row that is uploaded, ADF Desktop Integration does the following:

1. Creates a `DataControlFrame` savepoint on the server.
2. Applies row attribute value changes, and performs data validation.
3. In case of any error, reverts back to the savepoint state.

For more information about savepoints, see the "Using Trees to Display Master-Detail Objects" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

7.8.5 How to Create a Custom Upload Dialog

You display a page from Fusion web application that offers end users different options to those presented in the default upload dialog. You add a `Dialog` action before the action that invokes the ADF Table component's `Upload` action in the action set.

To create a custom upload dialog:

1. Create a page in the JDeveloper project where you develop the Fusion web application. For information on how to create this page, see Section 8.4, "Displaying Web Pages from a Fusion Web Application."
2. In addition to the `ADFdi_CloseWindow` element (for example, a span element) described in Section 8.4, "Displaying Web Pages from a Fusion Web Application," the page that you create in Step 1 must include the elements described in Table 7–6.

Table 7–6 Span Elements Required for Custom Upload

Name	Description
`ADFdi_AbortUploadOnFailure`	If you set this element to `True`, the action set stops uploading if it encounters a failure. If the element references `False`, the action set attempts to upload all rows and indicates if each row succeeded or failed to upload.
`ADFdi_DownLoadAfterUpload`	Set this element to `True` so the action set downloads data from the Fusion web application to the ADF Table component after the action set uploads modified data.

Note: The page you create must include both elements to prevent ADF Desktop Integration presenting the default upload dialog to end users.

3. Add a `Dialog` action to invoke the page you created in Step 1 before the action in the action set that invokes the ADF Table component's `Upload` action.

   For more information about displaying pages from a Fusion web application, see Section 8.4, "Displaying Web Pages from a Fusion Web Application."

7.8.6 What Happens at Runtime When a Custom Upload Dialog Appears

When a custom dialog appears, the page from the Fusion web application that you configure the `Dialog` action in the action set to display appears instead of the default upload dialog.

Note: If there is no server connectivity when the end user tries to upload data, or if the end user is using the integrated Excel workbook in offline mode, the end user gets an error when the `Dialog` action fails to find the custom upload page. ADF Desktop Integration does not revert to the standard dialog when server connectivity is not available.

For more information about displaying a page from the Fusion web application, see Section 8.4, "Displaying Web Pages from a Fusion Web Application." Otherwise, the runtime behavior of the action set that you configure to upload data is as described in Section 7.8.2, "What Happens at Runtime When an ADF Table Component Uploads Data."

7.9 Configuring an ADF Table Component to Delete Rows in the Fusion Web Application

The ADF Table component exposes an action (`DeleteFlaggedRows`) that, when invoked, deletes the rows in the Fusion web application that correspond to the flagged rows in the ADF Table component. A *flagged row* in an ADF Table component is a row where the end user has double-clicked or typed a character in the cell of the `_ADF_FlagColumn` column as described in Section 7.10, "Batch Processing in an ADF Table Component." The `_ADF_FlagColumn` column must be present in the ADF Table component to configure it to delete rows in the Fusion web application.

In addition, the page definition file that you associate with the worksheet that hosts the ADF Table component must expose a `Delete` action binding.

7.9.1 How to Configure an ADF Table Component to Delete Rows in the Fusion Web Application

To delete rows from an ADF Table component, you must add the `Delete` action binding to the page definition file, configure `RowActions` group of properties of the ADF Table component, and configure an action set to invoke the `DeleteFlaggedRows` action.

To configure an ADF Table component to delete rows in a Fusion web application:

1. Open your Fusion web application in JDeveloper.
2. If not present, add a `Delete` action binding to the page definition file that is associated with the Excel worksheet that hosts the ADF Table component.

   For more information, see Section 4.3, "Working with Page Definition Files for an Integrated Excel Workbook."
3. Open the property inspector for the ADF Table component and set values for the `RowActions` group of properties as described in Table 7–7.

Table 7–7 RowActions properties of ADF Table component

Set this property...	To...
`DeleteRowActionID`	The `Delete` action binding that the page definition file exposes.

Table 7–7 (Cont.) RowActions properties of ADF Table component

Set this property...	To...
`DeleteRowEnabled`	`True` to enable the ADF Table component to delete rows in the Fusion web application. `False` is the default value.

For more information about ADF Table component properties, see Section A.9, "ADF Table Component Properties and Actions."

4. Click **OK**.
5. Open the integrated Excel workbook.
6. Open the Action Collection Editor to configure an action set for the Oracle ADF component, ribbon control, or worksheet event that the end user uses to invoke the action set at runtime.
7. Add the ADF Table component's `DeleteFlaggedRows` action to the list of actions that the action set invokes at runtime.

   For more information about invoking action sets, see Section 8.2, "Using Action Sets."
8. Click **OK**.

7.9.2 What Happens at Runtime When an ADF Table Component Deletes Rows in a Fusion Web Application

The end user flags rows to delete, as described in Section 7.10.2, "Row Flagging in an ADF Table Component." The end user then invokes the action set. The following sequence of events occurs:

1. If specified, the action binding referenced by the `BatchOptions.StartBatchActionID` property is invoked.

   Failures from this step are treated as errors. An error stops the action set invoking. It also returns the error condition to the action set. If an action binding is specified for the `ActionSet.FailureActionID` property, the action set invokes the specified action binding.

   For more information about configuring batch options, see Section 7.10, "Batch Processing in an ADF Table Component."
2. The action set invokes the `Delete` action binding specified by `RowActions.DeleteRowActionID`.

   Note: Rows inserted since the last invocation of the ADF Table component's `Download` action but not uploaded to the Fusion web application are ignored even if flagged for deletion.
3. If no errors occur during the invocation of the `Delete` action binding, a success message entry appears in the `_ADF_StatusColumn` column. If a failure occurs, the ADF Table component stops invocation of the `Delete` action binding and continues to Step 4.
4. If an action binding is specified for the `BatchOptions.CommitBatchActionID` property, the action set invokes it. If this step fails, the action set stops processing batches. If no failures occur, the action set processes the next batch by invoking the

action binding specified by the `BatchOptions.StartBatchActionID` property, and so on until the action set processes all batches.

5. If the action set processes all batches successfully, it invokes the action binding specified by its `ActionOptions.SuccessActionID` property if an action binding is specified for this property. It then removes the rows deleted in the Fusion web application by invocation of the `Delete` action binding specified by `RowActions.DeleteRowActionID` from the worksheet and returns a success code to the action set.

   If failures occur while the action set processes the batches, the action set invokes the action binding specified by its `ActionOptions.FailureActionID` property if an action binding is specified for this property. This action binding returns a failure code to the action set.

6. If an unexpected exception occurs while the action set invokes its actions, an error code is returned to the action set. All row level errors are displayed in the Status column, and all batch level errors can be tracked through `Table.errors`. For more information about error handling, see Section 12.4, "Error Reporting in an Integrated Excel Workbook."

7.10 Batch Processing in an ADF Table Component

The ADF Table component uploads modified rows from the Excel workbook in batches rather than row-by-row. You can configure batch option properties that determine the size of batches and what actions the ADF Table component invokes when it uploads a batch.

Note that end users might encounter unexpected reports of errors under certain circumstances while uploading data from ADF Table components. After posting changes from a batch, ADF Desktop Integration runs the action specified by the `CommitBatchActionID`. Errors that occur during the commit action might continue to be reported on subsequent batch commit actions, even though those batches of records do not contain the error. This happens when any pending model updates that exist when the `CommitBatchActionID` gets called are not automatically reverted when commit fails.

To avoid any such error, you must create a custom action for the `CommitBatchActionID` that first attempts to commit the pending model changes. However, if an exception occurs during commit, the custom method should first rollback the pending model changes, so that any subsequent batch commit attempts can succeed.

Note: It is important that the commit exception gets thrown again after rollback so that the commit errors are reported, as expected on the client.

7.10.1 Configuring Batch Options for an ADF Table Component

The ADF Table component has a group of properties (`BatchOptions`) that allow you to configure how the ADF Table component manages batches of rows. Information about these properties can be found in Section A.9, "ADF Table Component Properties and Actions."

To configure batch options for an ADF Table component:

1. Open the integrated Excel workbook.

2. Select the cell in the Excel worksheet that references the ADF Table component, and then click the **Edit Component** button in the **Oracle ADF** tab.
3. Set values for the `BatchOptions` group of properties in the property inspector that appears.

Table 7–8 RowData.BatchOptions Properties

Set this property...	To...
`BatchSize`	Specify how many rows to process before an ADF Table component action (`Upload` or `DeleteFlaggedRows`) invokes the action binding specified by `CommitBatchActionID`. Any value other than a positive integer results in all rows being processed in a single batch. The default value is 100 rows.
`CommitBatchActionID`	The action binding to invoke after the ADF Table component processes each batch. Typically, this is the `Commit` action binding.
`LimitBatchSize`	`True` When `True`, the ADF Table component processes rows in batches determined by the value of `BatchSize`. When `False`, the ADF Table component uploads all modified rows in a single batch. `True` is the default value.
`StartBatchActionID`	Specify the action binding to invoke at the beginning of each batch.

4. Click **OK**.

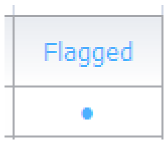
Note that a failure at the Entity level is not considered a batch failure. A failure at commit level (for example, a wrong value for a foreign key attribute) is considered as batch failure.

7.10.2 Row Flagging in an ADF Table Component

By default, the ADF Table component includes a column, `_ADF_FlagColumn`, that facilitates the selection of rows for flagged-row processing. Double-clicking a cell of the `_ADF_FlagColumn` column flags the corresponding row for processing by actions invoked by a component action.

When the end user double clicks a cell of the `_ADF_FlagColumn` column, a solid circle appears, or disappears, in the cell to indicate that the row is flagged, or not. Figure 7–6 shows an example of a flagged column.

Figure 7–6 Flagged Column in an ADF Table Component



Note: By default, the solid circle character indicates a row flagged for flagged-row processing. However, any nonempty cell in a `_ADF_FlagColumn` column flags the corresponding row for flagged-row processing.

The following component actions can be invoked on flagged rows:

- `DeleteFlaggedRows`
- `DownloadFlaggedRows`

You can use the `FlagAllRows` component action to flag all rows, and the `UnflagAllRows` component action to unflag all rows of the ADF Table component.

Use of these component actions is dependent on the appearance of the `_ADF_FlagColumn` column in the ADF Table component. If you remove the `_ADF_FlagColumn` column from the ADF Table component, you cannot invoke any of these component actions. For more information about these component actions, see Section A.9.3, "ADF Table Component Actions."

At runtime, the end user can invoke any of the previously listed component actions from an action set. The invoked component action processes all flagged rows. For example, it downloads or deletes all flagged rows. For more information about configuring an action set to invoke a component action, see Section 8.2.2, "How to Invoke Component Actions in an Action Set."

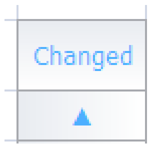
7.11 Special Columns in the ADF Table Component

By default, the ADF Table component includes some columns when you insert an ADF Table component in a worksheet. You can retain or remove these columns, if required. The following list describes the columns and the purpose they serve:

- `_ADF_ChangedColumn`

  The cells in this column track changes to the rows in the ADF Table component. If a change has been made to data in a row of the ADF Table component since download or the last successful upload, a character that resembles an upward pointing arrow appears in the corresponding cell of the `_ADF_ChangedColumn` column. This character toggles (appears or disappears) when a user double-clicks a cell in this column. Figure 7–7 shows an example.

  Figure 7–7 Changed Column in an ADF Table Component

  > **Note:** If the end user does not want the ADF Table component's `Upload` action to upload changes in the rows flagged by this column, he or she must clear the entry that appears in the corresponding cell.

  A confirmation dialog appears to end users when the ADF Table component's `Download` action is invoked, and one or more rows in this column are flagged as changed. The end user clicks **OK** to allow the `Download` action to execute, or **Cancel** to stop the execution of the `Download` action.

- `_ADF_FlagColumn`

  When the end user double-clicks a cell in this column, the corresponding row is flagged for flagged-row processing. A solid circle character appears to indicate that the row is flagged for flagged-row processing. For more information about the use of this column, see Section 7.10.2, "Row Flagging in an ADF Table Component."

A confirmation dialog appears to end users when the ADF Table component's `DownloadFlaggedRows` action is invoked, and one or more rows in `_ADFChangedColumn` and `_ADF_FlagColumn` are flagged. The end user clicks **OK** to allow the action to execute or **Cancel** to stop the execution of the action.

Note: By default, the solid circle character indicates a row flagged for flagged-row processing. However, any nonempty cell in a `_ADF_FlagColumn` flags the corresponding row for flagged-row processing.

- `_ADF_StatusColumn`

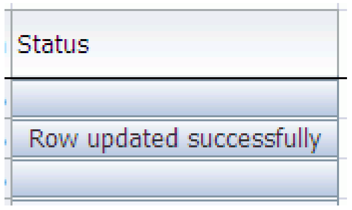
  This column reports the results of invocation of the following ADF Table component actions:

  – `DeleteFlaggedRows`

  – `Upload`

  A message appears in the cell of the `_ADF_StatusColumn` to indicate the result of the invocation for the corresponding row. If the end user invokes a `DoubleClickActionSet` defined in an ADF Table column and an error occurs, the errors are also reported in the status column of the corresponding row. Figure 7–8 shows an example of Status column message.

Figure 7–8 Status Column in an ADF Table Component


Status

Row updated successfully

- `_ADF_RowKeyColumn`

  This column, also referred as the **Key** column, contains important information about the ADF Table component that is used by ADF Desktop Integration at runtime. The column appears both at runtime and design time. You can remove the column from the table at design time, but note that it automatically appears at runtime.

  For more information about the `_ADF_RowKeyColumn` see Section 7.12, "Configuring ADF Table Component Key Column."

The ADF Table component treats the properties of the `_ADF_ChangedColumn`, `_ADF_FlagColumn`, and `_ADF_StatusColumn` columns differently to the properties of other columns that it references. It ignores the values set for properties such as `InsertComponent`, `InsertUsesUpdate`, and `UpdateComponent` unless it invokes the `DisplayRowErrors` action described in Table A–11. It reads the values for properties related to style and appearance, for example `CellStyleName` and `HeaderStyleName`.

7.12 Configuring ADF Table Component Key Column

When you add ADF Table to your integrated Excel workbook, the **Key** column (column ID: `_ADF_RowKeyColumn`) appears automatically at design time. The **Key** column contains important information that is used by ADF Desktop Integration for proper functioning of the table. Note that you must not remove the **Key** column at runtime.

7.12.1 How to Configure Key Column

You can configure the **Key** column's position, style properties, and the header label. By default, the `_ADFDI_TableKeyCellStyle` style is applied to it.

To configure Key column:

1. Open the integrated Excel workbook.
2. Select the cell in the Excel worksheet that references the ADF Table component and click the **Edit Properties** button in the **Oracle ADF** tab.
3. In the Edit Component: ADF Table dialog, click the ellipsis button (...) beside the input field for **Columns** to invoke the TableColumn Collection Editor.
4. Select the column with `ID` as `_ADF_RowKeyColumn`.
5. Change the column properties as desired, but do not change the following properties:
   - `DynamicColumn`
   - `InsertComponent`
   - `InsertUsesUpdate`
   - `UpdateComponent`
   - `ID`
   - `Visible`
6. If desired, change the position of the column using **Up** and **Down** arrow keys.
7. Click **OK** to close TableColumn Collection Editor.
8. Click **OK** to close the Edit Component: ADF Table dialog.

7.12.2 How to Manually Add Key Column At Design Time

If you are using the integrated Excel workbook prepared and configured using an old version of ADF Desktop Integration, the **Key** column would not be available at design time. It would only appear at runtime. If you want to configure the Key column properties, you can add it in workbook at desgn time.

To manually add key column at design time:

1. Open the integrated Excel workbook.
2. Select the cell in the Excel worksheet that references the ADF Table component, and then click the **Edit Properties** button in the **Oracle ADF** tab.
3. Add a new column in the ADF Table, and specify the properties as described in Table 7–9. For more information about adding a column, see Section 7.3.1, "How to Add a Column in an ADF Table Component.".

Table 7–9 Key Column Properties

Set this property...	To ...
`CellStyleName`	`_ADFDI_TableKeyCellStyle`
`HeaderStyleName`	`_ADFDI_HeaderStyle`
`DynamicColumn`	`False`
`HeaderLabel`	`#{_ADFDIres[COMPONENTS_TABLE_ROWKEY_COL_LABEL]}`
`ID`	`_ADF_RowKeyColumn`
`InsertUsesUpdate`	`True`
`UpdateComponent`	`OutputText` The **Value** property must be empty.
`Visible`	`True`

If desired, you may change the position of the Key column using **Up** and **Down** arrow keys.

4. Click **OK**.

Note: You must specify the `ID` property of the new column as `_ADF_RowKeyColumn`, otherwise the column will not be considered as a **Key** column, and another **Key** column would automatically appear at runtime.

7.13 Creating a List of Values in an ADF Table Component Column

Use the TreeNodeList subcomponent when you want to render a dropdown list of values in an ADF Table component column. The list of values can display a maximum of two hundred and fifty values at runtime. Unlike other ADF Desktop Integration components, the TreeNodeList subcomponent does not appear in the components palette described in Section 5.5, "Using the Components Palette." Instead, you invoke it as a subcomponent when you specify values for the `InsertComponent` or `UpdateComponent` properties of an ADF Table component column. For information about the properties of an ADF Table component column, see Section A.9.2, "ADF Table Component Column Properties."

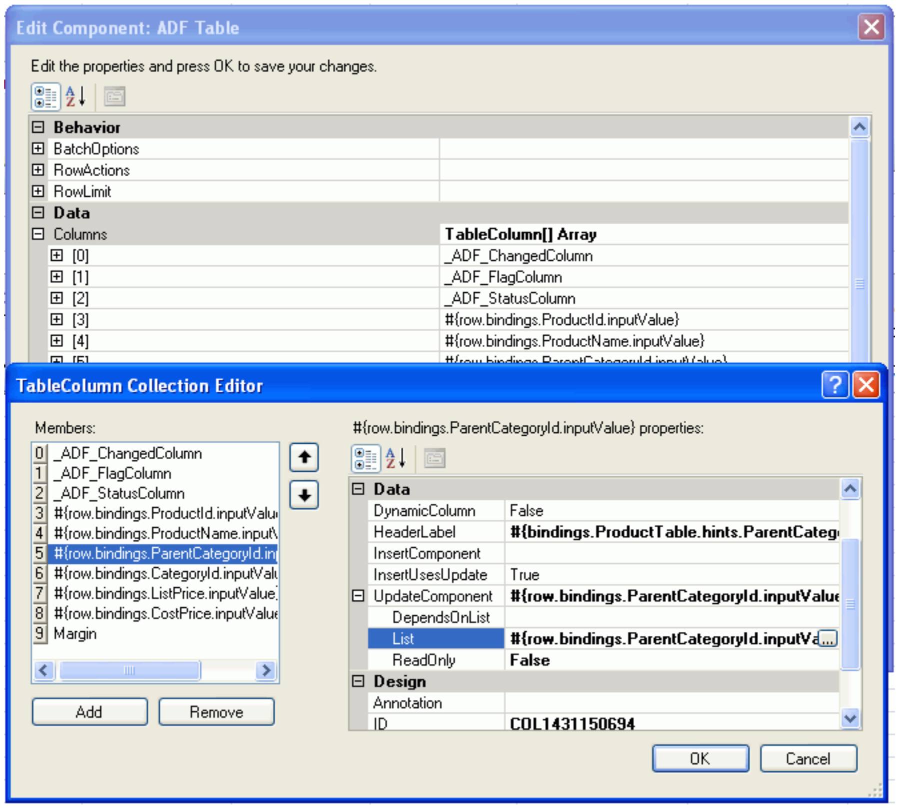
After you invoke the TreeNodeList subcomponent, you must specify a tree binding attribute associated with a model-driven list as a value for the TreeNodeList subcomponent's `List` property. The tree binding attribute associated with a model-driven list populates the dropdown menu in the Table component's column with a list of values after invocation of the Table component's `Download` action.

Note: You can create a model-driven list of values in your ADF Table component by choosing `ModelDrivenColumnComponent` as the subcomponent type. For more information about creating a model-driven list, see Section 7.14, "Adding a ModelDrivenColumnComponent Subcomponent to Your ADF Table Component."

For information about the properties of a TreeNodeList subcomponent, see Section A.6, "TreeNodeList Subcomponent Properties."

Figure 7–9 shows the property inspector for an ADF Table component column in `AdvEditPriceList-DT.xlsx` after `TreeNodeList` is selected as the subcomponent for the column’s `UpdateComponent` property.

Figure 7–9 ADF Table Component Column Configured to Display a List of Values



Edit the properties and press OK to save your changes.

Behavior	
BatchOptions	
RowActions	
RowLimit	
Data	
Columns	**TableColumn[] Array**
[0]	_ADF_ChangedColumn
[1]	_ADF_FlagColumn
[2]	_ADF_StatusColumn
[3]	#{row.bindings.ProductId.inputValue}
[4]	#{row.bindings.ProductName.inputValue}

Members:

0 _ADF_ChangedColumn
1 _ADF_FlagColumn
2 _ADF_StatusColumn
3 #{row.bindings.ProductId.inputValu
4 #{row.bindings.ProductName.input\
5 #{row.bindings.ParentCategoryId.in
6 #{row.bindings.CategoryId.inputVal
7 #{row.bindings.ListPrice.inputValue
8 #{row.bindings.CostPrice.inputValue
9 Margin

#{row.bindings.ParentCategoryId.inputValue} properties:

Data	
DynamicColumn	False
HeaderLabel	**#{bindings.ProductTable.hints.ParentCateg**
InsertComponent	
InsertUsesUpdate	True
UpdateComponent	**#{row.bindings.ParentCategoryId.inputValue**
DependsOnList	
List	**#{row.bindings.ParentCategoryId.inputVa**
ReadOnly	**False**
Design	
Annotation	
ID	**COL1431150694**

7.13.1 How to Create a List of Values in an ADF Table Component Column

You add a column to the ADF Table component column and select `TreeNodeList` as the subcomponent. You then specify a tree binding attribute as the value for the `TreeNodeList` subcomponent’s `List` property. A model-driven list must be associated with the tree binding attribute that you specify.

To create a list of values in an ADF Table component column:

1. Open the integrated Excel workbook.
2. Select the cell in the Excel worksheet that references the ADF Table component and click the **Edit Component** button in the **Oracle ADF** tab.
3. In the Edit Component: ADF Table dialog, click the ellipsis button (...) beside the input field for **Columns** to invoke the TableColumn Collection Editor.
4. Click **Add** to add a new column.
5. Choose the appropriate option for the newly created column:
   - Click the ellipsis button (**...**) beside the input field for **InsertComponent** to configure the runtime list of values for insert operations.
   - Click the ellipsis button (**...**) beside the input field for **UpdateComponent** to configure the runtime list of values for update and download operations.

In both options, the Select subcomponent to create dialog appears.

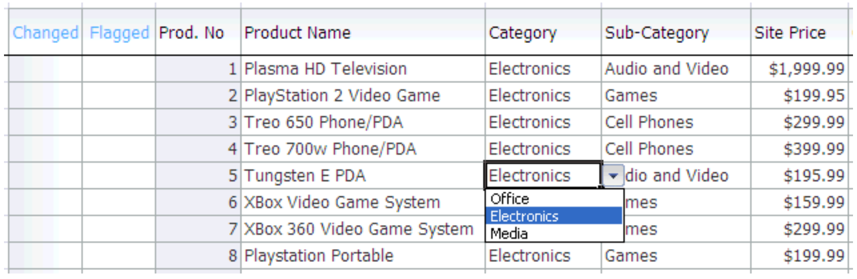
6. Select **TreeNodeList** and click **OK**.
7. Expand the property that you selected in Step 5 and configure values as follows:
   - Select a tree binding attribute associated with a model-driven list for the `List` property.
   - Select a value for `DependsOnList` only if you intend to create a dependent list of values as described in Section 8.8, "Creating Dependent Lists of Values in an Integrated Excel Workbook." The tree binding attribute or list binding you select for `DependsOnList` serves as the parent list of values in a dependent list of values.
   - Configure the `ReadOnly` property as desired.

     For information about these properties, see Section A.6, "TreeNodeList Subcomponent Properties."
8. Click **OK**.

7.13.2 What Happens at Runtime When a Column Renders a List of Values

At runtime, the ADF Table component invokes the `Download` action and populates each column. This action also populates the list of values in the column that you configure to render a list of values. Figure 7–10 shows an example from `AdvEditPriceList-DT.xlsx` of the Master Price List module where **Category** is the column configured to display a list of values.

Figure 7–10 Runtime View of an ADF Table Component Column Displaying a List of Values



Changed	Flagged	Prod. No	Product Name	Category	Sub-Category	Site Price
		1	Plasma HD Television	Electronics	Audio and Video	$1,999.99
		2	PlayStation 2 Video Game	Electronics	Games	$199.95
		3	Treo 650 Phone/PDA	Electronics	Cell Phones	$299.99
		4	Treo 700w Phone/PDA	Electronics	Cell Phones	$399.99
		5	Tungsten E PDA	Electronics	dio and Video	$195.99
		6	XBox Video Game System		mes	$159.99
		7	XBox 360 Video Game System		mes	$299.99
		8	Playstation Portable	Electronics	Games	$199.99

7.14 Adding a ModelDrivenColumnComponent Subcomponent to Your ADF Table Component

You can add a ModelDrivenColumnComponent subcomponent to an ADF Table component. The value of ModelDrivenColumnComponent is determined by the Control Type hint specified for each attribute on the server.

At design time, for a column, specify the subcomponent type as `ModelDrivenColumnComponent` for the `UpdateComponent` or `InsertComponent` properties. At runtime, if there is a model-driven list associated with the attribute, then the column uses a dropdown list using the TreeNodeList subcomponent. If there is no model-driven list associated with the attribute, or if any non-list-based control type is specified, then the column uses an InputText subcomponent.

For more information about creating a model-driven list, see the "How to Create a Model-Driven List" section of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Support for Dependent List of Values

When multiple ModelDrivenColumnComponent list subcomponents are exposed in an ADF Table component, then for each list ADF Desktop Integration determines whether it depends on another model-driven list. It verifies that the bind variable specified for a list references an attribute bound to another list.

If the list depends on another model-driven list, the subcomponent's `DependsOnList` value is set automatically at runtime.

As server-side list binding dependencies are determined only for lists in the same tree node, the following tree node list bindings are not supported:

- A binding that depends on a list binding in a different tree or tree node
- A binding that depends on a list binding in the page definition file

7.15 Adding a Dynamic Column to Your ADF Table Component

You can add dynamic columns to an ADF Table component so that the ADF Table component expands or contracts at runtime depending on the available attributes returned by the view object. The `DynamicColumn` property of the `Columns` group in the `TableColumn` array controls this behavior. To make a column dynamic, set the `DynamicColumn` property to `True`. A dynamic column in the `TableColumn` array is a column that is bound to a tree binding or a tree node binding whose attribute names are not known at design time. A dynamic column can expand to more than a single worksheet column at runtime.

The ADF Table component's dynamic column supports the following subcomponent types:

- InputText
- OutputText
- ModelDrivenColumnComponent

Note: ADF Desktop Integration does not support the subcomponent type TreeNodeList in a dynamic column.

Support for Model-Driven List of Values

You can also configure a dynamic column to support the List of Values subcomponent where the subcomponent type is determined from model metadata at runtime. At design time, specify the subcomponent type as `ModelDrivenColumnComponent` for the `UpdateComponent` or `InsertComponent` properties. At runtime, during dynamic column expansion, the model-driven runtime component is determined before caching the list of values. The remote servlet allows the client to retrieve Model metadata, allowing the client to choose the desired column subcomponent type. For more information about ModelDrivenColumnComponent, see Section 7.14, "Adding a ModelDrivenColumnComponent Subcomponent to Your ADF Table Component."

7.15.1 How to Configure a Dynamic Column

You configure a dynamic column by specifying an EL expression with the following format for the `Value` property of the component specified by the ADF Table component column's `InsertComponent` property as a subcomponent:

```
#{bindings.TreeID.[TreeNodeID].AttributeNamePrefix*.inputValue}
```

or:

```
#{bindings.TreeID.AttributeNamePrefix*.inputValue}
```

where:

- `TreeID` is the ID of the tree binding used by the ADF Table component
- `TreeNodeID` is an optional value that specifies the tree node binding ID. If you omit this value, all matching attributes from the tree binding display regardless of which tree node binding the attribute belongs to.
- `AttributeNamePrefix` identifies a subset of attributes that exist within the tree binding's underlying iterator. If you do not specify a value for `AttributeNamePrefix`, all attributes for the tree binding or tree binding node are returned. Always use the `*` character.

> **Note:** While adding a dynamic column, ensure that tree node attribute names are not specified in the page definition file. At runtime, the tree node object returns all attribute names from the underlying iterator. If there are attribute names specified in the page definition file, the tree node object limits the list of available attribute names based on that list.

The following example returns all attributes that begin with the name "`period`" in the `model.EmpView` node of the `EmpTree` binding:

```
#{bindings.EmpTree.[model.EmpView].period*.inputValue}
```

7.15.2 What Happens at Runtime When Data Is Downloaded or Uploaded

When the ADF Table component's `Download` or `DownloadForInsert` action is invoked, the ADF Table component automatically updates the dynamic columns so that they contain an up-to-date set of matching attributes. For each invocation of `Download`, ADF Desktop Integration requires that all rows must have the same set of attributes for the dynamic column. It may generate errors if the set of attributes changes from row to row during `Download`.

If a dynamic column supports both `Insert` and `Update` operations, you should specify the same EL expression for the `Value` properties of the dynamic column's `InsertComponent` and `UpdateComponent` subcomponents. At runtime, the ADF Table component expands to include a dynamic column that displays the value of the attribute binding returned by the EL expression.

When the ADF Table component's `Upload` action is invoked, the workbook prompts the end user to determine if the end user wants to continue to upload data when the previously downloaded attributes no longer exist in the tree binding.

Support for View Objects with Declarative SQL Mode

To support view objects that are configured with declarative SQL mode and customized at runtime, ADF Desktop Integration ignores all attributes with the `selected` property set to `False`. On the server side, the `JUCtrlHierNodeBinding` object determines the attribute list and passes it to the integrated Excel workbook on request.

7.15.3 How to Specify Header Labels for Dynamic Columns

Use the following syntax to write EL expressions for the `HeaderLabel` property of a dynamic column:

```
#{bindings.TreeID.[TreeNodeID].hints.AttributeNamePrefix*.label}
```

or:

```
#{bindings.TreeID.hints.AttributeNamePrefix*.label}
```

Specify the same tree binding ID, tree node binding ID, and attribute name prefix values in the `HeaderLabel` property of the dynamic column as the values you specify for the `Value` properties of the dynamic column's `InsertComponent` and `UpdateComponent` if the dynamic column supports `Insert` and `Update` operations.

> **Note:** The ADF Table component ignores the value of a column's `Visible` property when you configure a column to be dynamic. For more information about ADF Table component column properties, see Table A–10.

If you want the mandatory columns, where the end user must enter a value, to be marked with a character or a string, you must configure the `HeaderLabel` property. Use the following syntax to write EL expression to add a character or string to all mandatory columns:

```
=IF(#{bindings.TreeID.[TreeNodeID].hints.*.mandatory}, "<prefix_
for_mandatory_cols>", "") &
"#{bindings.TreeID.[TreeNodeID].hints.*.label}"
```

For example, the following EL expression adds an asterisk (*) character to the mandatory columns label:

```
=IF(#{bindings.MyTree.[myapp.model.MyChildNode].hints.*.mandator
y}, "* ", "") &
"#{bindings.MyTree.[myapp.model.MyChildNode].hints.*.label}"
```

7.15.4 How to Specify Styles for Dynamic Columns According to Attribute Data Type

You can specify different styles for each data type according to the data type of the column. Use the following syntax to write EL expressions for the `CellStyleName` property of a dynamic column:

```
=IF("#{bindings.TreeID.[TreeNodeID].hints.*.dataType}"="<data_
type>", <custom_style_expression1>, <custom_style_expression2>)
```

In the following example, the `MyDateStyle` style is applied to all `date` columns, and `MyDefaultStyle` is applied to other data type columns:

```
=IF("#{bindings.MyTree.[myapp.model.MyChildNode].hints.*.dataTyp
e}"="date", "MyDateStyle", "MyDefaultStyle")
```

The following example shows another scenario where the `MyDateStyle` style is applied to all `date` data type columns, `MyNumberStyle` is applied to all `number` data type columns, and `MyDefaultStyle` is applied to other data type columns:

```
=IF("#{bindings.MyTree.[myapp.model.MyChildNode].hints.*.dataTyp
e}"="date", "MyDateStyle",
IF("#{bindings.MyTree.[myapp.model.MyChildNode].hints.*.dataType
}"="number", "MyNumberStyle", "MyDefaultStyle"))
```

For more information about EL expressions, see Appendix B, "ADF Desktop Integration EL Expressions."

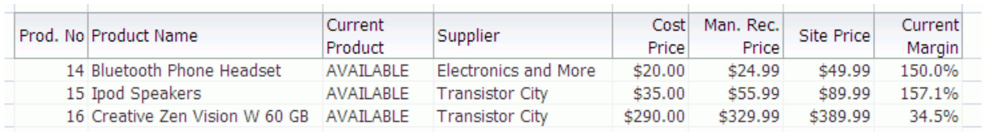
7.16 Creating an ADF Read-Only Table Component

At runtime, the ADF Read-only Table component renders a table across a continuous range of cells that displays data from the tree binding that the ADF Read-only Table component references. Use this component to display data that you do not want the end user to edit.

This component supports several properties, such as `RowLimit`, that determine how many rows the component downloads when it invokes its `Download` action. It also includes a group of properties (`Columns`) that determine what columns from the tree binding appear at runtime in the Excel worksheet. The `TreeID` property specifies the tree binding that the component references. More information about these properties and others that the ADF Read-only Table component supports can be found in Section A.10, "ADF Read-only Table Component Properties and Actions."

Figure 7–11 shows the columns that an ADF Read-only Table component which references the `ProductList` tree binding in the `ExcelReadOnlyPageDef.xml` page definition file of the Master Price List module renders at runtime.

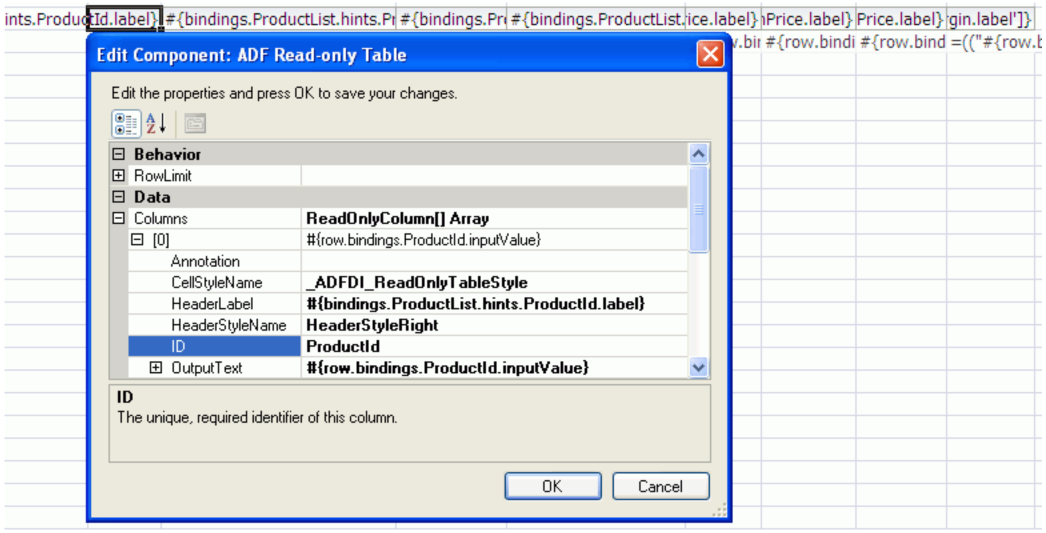
Figure 7–11 Columns in an ADF Read-only Table Component at Runtime



Prod. No	Product Name	Current Product	Supplier	Cost Price	Man. Rec. Price	Site Price	Current Margin
14	Bluetooth Phone Headset	AVAILABLE	Electronics and More	$20.00	$24.99	$49.99	150.0%
15	Ipod Speakers	AVAILABLE	Transistor City	$35.00	$55.99	$89.99	157.1%
16	Creative Zen Vision W 60 GB	AVAILABLE	Transistor City	$290.00	$329.99	$389.99	34.5%

Figure 7–12 shows the corresponding view of the same ADF Read-only Table component at design time with the property inspector in the foreground.

Figure 7–12 ADF Read-only Table Component at Design Time



Edit the properties and press OK to save your changes.

⊟ Behavior	
⊞ RowLimit	
⊟ Data	
⊟ Columns	ReadOnlyColumn[] Array
⊟ [0]	#{row.bindings.ProductId.inputValue}
Annotation	
CellStyleName	_ADFDI_ReadOnlyTableStyle
HeaderLabel	#{bindings.ProductList.hints.ProductId.label}
HeaderStyleName	HeaderStyleRight
ID	ProductId
⊞ OutputText	#{row.bindings.ProductId.inputValue}

ID
The unique, required identifier of this column.

7.16.1 How to Insert an ADF Read-only Table Component

You use the ADF Desktop Integration Designer task pane to insert an ADF Read-only Table component into a worksheet.

To insert an ADF Read-only Table component:

1. Open the integrated Excel workbook.
2. Select the cell in the Excel worksheet where you want to anchor the component.
3. In the bindings palette, select the binding to create the ADF Read-only Table component, and then click **Insert Binding**.
4. In the dialog that appears, select **ADF Read-only Table**.

> **Note:** You can also insert an ADF Read-only Table component by using the components palette or **Oracle ADF** tab. Select **ADF Read-only Table** and click **Insert Component**. If you use the components palette to create the component, you would have to add each column to appear in the component at runtime.

5. Configure properties in the property inspector that appears to determine the columns to appear and the actions the component invokes at runtime.
6. Click **OK**.

> **Note:** You can modify the properties of the component at a later time by selecting the cell in the worksheet that anchors the component and then displaying the property inspector.

7.16.2 How to Manually Add a Column to the ADF Read-only Table Component

You can manually add additional columns to an ADF Read-only Table component or re-add columns that you previously removed.

To manually add a column to the ADF Read-only Table component:

1. Open the integrated Excel workbook.
2. Select the cell in the worksheet that hosts the ADF Read-only Table component and click the **Edit Component** button in the **Oracle ADF** tab to display the Edit Component: ADF Read-only Table dialog.
3. Click the ellipsis button (**...**) beside the input field for **Columns** to invoke the ReadOnlyColumn Collection Editor.
4. Click **Add** to add a new column to the ADF Read-only Table component.
5. Set values for the properties of the new column.

   For information about the properties of an ADF Read-only Table component column, see Table A–13.
6. Click **OK**.

7.17 Limiting the Number of Rows Your Table-Type Component Downloads

You can configure the number of rows that an ADF Table or ADF Read-only Table component downloads by setting values for the component's `RowLimit` group of properties. You can also display a warning message, if desired, that alerts the end user when the number of rows available to download exceeds the number of rows specified for download.

7.17.1 How to Limit the Number of Rows a Component Downloads

Specify the number of rows that the component downloads when it invokes its `Download` action as a value for the `RowLimit.MaxRows` property. Optionally, write an EL expression for the `RowLimit.WarningMessage` property so that the end user receives a message if the number of rows available to download exceeds the number specified by `RowLimit.MaxRows`.

To limit the number of rows a table-type component downloads:

1. Open the integrated Excel workbook.
2. Select the cell in the Excel worksheet that references the table-type component and click the **Edit Component** button in the **Oracle ADF** tab.

   For more information, see Section 8.2, "Using Action Sets."
3. Configure properties for the `RowLimit` group of properties, as described inTable 7–10. For more information about these properties, see Section A.1, "Frequently Used Properties in the ADF Desktop Integration."

Table 7–10 RowLimit Group of Properties

Set this property to...	This value...
`RowLimit.Enabled`	Set to `True` to limit the number of rows downloaded to the value specified by `RowLimit.MaxRows`.
`RowLimit.MaxRows`	Specify an EL expression that evaluates to the maximum number of rows to download.

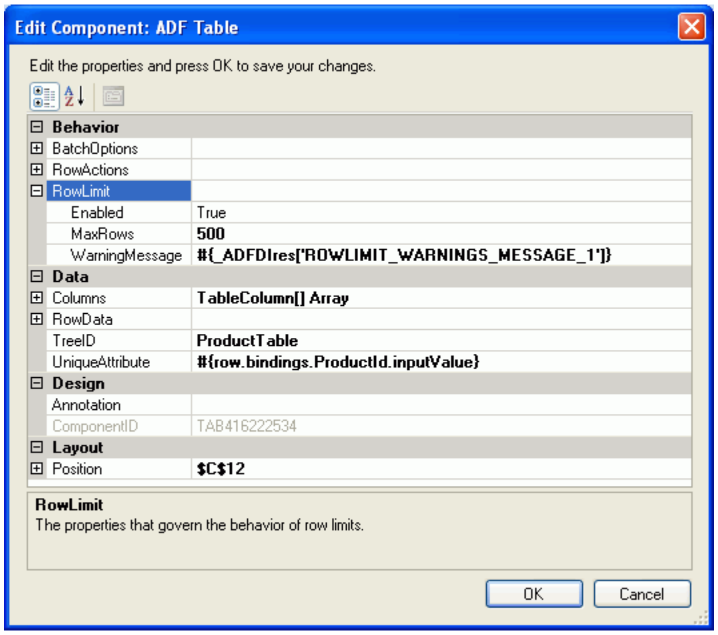
Table 7–10 (Cont.) RowLimit Group of Properties

Set this property to...	This value...
`RowLimit.WarningMessage`	Write an EL expression for this property to generate a message for the end user if the number of rows available to download exceeds the number specified by `RowLimit.MaxRows`. The default value also generates a message: `#{_ADFDIres['ROWLIMIT_WARNINGS_MESSAGE_1']}` If the value for this property is null, the `Download` action downloads the number of rows specified by `RowLimit.MaxRows` without displaying a message to the end user.

4. Click **OK**.

Figure 7–13 shows the Edit Component dialog in the `EditPriceList-DT.xlsx` workbook where the row limit of an ADF Table component is configured.

Figure 7–13 Limiting Number of Rows of an ADF Table Component



Edit the properties and press OK to save your changes.

Behavior	
BatchOptions	
RowActions	
RowLimit	
Enabled	True
MaxRows	500
WarningMessage	#{_ADFDIres['ROWLIMIT_WARNINGS_MESSAGE_1']}
Data	
Columns	TableColumn[] Array
RowData	
TreeID	ProductTable
UniqueAttribute	#{row.bindings.ProductId.inputValue}
Design	
Annotation	
ComponentID	TAB416222534
Layout	
Position	C12

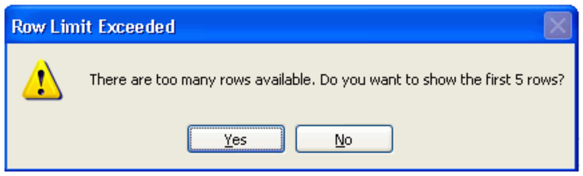
RowLimit
The properties that govern the behavior of row limits.

7.17.2 What Happens at Runtime When You Limit the Number of Rows a Component Downloads

When invoked, the Table-type component's `Download` action downloads the number of rows that you specified as the value for `RowLimit.MaxRows` from the Fusion web application. A message dialog similar to the one in Figure 7–14 appears if you specify an EL expression for `RowLimit.MaxRows` or do not modify its default value:

```
#{_ADFDIres['ROWLIMIT_WARNINGS_MESSAGE_1']}
```

Figure 7–14 Row Limit Exceeded Warning Message


Row Limit Exceeded

There are too many rows available. Do you want to show the first 5 rows?

7.18 Clearing the Values of Cached Attributes in an ADF Table Component

The `RowData` group of properties described in Table A–9 allow you to specify data to cache in the ADF Table component. For more information about this functionality, see the following:

- Section 12.7, "Handling Data Conflicts When Uploading Data from a Workbook"
- Chapter 15, "Using an Integrated Excel Workbook Across Multiple Web Sessions and in Disconnected Mode"

The ADF Table component exposes an action (`ClearCachedRowAttributes`) that, when invoked, clears the values of cached attributes for the current row of the ADF Table component.

Do not configure a component (for example, an ADF Table component's column or an ADF Input Text component) so that an end user can view or edit an attribute binding that you have also specified for an element in the `RowData.CachedAttributes` array. The `RowData.CachedAttributes` array caches the values retrieved by the worksheet `DownSync` action. The worksheet `UpSync` action sends the values of the `RowData.CachedAttributes` array to the Fusion web application. This may override edits an end user makes to an attribute binding exposed through a component in the worksheet.

7.18.1 How to Clear the Values of Cached Attributes in an ADF Table Component

Configure a `DoubleClickActionSet` that includes an action to invoke the ADF Table component's `ClearCachedRowAttributes` action.

To clear the values of cached attributes in an ADF Table component:

1. Open the integrated Excel workbook.
2. Open the Action Collection Editor for the Oracle ADF component that is going to invoke the `DoubleClickActionSet` at runtime.

   For more information about invoking action sets, see Chapter 8.2, "Using Action Sets."
3. Add an action to the `DoubleClickActionSet` that invokes the ADF Table component's `ClearCachedRowAttributes` action.
4. Click **OK**.

7.18.2 What Happens at Runtime When the ADF Table Component Clears Cached Values

The action set invokes the ADF Table component's `ClearCachedRowAttributes` action. This action clears the cached values specified by the

RowData.CachedAttributes property for the current row of the ADF Table component.

7.19 Tracking Changes in an ADF Table Component

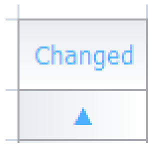
End users can create or modify data in the cells of an integrated Excel workbook that hosts an ADF Table component.

If a column is updatable and not read-only, change tracking is activated. End users can make the following changes to activate change tracking:

- Edit cell values
- Insert or delete cell values
- Paste values to cells in the ADF Table component column that they copied elsewhere

A character that resembles an upward pointing arrow appears in a row of the _ADF_ChangedColumn column if the end user makes a change to data in a corresponding row. Figure 7–15 shows an example.

Figure 7–15 Changed Column in an ADF Table Component



This character appears if the end user makes a change to data hosted by a component where the component's `ReadOnly` property value is `False`. The ADF Input Text and TreeNodeList subcomponents both have a `ReadOnly` property. You can write an EL expression or a static string for this `ReadOnly` property that evaluates to `True` or `False`. If you write a static string or an EL expression that evaluates to `True`, no character appears in the `_ADF_ChangedColumn` column. For more information about `ReadOnly` EL expressions and change tracking, see Section 7.8.2, "What Happens at Runtime When an ADF Table Component Uploads Data."

If you write an EL expression for this `ReadOnly` property that evaluates to `True`, ADF Desktop Integration evaluates it differently to other EL expressions during change tracking. This is because it is not desirable to invoke a connection to the Fusion web application if the end user makes changes to data in an ADF Table component while working in disconnected mode. Instead, ADF Desktop Integration substitutes an empty string value for any part of an EL expression that requires a connection to the Fusion web application. This behavior also applies to the ADF Table component column's `CellStyleName` property.

Note: During change tracking, cell styles are applied when the end user inserts new worksheet rows.

For example, the end user in disconnected mode makes a change to a data value hosted by the ADF Input Text component in an ADF Table component column. During change tracking, ADF Desktop Integration substitutes an empty string value in the parts of the EL expression for the ADF Input Text component's `ReadOnly` property and the ADF Table component column's `CellStyleName` property that require a connection to the Fusion web application. For this reason, write EL expression for these properties that evaluate as you intend if an empty string value is substituted for a part of the expression that requires a connection to the Fusion web application to retrieve a runtime value.

The ADF Output Text component does not have a `ReadOnly` property. Changes that you make to a value hosted by this component, or the ADF Input Text and TreeNodeList subcomponents, do not result in a change to the `_ADF_ChangedColumn` column.

8 Adding Interactivity to Your Integrated Excel Workbook

This chapter describes how to add interactivity options to your integrated Excel workbook.

This chapter includes the following sections:

8.1 Introduction to Adding Interactivity to an Integrated Excel Workbook

Adding interactivity to an integrated Excel workbook permits end users to execute action sets that invoke Oracle ADF functionality in the workbook. It also provides status messages, alert messages, and error handling in the integrated Excel workbook while these action sets execute. In addition to end-user gestures (double-click, click, select) on the ADF Desktop Integration components that invoke action sets, you can configure workbook and worksheet ribbon buttons that end users use at runtime to invoke action sets.

The action sets that end users invoke can make use of functionality defined in the Excel workbook and in pages of the Fusion web application with which you integrate the Excel workbook. For example, the `EditPriceList-DT.xlsx` workbook in the Master Price List module renders an ADF Button component that, at runtime, invokes a page from the Fusion web application. The invoked page allows end users to specify additional search criteria to what can be specified in the workbook's search form which is rendered using ADF Button, ADF Input Text, and ADF Label components.

In addition to action sets, you can configure Excel functionality, such as macros and Excel formulas, to manage the data that you want to download from or upload to your Fusion web application.

8.2 Using Action Sets

An *action set* is an ordered list of one or more of the following actions that execute in a specified order:

- `ADFmAction`
- `ComponentAction`
- `WorksheetMethod`
- `Confirmation`
- `Dialog`

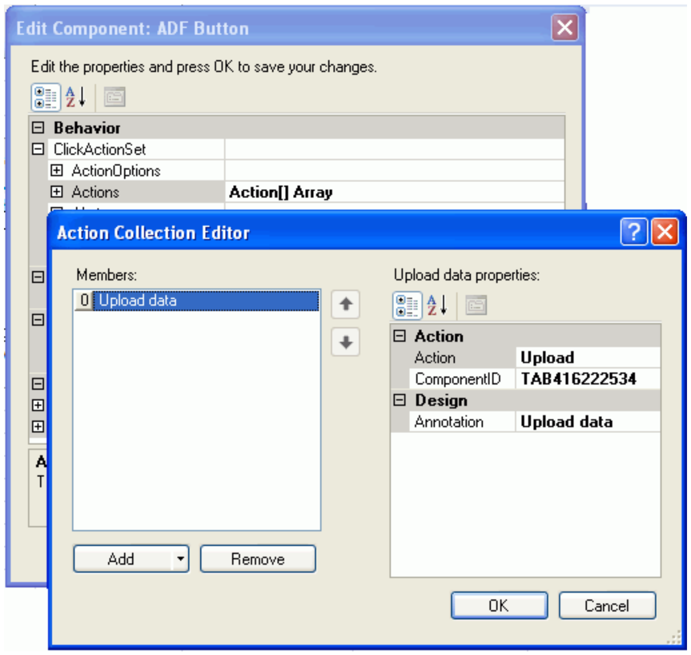
An action set can be invoked by an end-user's gesture (for example, clicking an ADF Button) or an Excel worksheet event. Where an end-user gesture invokes an action set, the name of the action set property in the ADF component's property inspector is prefaced by the name of the gesture required. The following list describes the property names that ADF Desktop Integration displays in property inspectors, and what user gesture can invoke an action set:

- `ClickActionSet` for an ADF Button component, as the end user clicks the button to invoke the associated action set
- `DoubleClickActionSet` for an ADF InputText or ADF Output Text component, as the end user double-clicks these components to invoke the associated action set
- `SelectActionSet` for a worksheet ribbon button, as the end user selects a button to invoke the associated action set
- `ActionSet` for a worksheet event, as no explicit end-user gesture is required to invoke the action set

You invoke the Action Collection Editor from an ADF component, worksheet ribbon button, or worksheet event to define or configure an action set. In addition to defining the actions that an action set invokes, you can configure the action set's `Alert` properties to provide feedback on the result of invocation of an action set. You configure the `Status` properties for an action set to display a status message to end users while an action set executes the actions you define. For information about opening the Action Collection Editor, see Section 5.12, "Using the Collection Editors."

The Master Price List module provides many examples of action sets in use. One example is the ADF Button component labeled **Upload Data** at runtime in the `EditPriceList-DT.xlsx` workbook. An action set has been configured for this ADF Button component that invokes the ADF Table component's `Upload` action illustrated by Figure 8–1 which shows the Action Collection Editor in design mode.

Figure 8–1 Action Set for Upload Data Button in the EditPriceList-DT.xlsx Workbook



Tip: Write a description in the **Annotation** field for each action that you add to the Action Collection Editor. The description you write appears in the **Members** list view and, depending on how you write it, may be more meaningful than the default entry that ADF Desktop Integration generates.

Note: ADF Desktop Integration invokes the actions in an action set in the order that you specify in the **Members** list view.

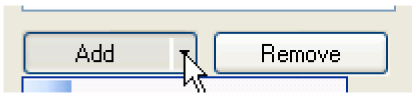
8.2.1 How to Invoke an ADF Model Action in an Action Set

You can invoke multiple ADF Model actions in an action set. An ADF Model action is also known as an action binding in the JDeveloper project where you develop your Fusion web application. Page definition files define what action bindings are available to invoke in a worksheet that you integrate with your Fusion web application. For more information about page definition files and action bindings in an integrated Excel workbook, see Section 4.3, "Working with Page Definition Files for an Integrated Excel Workbook."

You use the Action Collection Editor to specify an ADF Model action to invoke.

To invoke an ADF Model action in an action set:

1. Open the integrated Excel workbook.
2. Open the Action Collection Editor and invoke the dropdown list from the **Add** button illustrated here.



3. Select **ADFmAction** and configure its properties as described in the following list:

- `ActionID`

  Click the ellipsis button (**...**) beside the input field for `ActionID` to invoke the Binding ID picker and select the ADF Model action that the action set invokes at runtime.

- `Annotation`

  Optionally, enter a comment about the purpose of the action that you are configuring. The value you set for this property has no functional impact.

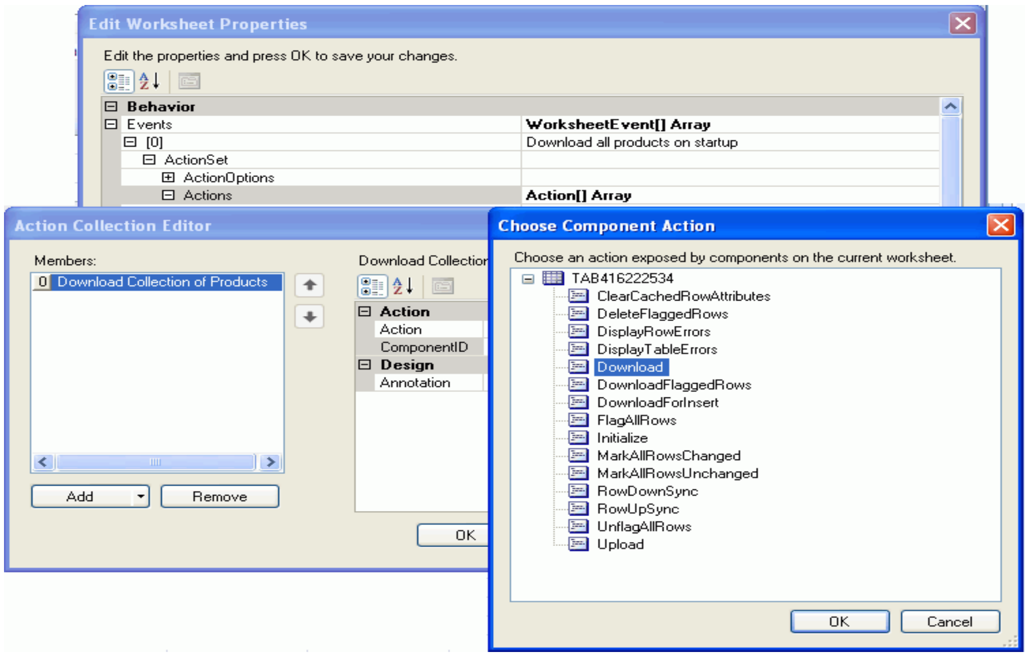
4. Click **OK**.

8.2.2 How to Invoke Component Actions in an Action Set

The ADF Table and the ADF Read-only Table components in ADF Desktop Integration expose actions that can be used to manage the transfer of data between Excel worksheets that you integrate with a Fusion web application. The ADF Read-only Table component exposes one component action, `Download`, while the ADF Table component exposes many other actions. More information about the actions for both components can be found in Appendix A, "ADF Desktop Integration Component Properties and Actions."

You configure action sets to invoke one or more component actions by referencing the component action in the array of actions. For example, Figure 8–2 shows the Choose Component Action dialog where the actions exposed by the ADF Table and ADF Read-only Table components present in a worksheet can be invoked by a `SelectActionSet` action set.

Figure 8–2 Choose Component Method Dialog


Edit Worksheet Properties

Edit the properties and press OK to save your changes.

⊟ Behavior	
⊟ Events	WorksheetEvent[] Array
⊟ [0]	Download all products on startup
⊟ ActionSet	
⊞ ActionOptions	
⊟ Actions	Action[] Array

Action Collection Editor

Members:

0 Download Collection of Products

Download Collection

⊟ Action	
Action	
ComponentID	
⊟ Design	
Annotation	

Choose Component Action

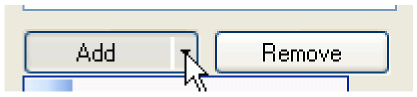
Choose an action exposed by components on the current worksheet.

- TAB416222534
  - ClearCachedRowAttributes
  - DeleteFlaggedRows
  - DisplayRowErrors
  - DisplayTableErrors
  - Download
  - DownloadFlaggedRows
  - DownloadForInsert
  - FlagAllRows
  - Initialize
  - MarkAllRowsChanged
  - MarkAllRowsUnchanged
  - RowDownSync
  - RowUpSync
  - UnflagAllRows
  - Upload

Note: An Excel worksheet must include an ADF Table or ADF Read-only Table component before one or more of these components' actions can be invoked by an action set.

To invoke a component action from an action set:

1. Open the integrated Excel workbook.
2. Open the Action Collection Editor and invoke the dropdown list from the **Add** button illustrated here.



3. Select **ComponentAction** and configure its properties as described in the following list:
   - `ComponentID`

     Click the ellipsis button (**...**) beside the input field for `ComponentID` to invoke the Choose Component Method dialog and select the component action that the action set invokes at runtime. This populates the `ComponentID` and `Method` input fields.
   - `Action`

     The component's action that the action set invokes at runtime.
   - `Annotation`

     Optionally, enter a comment about the purpose of the action that you are configuring. The value you set for this property has no functional impact.
4. Click **OK**.

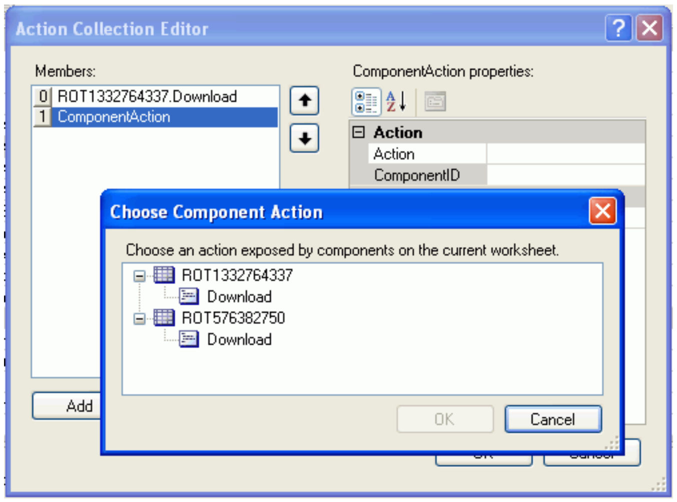
8.2.3 What You May Need to Know About an Action Set Invoking a Component Action

Note the following pieces of information about the behavior of action sets in integrated Excel workbooks.

Verifying an Action Set Invokes the Correct Component Action

When creating an action set, ensure that you invoke the component action from the correct instance of a component when a worksheet includes multiple instances of an ADF Read-only Table or ADF Table component. Figure 8–3 shows the Choose Component Action dialog displaying two instances of the ADF Read-only Table component. Use the value of the `ComponentID` property described in Table A–1 to correctly identify the instance of a component on which you want to invoke a component action.

Figure 8–3 Choose Component Method Dialog



Invoking Action Sets in a Disconnected Workbook

End users can use integrated Excel workbooks while disconnected from a Fusion web application, as described in Chapter 15, "Using an Integrated Excel Workbook Across Multiple Web Sessions and in Disconnected Mode." Some component actions, such as the `Download` action of the ADF Table component, require a connection to the Fusion web application to complete successfully. If the end user invokes an action set that includes such a component action, the integrated Excel workbook attempts to connect to the Fusion web application and, if necessary, invokes the authentication process described in Section 11.2, "Authenticating the Excel Workbook User."

8.2.4 How to Invoke an Action Set from a Worksheet Event

ADF Desktop Integration provides several worksheet events that, when triggered, can invoke an action set. The following worksheet events can invoke an action set:

- `Startup`
- `Shutdown`

  Do not invoke a `Dialog` action from this event if the `Dialog` action's `Target` property is set to `TaskPane`.
- `Activate`
- `Deactivate`

You add an element to the array of events (`WorksheetEvent[] Array`) referenced by the `Events` worksheet property. You specify an event and the action set that it invokes in the element that you add. For more information about the `Events` worksheet property and the worksheet events that can invoke an action set, see Table A–19. See Table A–14 for more information about action sets.

Use the WorkSheetEvent Collection Editor to specify an action set to be invoked by a worksheet event.

To invoke an action set from a worksheet event:

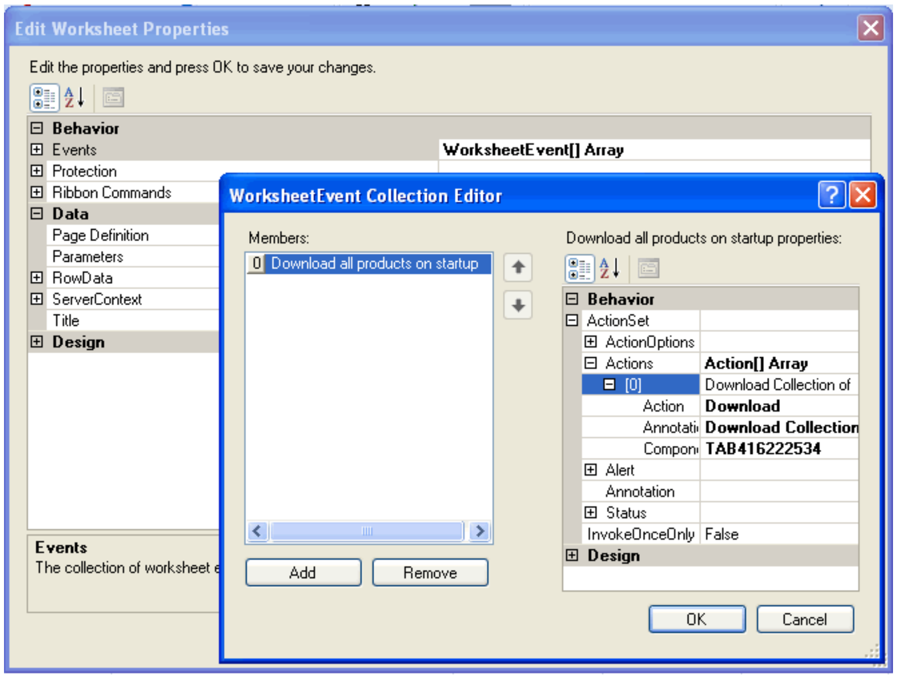
1. Open the integrated Excel workbook.
2. In the ADF Desktop Integration task pane, click **Worksheet Properties** to display the Edit Worksheet Properties dialog.

3. Click the ellipsis button (...) beside the input field for the `Events` property to display the WorksheetEvent Collection Editor.

4. Click **Add** to add a new element that specifies an event and a corresponding action set that the event invokes.

   Figure 8–4 shows an example from the `EditPriceList-DT.xlsx` file in the Master Price List module where the worksheet event, `Startup`, invokes an action set that invokes the ADF Table component's `Download` action.

Figure 8–4 Worksheet Startup Event Invokes an Action Set



Edit Worksheet Properties

Edit the properties and press OK to save your changes.

Behavior	
Events	**WorksheetEvent[] Array**
Protection	
Ribbon Commands	
Data	
Page Definition	
Parameters	
RowData	
ServerContext	
Title	
Design	

Events
The collection of worksheet e

WorksheetEvent Collection Editor

Members:
0 Download all products on startup

Download all products on startup properties:

Behavior	
ActionSet	
ActionOptions	
Actions	**Action[] Array**
[0]	Download Collection of
Action	**Download**
Annotati	**Download Collection**
Compon	**TAB416222534**
Alert	
Annotation	
Status	
InvokeOnceOnly	False
Design	

Add Remove OK Cancel

5. Click **OK**.

8.2.5 How to Display a Status Message While an Action Set Executes

You can display a status message to end users while an action set executes by specifying values for the `Status` properties in an action set.

Some of the default values for properties in the `ActionSet.Status` group are EL expressions that resolve to strings defined in the reserved resource bundle at runtime. You can replace these default values with EL expressions that refer to your custom resource bundles. For more information, see Section 10.2, "Using Resource Bundles in an Integrated Excel Workbook."

You use the Action Collection Editor to configure values for the `ActionSet.Status` properties.

To display a status message:

1. Open the integrated Excel workbook.

2. Open the Action Collection Editor.

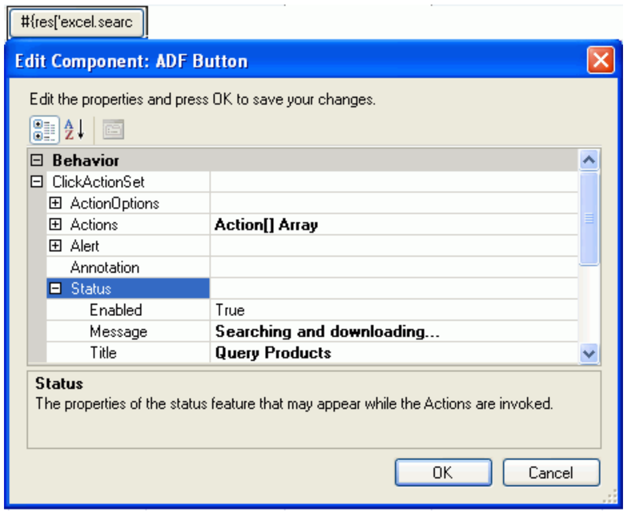
3. Set values for the properties in the `ActionSet.Status` group of properties as described in the following table.

Table 8–1 ActionSet.Status Group of Properties

For this property...	Enter or select this value...
`Enabled`	`True` to display a status message. `True` is the default value.
`Message`	An EL expression or string that resolves to the status message to display at runtime. For example, the **Search** button in the Master Price List module's `EditPriceList-DT.xlsx` file has the following value configured for the `Message` property: `Searching and downloading...`
Title	An EL expression or string that resolves to the title of the status message to display at runtime. For example, the **Search** button in the Master Price List module's `EditPriceList-DT.xlsx` file has the following value configured for the `Title` property: `Query Products`
For this property...	Enter or select this value...

Figure 8–5 shows the values configured for the `ActionSet.Status` group of properties of the Search ADF Button component in the `EditPriceList-DT.xlsx` workbook of the Master Price List module that is labeled **Search** at runtime.

Figure 8–5 Status Message Properties in an Action Set



Edit the properties and press OK to save your changes.

Behavior	
ClickActionSet	
ActionOptions	
Actions	Action[] Array
Alert	
Annotation	
Status	
Enabled	True
Message	Searching and downloading...
Title	Query Products

Status
The properties of the status feature that may appear while the Actions are invoked.

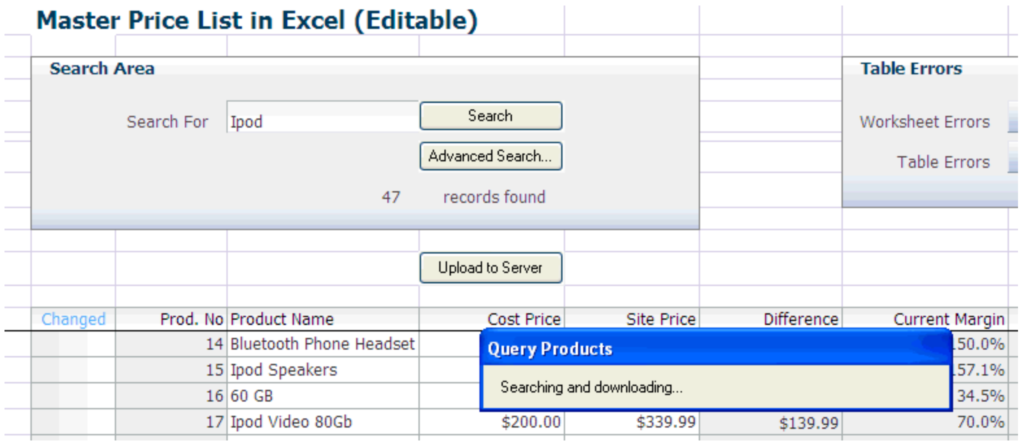
For more information about the `ActionSet.Status` group of properties, see the entry for `Status` in Table A–14.

4. Click **OK**.

8.2.6 What Happens at Runtime When an Action Set Displays a Status Message

Once an action set is invoked, a status message appears if the `ActionSet.Status` properties are configured to display a status message. Figure 8–6 shows the status message that appears at runtime when the action set configured for the **Search** button in the `EditPriceList-DT.xlsx` workbook executes.

Figure 8–6 Runtime View of Status Message



Master Price List in Excel (Editable)

Search Area

Search For: Ipod

47 records found

Table Errors

Worksheet Errors

Table Errors

Changed	Prod. No	Product Name	Cost Price	Site Price	Difference	Current Margin
	14	Bluetooth Phone Headset				50.0%
	15	Ipod Speakers				57.1%
	16	60 GB				34.5%
	17	Ipod Video 80Gb	$200.00	$339.99	$139.99	70.0%

Query Products

Searching and downloading...

8.2.7 How to Provide an Alert After the Invocation of an Action Set

You can display an alert message to end users that notifies them when an action set operation completes successfully or fails. For example, you can display a message when all actions in an action set succeed or when there was at least one failure. The `ActionSet.Alert` group of properties configures this behavior.

Note: An alert message does not appear if the end user cancels the execution of an action set. For example, you configure an alert message to appear after an action set that invokes a web page in a popup dialog completes execution. At runtime, the end user cancels execution of the action set by closing the popup dialog using the close button of the Excel web browser control that hosts the popup dialog. In this scenario, no alert message appears. For more information about displaying web pages, see Section 8.4, "Displaying Web Pages from a Fusion Web Application."

Many of the default values for properties in the `ActionSet.Alert` group are EL expressions that resolve to strings defined in the reserved resource bundle at runtime. You can replace these default values with EL expressions that refer to your custom resource bundles. For more information, see Section 10.2, "Using Resource Bundles in an Integrated Excel Workbook."

You use the Action Collection Editor to configure values for the `ActionSet.Alert` group of properties.

To add an alert to an action set:

1. Open the integrated Excel workbook.
2. Open the Action Collection Editor.
3. Set values for the properties in the `ActionSet.Alert` group of properties as described in Table 8–2.

Table 8–2 ActionSet.Alert Group of Properties

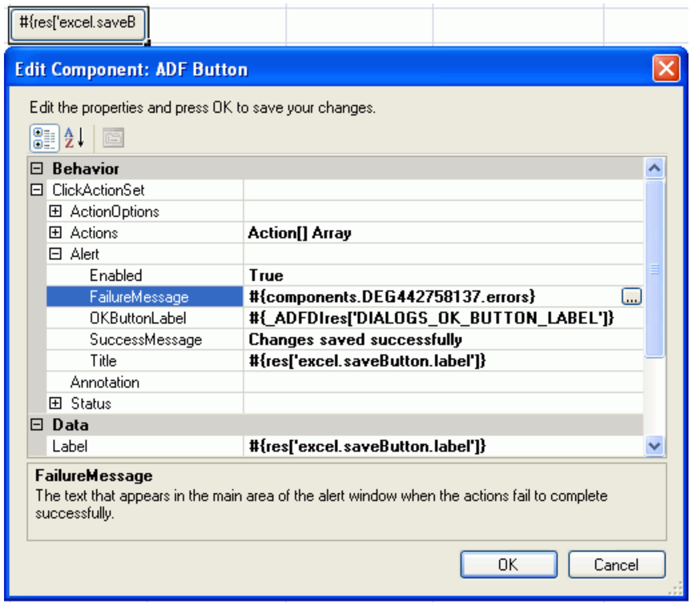
For this property...	Enter or select this value...
`Enabled`	Select `True` from the dropdown list to display an alert message once the action set completes. The default value is `False`.

Table 8–2 (Cont.) ActionSet.Alert Group of Properties

For this property...	Enter or select this value...
`FailureMessage`	Specify an EL expression or string that evaluates to a message to appear in the dialog if errors occur during execution of the action set. For example, the **Upload to Server** button in the Master Price List module's `EditPriceList-DT.xlsx` workbook has the following value configured for the `FailureMessage` property: `#{components.TAB442758137.errors}` The **Upload to Server** button invokes an action set that, in turn, invokes the ADF Table component's `Upload` action. The EL expression specified for `FailureMessage` retrieves error messages if the `Upload` action encounters errors. For more information about error handling, see Section 12.4, "Error Reporting in an Integrated Excel Workbook."
`OKButtonLabel`	Specify an EL expression that evaluates to a message to appear in the **OK** button of the dialog. The default EL expression is: `#{_ADFDIres['DIALOGS_OK_BUTTON_LABEL']}`
`SuccessMessage`	Specify an EL expression that evaluates to a message to appear in the dialog if no errors occur during the execution of the action set. For example, the **Save Changes** button in the Master Price List module's `EditPriceList-DT.xlsx` workbook has the following value configured for the `SuccessMessage` property: `Changes saved successfully`

Figure 8–7 shows the values configured for an ADF Button component's `ActionSet.Alert` group of properties in the `EditPriceList-DT.xlsx` workbook of the Master Price List module. This ADF Button component is labeled **Upload to Server** at runtime.

Figure 8–7 Alert Message Properties in an Action Set



Edit the properties and press OK to save your changes.

Behavior	
ClickActionSet	
ActionOptions	
Actions	Action[] Array
Alert	
Enabled	True
FailureMessage	#{components.DEG442758137.errors}
OKButtonLabel	#{_ADFDIres['DIALOGS_OK_BUTTON_LABEL']}
SuccessMessage	Changes saved successfully
Title	#{res['excel.saveButton.label']}
Annotation	
Status	
Data	
Label	#{res['excel.saveButton.label']}

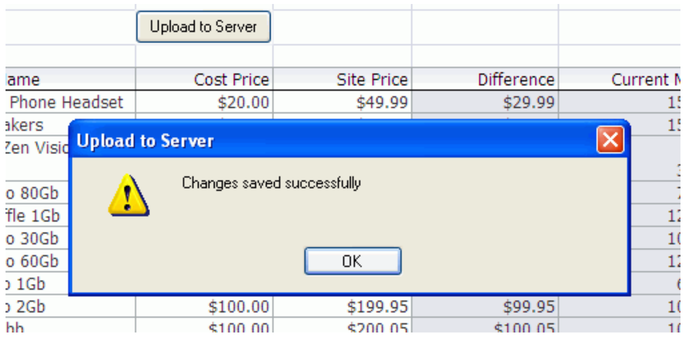
FailureMessage
The text that appears in the main area of the alert window when the actions fail to complete successfully.

4. Click **OK**.

8.2.8 What Happens at Runtime When an Action Set Provides an Alert

Figure 8–8 shows the alert message that appears at runtime when the action set invoked by the ADF Button component labeled **Upload to Server** successfully completes execution.

Figure 8–8 Runtime View of an Alert Message



Changes saved successfully

8.2.9 How to Configure Error Handling for an Action Set

You specify values for an action set's `ActionOptions` properties to determine what an action set does if one of the following events occurs:

- An action in the action set fails
- All actions in the action set complete successfully

For information about how to invoke these editors, or about an ADF component's property inspector, see Chapter 5, "Getting Started with the Development Tools." More information about action set properties can be found in Table A.11.

To configure error handling for an action set:

1. Open the integrated Excel workbook.
2. Open the appropriate editor or property inspector and configure values for the action set's `ActionOptions` properties as described in the following table.

Table 8–3 ActionOptions Properties

Set this property...	To...
`AbortOnFailure`	`True` (default value) so that the action set does not any execute any further actions if the current action fails. When set to `False`, the action set executes all actions regardless of the success or failure of previous actions.
`FailureActionID`	Specify an ADF Model action to invoke if an action set does not complete successfully. For example, you could specify an ADF Model action that rolls back changes made during the unsuccessful invocation of the action set. Note that calling an action set that changes a record set's currency during the execution of `FailureActionID` methods is not supported. The Rollback method also should not be specified as the `FailureActionID` in an action set.

Table 8–3 (Cont.) ActionOptions Properties

Set this property...	To...
`SuccessActionID`	Specify an ADF Model action to invoke if an action set completes successfully. For example, you could specify an action binding that executes a `commit` action. A value for this property is optional and you can specify a final action, such as an action binding that executes a `commit` action, in the action set itself. Note that calling an action set that changes a record set's currency during the execution of `SuccessActionID` methods is not supported.

3. Click **OK**.

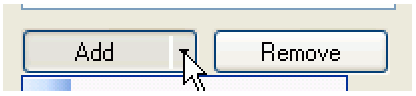
8.2.10 How to Invoke a Confirmation Action in an Action Set

The Confirmation action presents the end user with a simple message dialog that displays the title and prompt message specified in the Confirmation action properties.

The execution of the action set pauses until the end user clicks one of the two buttons provided. If the user clicks **OK**, the action sets proceed with the remaining actions in the Action Set. If the user clicks **Cancel**, the action set is aborted at that point and the remaining actions are not invoked. As there is no error or success, the `FailureActionID` or `SuccessActionID` action is not invoked.

To invoke a Confirmation action from a component

1. Open the integrated Excel workbook.
2. Open the Action Collection Editor and click the down arrow in the **Add** button to open a dropdown list, as illustrated here.



3. Select **Confirmation** and configure its Data properties as described in the following list:

   - `CancelButtonLabel`

     Specify an EL expression or string that evaluates to a message to appear in the **Cancel** button of the dialog. The default EL expression is:

     ```
     #{_ADFDIres['DIALOGS_CANCEL_BUTTON_LABEL']}
     ```

   - `OKButtonLabel`

     Specify an EL expression or string that evaluates to a message to appear in the **OK** button of the dialog. The default EL expression is:

     ```
     #{_ADFDIres['DIALOGS_OK_BUTTON_LABEL']}
     ```

   - `Prompt`

     Specify an EL expression or string that evaluates to a message to appear as the prompt of the dialog. The default EL expression is:

     ```
     #{_ADFDIres['DIALOGS_ACTION_CONFIRM_PROMPT']}
     ```

   - `Title`

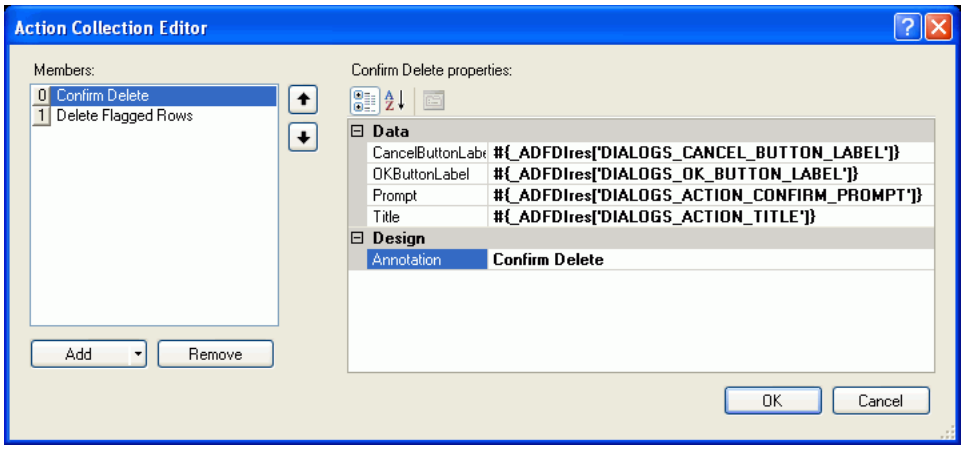
Specify an EL expression or string that evaluates to a title of the confirmation dialog to display at runtime. The default EL expression is:

```
#{_ADFDIres['DIALOGS_ACTION_TITLE']}
```

4. Optionally, enter a comment in the `Annotation` property about the purpose of the action that you are configuring. The value you set for this property has no functional impact.
5. Click **OK**.

Figure 8–9 shows the Action Collection Editor with default attribute values for a Delete button.

Figure 8–9 Confirmation Action Attributes



Data	
CancelButtonLabel	#{_ADFDIres['DIALOGS_CANCEL_BUTTON_LABEL']}
OKButtonLabel	#{_ADFDIres['DIALOGS_OK_BUTTON_LABEL']}
Prompt	#{_ADFDIres['DIALOGS_ACTION_CONFIRM_PROMPT']}
Title	#{_ADFDIres['DIALOGS_ACTION_TITLE']}
Design	
Annotation	Confirm Delete

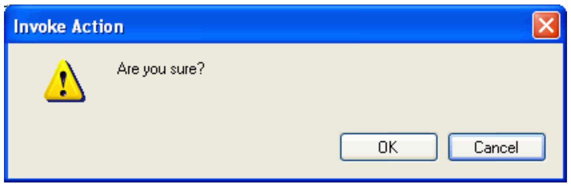
8.2.11 What Happens at Runtime When an Action Set Provides a Confirmation

Once the action set is invoked, the user is prompted with a confirmation dialog. If the user clicks **OK**, the next action operation is performed; and if the user clicks **Cancel**, the Action Set execution terminates without an error.

Note: If the user cancels a Confirmation action, the `FailureActionID` binding does not run.

Figure 8–10 shows a default Confirmation dialog with OK and Cancel buttons.

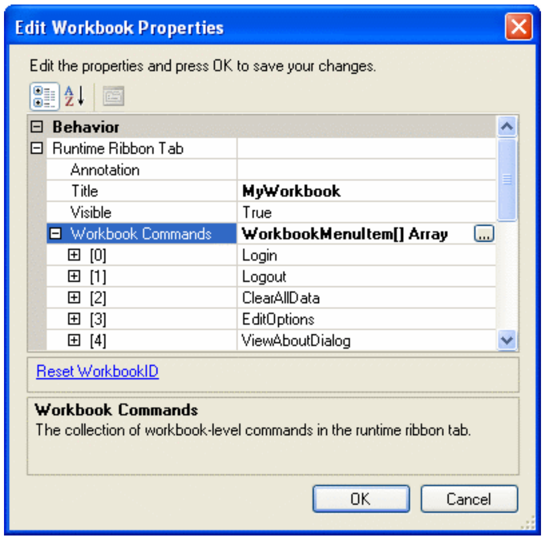
Figure 8–10 Confirmation Dialog



8.3 Configuring the Runtime Ribbon Tab

You can configure the runtime ribbon tab in the Excel Ribbon with items that invoke Oracle ADF functionality in your integrated Excel workbook. In the Runtime Ribbon Tab group, setting the `Visible` workbook property to `True` makes this tab appear at runtime. The `Title` property determines the title of the tab that the end user sees at runtime. By default, the title is **MyWorkbook**, as illustrated in Figure 8–11.

Figure 8–11 Workbook Properties for Runtime Ribbon Tab


Edit Workbook Properties

Edit the properties and press OK to save your changes.

Behavior	
Runtime Ribbon Tab	
Annotation	
Title	**MyWorkbook**
Visible	True
Workbook Commands	**WorkbookMenuItem[] Array**
[0]	Login
[1]	Logout
[2]	ClearAllData
[3]	EditOptions
[4]	ViewAboutDialog

Reset WorkbookID

Workbook Commands
The collection of workbook-level commands in the runtime ribbon tab.

At runtime, the tab appears as the last tab in the Ribbon and all your configured commands appear in various groups of the tab, as illustrated by Figure 8–12.

Figure 8–12 Runtime View of the Ribbon Tab

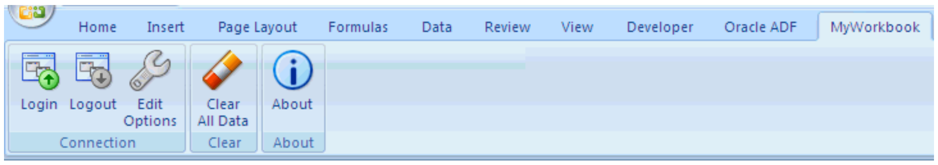
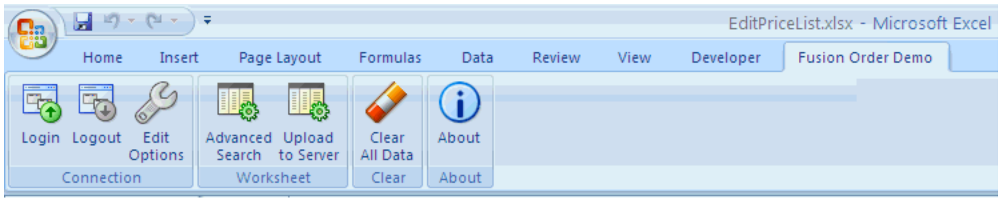


Figure 8–13 illustrates the runtime ribbon tab in `EditPriceList.xlsx` with two commands configured for worksheet. At runtime, the commands are divided into four groups: items that invoke commands on the workbook, items that invoke commands on the current worksheet, a command group to clear all data, and a command workgroup to display ADF Desktop Integration version information.

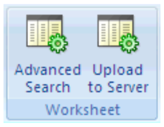
Figure 8–13 Runtime View of Ribbon Tab in `EditPriceList.xlsx`



You configure the `Workbook Commands` property in the properties of the workbook so that the runtime ribbon tab contains commands that allow the end user to invoke workbook actions such as `Login` and `Logout`. You configure the `Ribbon Commands` property in the properties of the worksheet so that the ADF Desktop Integration tab contains items allowing a user to invoke an action set. Worksheet command items appear when the worksheet is active. If you remove a workbook command, it does not appear in the runtime tab for that workbook. If you remove all the commands for a given group, the group does not appear when that workbook is active.

Figure 8–14 shows the Worksheet group at runtime where the worksheet actions, that invoke `SelectActionSet` action sets, appear.

Figure 8–14 Runtime Worksheet Group



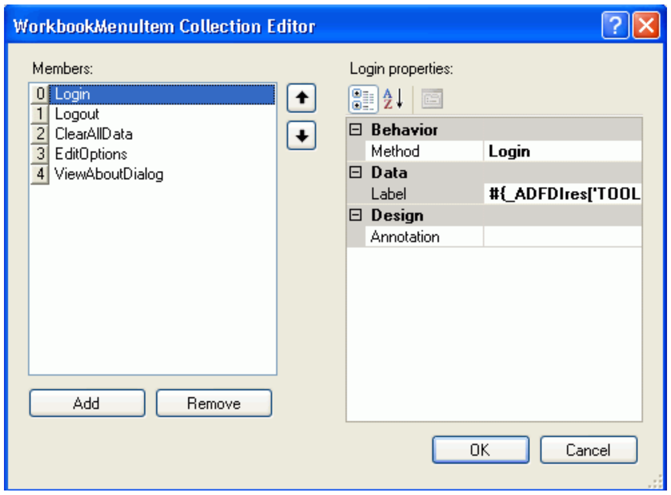
8.3.1 How to Define a Workbook Command Button for the Runtime Ribbon Tab

To define a workbook command button for the runtime ribbon tab, you configure some workbook properties. The following procedure shows how to create or remove an item in the Workbook group by using the workbook action, `Login`, as an example.

To define a workbook command button:

1. Open the integrated Excel workbook.
2. Click **Workbook Properties** in the ADF Desktop Integration task pane.
3. Click **Workbook Commands** and then click the ellipsis button (**...**) beside the `WorkbookMenuItem[]` array to display the dialog as illustrated in Figure 8–15.

Figure 8–15 WorkbookMenuItem Collection Editor



4. Click **Add** and specify values for the properties of the workbook command buttons as follows:

   – `Method`

     Specify the workbook action that you want the workbook command button to invoke.

   – `Label`

Enter a value in the input field that appears as the label at runtime. Alternatively, invoke the expression builder by clicking the ellipsis button (**...**) and write an EL expression that resolves to a string value in a resource bundle.

Note that the runtime value that appears in the label cannot exceed 1024 characters.

For more information about using resource bundles, see Section 10.2, "Using Resource Bundles in an Integrated Excel Workbook."

For more information about labels, see Section 9.4, "Using Labels in an Integrated Excel Workbook."

5. Click **OK**.

Note: The order of workbook commands in the workbook collection editor is ignored at runtime. The order and grouping of the workbook-level commands is always the same.

8.3.2 How to Configure a Worksheet Command for the Runtime Ribbon Tab

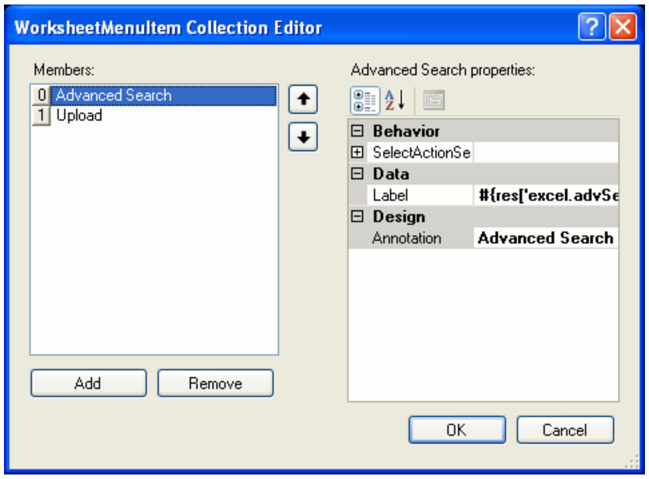
To define a worksheet command, you configure properties for the worksheet using the property inspector. By default, no command buttons are defined for the Worksheet group in the worksheet properties. You add members to the list that is referenced by the `Ribbon Commands` property in the properties of the worksheet.

CAUTION: Set the Runtime Ribbon Tab.Visible workbook property to TRUE to display command buttons. If the Runtime Ribbon Tab.Visible is set to FALSE, no command buttons appear. For more information about workbook properties, see Table A–18.

To define a worksheet command button:

1. Open the integrated Excel workbook.
2. Click **Worksheet Properties** in the ADF Desktop Integration task pane.
3. Click the ellipsis button (**...**) beside the input field for the `Ribbon Commands` property to invoke the editor, as illustrated in Figure 8–16. Figure 8–14 displays how the commands appear at runtime.

Figure 8–16 Worksheet Properties Collection Editor


WorksheetMenuItem Collection Editor

Members:

0	Advanced Search
1	Upload

Advanced Search properties:

Behavior	
SelectActionSe	
Data	
Label	#{res['excel.advSe
Design	
Annotation	Advanced Search

4. Click **Add** to add a new ribbon button in the **Members** list of the collection editor.
5. Configure the properties of `SelectActionSet` to specify the type of action(s) that the ribbon button invokes.
6. Click **OK**.

Note: At runtime, the worksheet commands appear in the same order as they are defined in the worksheet collection editor.

8.4 Displaying Web Pages from a Fusion Web Application

You configure a `Dialog` action in an action set to display pages from the Fusion web application with which you integrate your Excel workbook. These pages provide additional functionality for your integrated Excel workbook. Examples of additional functionality that you can provide include search dialogs and display pick dialogs that interact with your Fusion web application. You can also configure upload options.

The `Dialog` action in an action set can be configured to display in one of the following two types of dialog:

- Popup dialog
- Runtime task pane

The value for the `Dialog.Target` property (`Popup` or `TaskPane`) of the component's action set determines where a web page is rendered.

The value for the `Dialog.Page` property specifies the web page to display when the action is invoked. Valid values include a URL relative to the value of the `WebAppRoot` property or an absolute URL. For example, the `EditPriceList-DT.xlsx` workbook in the Master Price List module specifies the following relative URL as a value for the page to invoke when a user clicks the **Advanced Search** button at runtime:

```
/faces/secured/excelAdvSearch.jspx
```

Absolute URLs such as the following are also valid:

```
http://www.oracle.com/technetwork/middleware/index.html
```

Note: The HTML `<select>` components, such as list box or dropdown list, do not follow `z-order` configuration when the page is displayed through `Dialog` actions. In the .NET Web Browser control, on a web page with layered and overlapping components, the `<select>` components might appear on top of other components.

8.4.1 How to Display a Web Page in a Popup Dialog

You can configure a `Dialog` action in an action set to invoke a web page from your Fusion web application in a modal popup dialog hosted by Excel's web browser control. This feature provides end users with functionality that allows them to, for example, input values displayed by a page from the Fusion web application into the integrated Excel workbook.

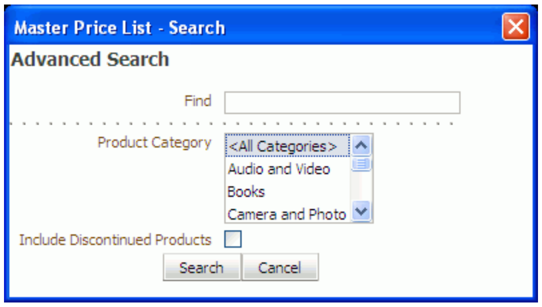
The web page that the action set invokes must contain a reserved HTML Document Object Model (DOM) element (for example, a span element) that has a case-sensitive ID attribute set to `ADFdi_CloseWindow`. Example 8–1 shows how you can automatically set the value of the span element in the `excelAdvSearch.jspx` page of the Master Price List module using the `rendered` property of the `f:verbatim` tag.

Example 8–1 Use of HTML Document Object Model Span Element

```
<f:verbatim rendered="#{requestScope.searchAction eq 'search'}">
<span id="ADFdi_CloseWindow">Continue</span>
</f:verbatim>
<f:verbatim rendered="#{requestScope.searchAction eq 'cancel'}">
<span id="ADFdi_CloseWindow">Abort</span>
</f:verbatim>
```

Figure 8–17 shows the `excelAdvSearch.jspx` page hosted by the `EditPriceList-DT.xlsx` workbook's browser control.

Figure 8–17 Advanced Search Popup Dialog



In scenarios where you cannot use the `rendered` property of the `f:verbatim` tag as outlined in Example 8–1, you may need to:

1. Create a backing bean that exposes the `Dialog` action's result value as a property
2. Use an action listener to invoke the backing bean, and an EL expression in the span element to set the value `ADFdi_CloseWindow` to the bean property value.

Whichever approach you take, ADF Desktop Integration monitors the value of `ADFdi_CloseWindow` to determine when to close the popup dialog. If `ADFdi_CloseWindow` references:

- An empty string or is not present, the popup dialog remains open.

- "`Continue`", the popup dialog closes and the action set invokes its next action.

  The following example shows `ADFdi_CloseWindow` assigned a value of "`Continue`":

  ```
  var closeWindowSpan = document.getElementById("ADFdi_
  CloseWindow");
  ```

  ```
  closeWindowSpan.innerHTML = "Continue";
  ```

- Some other string value, the popup dialog remains open.

You set the `Target` property for a `Dialog` action to `Popup` to display a web page from the Fusion web application in a modal popup dialog hosted by Excel's web browser control. Displaying a web page in a modal popup dialog differs from displaying a web page in Excel's task pane, because the `Dialog` action that the action set invokes cannot continue execution until it receives user input. While the popup dialog is open, the end user cannot interact with any other part of the integrated Excel workbook, as the popup dialog retains focus.

End users can navigate between multiple web pages from the Fusion web application within the browser control until they close the browser control, or ADF Desktop Integration closes it.

To immediately synchronize the changes that the end user makes to a data control through a popup dialog, specify the next action in the action set after the `Dialog` action to download all modified bindings to the worksheet (use the `DownSync` worksheet action) or ADF Table component (use the `Download` action). This scenario assumes that you specify `"Continue"` as the value for `ADFdi_CloseWindow`.

Note: If you configure the web page that appears in the popup dialog so that the end user can download an integrated Excel workbook, the Oracle ADF functionality in the integrated Excel workbook is disabled when the end user opens the workbook after download.

8.4.2 How to Display a Web Page in ADF Desktop Integration Runtime Task Pane

You set the `Dialog.Target` property for an action to `TaskPane` to display a web page specified by the `Dialog.Page` property in the ADF Desktop Integration task pane. In contrast to displaying a web page in a popup dialog, displaying a web page in the task pane allows an action set to continue executing actions while the web page displays. End users can access and interact with other parts of the integrated Excel workbook while the web page displays.

Note the following if you set the `Target` property of a `Dialog` action to `TaskPane`, ADF Desktop Integration ignores the value of `ADFdi_CloseWindow` (and other elements.

8.4.3 What You May Need to Know About Displaying Pages from a Fusion Web Application

You can keep the data an integrated Excel workbook contains synchronized with a Fusion web application by specifying additional actions in the action set that invokes the `Dialog` action. You can ensure that the Fusion web application page and the integrated Excel worksheet both use the same data control frame by setting the `ShareFrame` property of the `Dialog` action.

Notes:

- If your custom web page is based on ADF Faces and opens a popup window, the web page must be configured in a certain way to work properly. On the command component, set the `windowEmbedStyle` to **inlineDocument**. For more information, see *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.
- The `Dialog.Page` property does not accept EL expressions.

Keeping an Integrated Excel Workbook and a Fusion Web Application Synchronized

To ensure that data in the integrated Excel workbook and the Fusion web application remains synchronized while end users use pages from the Fusion web application, configure the action set that invokes the `Dialog` action to:

- Send changes from the integrated Excel workbook to the Fusion web application before invoking the `Dialog` action.

  Invoke the `RowUpSync` worksheet action to synchronize changes from the current row in the ADF Table component.
- Send changes from the Fusion web application to the integrated Excel workbook after invoking the `Dialog` action.

  Invoke the `RowDownSync` worksheet action to send changes from the Fusion web application to the current row in the ADF Table component.

For `DoubleClickActionSet`, you must ensure that the server-side model is in the same state after executing the action set as it was before executing the action set. In most cases, it is sufficient to roll back any and all uncommitted changes at the end of each `DoubleClickActionSet`, as there are no pending uncommitted changes when the action set execution begins.

For more information about synchronizing data between an integrated Excel workbook and a Fusion web application, see Chapter 15, "Using an Integrated Excel Workbook Across Multiple Web Sessions and in Disconnected Mode." For information about worksheet actions and ADF Table component actions, see Chapter A, "ADF Desktop Integration Component Properties and Actions."

Sharing Data Control Frames Between Integrated Excel Worksheets and Fusion Web Application Pages

Fusion web applications and integrated Excel workbooks both use data control frames to manage the transactions and state of view objects and, by extension, the bindings exposed in a page definition file. When you invoke a Fusion web application's page from an integrated Excel worksheet, you can ensure that the page and the integrated Excel worksheet both use the same data control frame by setting the `ShareFrame` property of the `Dialog` action that invokes the page to `True`.

The `Page` property in the `Dialog` action specifies the page that the `Dialog` action invokes. If the `Dialog` action invokes an absolute URL or a page that is not part of your Fusion web application, ADF Desktop Integration ignores the value of `ShareFrame` if `ShareFrame` is set to `True`.

Set `ShareFrame` to `False` in the following scenarios:

- The `Dialog.Page` property in the action set references an absolute URL or a page that is not part of your Fusion web application.

- The `Dialog.Page` property in the action set references a page that is part of your Fusion web application, but that does not need to share information with the integrated Excel worksheet. For example, a page that displays online help information.

For more information about data control frames in a Fusion web application, see the "Sharing Data Control Instances" section of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Configuring a Fusion Web Application for ADF Desktop Integration Frame Sharing

When you add the ADF Desktop Integration Technology scope to your Fusion web application, the application is automatically configured to support ADF Desktop Integration frame sharing. Frame sharing allows each worksheet of an integrated Excel workbook to use a dedicated `DataControl` frame. Web pages displayed in dialogs invoked from each worksheet can then share the same `DataControl` frame as the integrated Excel worksheet.

To verify that your Fusion web application is configured to support frame sharing:

1. Open your Fusion web application project in JDeveloper.
2. In the Application Navigator, expand the Application Resources panel.
3. Open the `adf-config.xml` file available in **Descriptors** > **ADF META-INF** folder.
4. Click the **Source** tab to open the source editor.
5. Confirm that the following `adf-desktopintegration-servlet-config` element is present in the file before the `</adf-config>` tag:

```
<adf-desktopintegration-servlet-config
xmlns="http://xmlns.oracle.com/adf/desktopintegration/servlet/config">
    <controller-state-manager-class>
       oracle.adf.desktopintegration.controller.impl.ADFcControllerStateManager
    </controller-state-manager-class>
</adf-desktopintegration-servlet-config>
```

6. Save the `adf-config.xml` file and close JDeveloper.

8.5 Inserting Values in ADF Table Columns from a Web Page Pick Dialog

You can configure the `DoubleClickActionSet` of an ADF Table component's column to invoke a Fusion web application page that renders a pick dialog where the end user selects a value to insert in the ADF Table component column.

This functionality is useful when you want to constrain the values that end users can enter in an ADF Table component. For example, you may want a runtime ADF Table component column to be read-only in the Excel worksheet so that end users cannot manually modify values to prevent them from introducing errors. Invoking a pick dialog rendered by a Fusion web application page allows the end user to change values in the ADF Table component without entering incorrect data.

In addition to configuring the `DoubleClickActionSet`, you configure the ADF Table component's `RowData.CachedAttributes` property to reference attribute binding values if you want:

- End users to modify values in the Fusion web application's page that you do not want to appear in the ADF Table component of the integrated Excel workbook
- An ADF Table component's column to be read-only in the integrated Excel workbook
- Cache data in an ADF Table component over one or more user sessions that is not visible to end users but is modified by a pick dialog

  For example, an ADF Table component displays a list of product names to end users. A pick dialog is invoked that refreshes the list of product names in the ADF Table component and, as part of the process, sets the value of product IDs. In this scenario, you specify the attribute binding value for the product ID in the ADF Table component's `RowData.CachedAttributes` property. After the action set executes, the ADF Table component displays the refreshed list of product names in the rows of the Excel worksheet and references the associated product IDs in its `RowData.CachedAttributes` property.

For information about populating values in the pick dialog, see the "Creating Databound Selection Lists and Shuttles" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

To invoke a pick dialog from an ADF Table component:

1. Open the integrated Excel workbook.
2. Select the cell in the Excel worksheet that anchors the ADF Table component and click the Edit Properties button in the **Oracle ADF** tab to display the property inspector.
3. Configure the ADF Table component's `RowData.CachedAttributes` property to reference attribute binding values.
4. Click the ellipsis button (**...**) beside the input field for **Columns** to display the TableColumn Collection Editor.
5. In the **Members** list, select the column from which the end user invokes the pick dialog at runtime.
6. Configure the `DoubleClickActionSet` of the `UpdateComponent` property, as described in Table 8–4.

Table 8–4 DoubleClickActionSet Properties

Add this action...	To...
`ADFmAction`	(Optional) Invoke the `CreateInsert` action binding if the end user invokes the `DoubleClickActionSet` from a newly created row in the Excel worksheet's ADF Table component. In this scenario, the ADF Table component's `RowUpSync` action (invoked in the next action) fails if the Fusion web application does not contain a placeholder row.
`ComponentAction`	Invoke the ADF Table component's `Table.RowUpSync` action to synchronize any pending changes in the current row of the ADF Table component to the Fusion web application.
`Dialog`	Configure the `Dialog` action to invoke the pick dialog page from the Fusion web application. Set the `Dialog` action's `ShareFrame` property to `True`. For more information, see Section 8.4, "Displaying Web Pages from a Fusion Web Application."

Table 8–4 (Cont.) DoubleClickActionSet Properties

Add this action...	To...
`ComponentAction`	Invoke the ADF Table component's `Table.RowDownSync` action to synchronize data from the row in the ADF Table component's iterator in the Fusion web application that corresponds to the current ADF Table component row in the worksheet. If you added a `CreateInsert` action binding, you should also add the `Delete` action binding to remove the placeholder row.

7. Click **OK**.

8.6 Creating ADF Databound Search Forms in an Integrated Excel Workbook

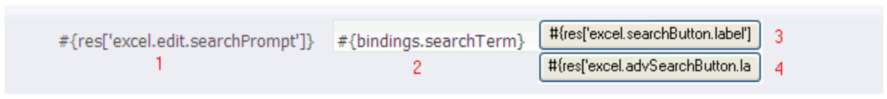
You can create forms in your integrated Excel workbooks using ADF Input Text and ADF Button components. End users can use the forms you create to insert data or query for information. This section uses the latter example to demonstrate how you create forms.

End users can enter a search term in the ADF Input Text component and retrieve matching results by clicking an ADF Button component. To present a more sophisticated user interface to end users for a search operation, you can invoke search forms from your Fusion web application. Results from these search operations can be downloaded to the ADF Table or ADF Read-only Table components in your integrated Excel workbook.

Figure 8–18 shows a design time view of the Oracle ADF components that the `EditPriceList-DT.xlsx` workbook in the Master Price List module uses to configure search options where:

1. ADF Label component is used in a simple search form
2. ADF Input Text component is used in a simple search form
3. ADF Button component is used in a simple search form
4. ADF Button component is used to invoke an advanced search form

Figure 8–18 Oracle ADF Components Used for Search in the EditPriceList-DT.xlsx Workbook


#{res['excel.edit.searchPrompt']}
1
#{bindings.searchTerm}
2
#{res['excel.searchButton.label']}
3
#{res['excel.advSearchButton.la
4

Note: ADF Desktop Integration does not support usage of the `FindMode` attribute in page definition files. For more information about the `FindMode` attribute, see the "*pageName*PageDef.xml" section of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

8.6.1 How to Create a Simple Search Form in an Integrated Excel Workbook

You insert an ADF Input Text component and configure it so that the end user can enter a search term. Insert an ADF Button component and configure its action set to:

1. Take the value the end user enters in the ADF Input Text component.

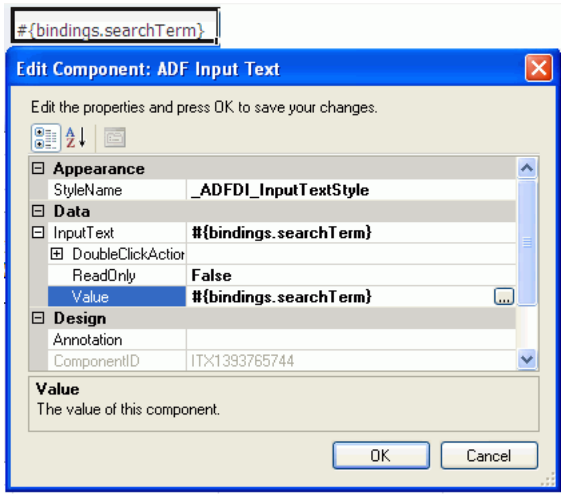
2. Query for the value.
3. Download the results to an ADF Table or ADF Read-only Table component in the integrated Excel workbook.

To create a simple search form in an integrated Excel workbook:

1. Open the integrated Excel workbook.
2. Insert an ADF Input Text component in the Excel worksheet cell where you want the end user to enter the search criteria.
3. Configure the ADF Input Text component so that it assigns the search term, that a user enters, to an attribute binding.

   Figure 8–19 shows an example from the `EditPriceList-DT.xlsx` workbook in the Master Price List module where an ADF Input Text component assigns the user-entered value to the `searchTerm` attribute binding. The `searchTerm`, which is a part of variable iterator, is then passed as a `NamedData` argument to the `executeSimpleProductQuery` method.

Figure 8–19 ADF Input Text Component for a Simple Search Form


#{bindings.searchTerm}

Edit Component: ADF Input Text

Edit the properties and press OK to save your changes.

Appearance	
StyleName	**_ADFDI_InputTextStyle**
Data	
InputText	**#{bindings.searchTerm}**
DoubleClickActior	
ReadOnly	**False**
Value	**#{bindings.searchTerm}**
Design	
Annotation	
ComponentID	ITX1393765744

Value
The value of this component.

4. Optionally, apply a style to the ADF Input Text component to indicate to end users that they can enter a search term in the cell.
5. Optionally, create an ADF Label component in an adjoining cell to indicate to end users that they can enter a search term in the ADF Input Text component you created in Step 2.
6. Create an ADF Button component in the Excel worksheet.
7. Set the `Label` property of the ADF Button component so that it displays a string at runtime to indicate to end users that they can start a search operation by clicking the button.
8. Open the Action Collection Editor to configure the array of actions (`Action[]Array`) in the `ClickActionSet` properties of the ADF Button component. Table 8–5 describes the actions to invoke in sequence.

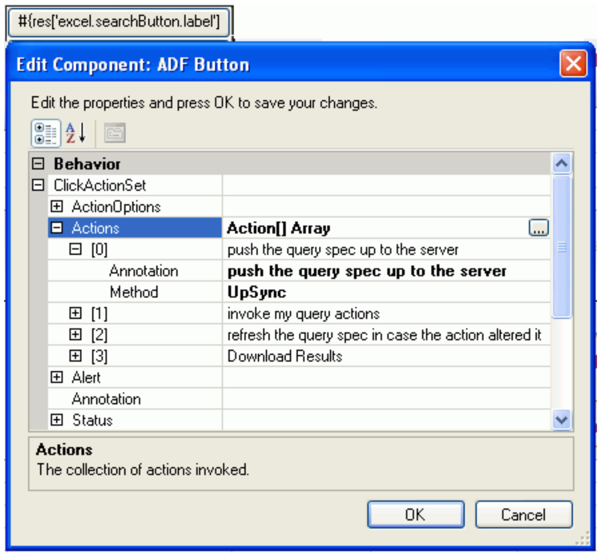
Table 8–5 ClickActionSet Properties of the ADF Button Component

Add this action...	To...
`Worksheet`	Invoke the `UpSync` worksheet action to copy the value entered in the cell that hosts an ADF Input Text or ADF List of Values component to the Fusion web application. For more information about worksheet actions, see Section A.13, "Worksheet Actions and Properties."
`ADFmAction`	Invoke an ADF Model action that is bound to the attribute binding you specified in Step 3. The ADF Model action queries for the end user's search term value referenced by the attribute binding. The corresponding example in the `EditPriceList-DT.xlsx` workbook is the `executeSimpleProductQuery` action binding, which is bound to the `searchTerm` attribute binding.
`Worksheet`	Invoke the `DownSync` worksheet action to synchronize any pending changes from the Fusion web application to the ADF Input Text, ADF Output Text, and ADF List of Values components in the worksheet. For more information about worksheet actions, see Section A.13, "Worksheet Actions and Properties."
`Component`	Invoke a `Download` action from the ADF Table or ADF Read-only Table components to download the results that the ADF Model action retrieved.

9. Click **OK**.

Figure 8–20 shows an example from the `EditPriceList-DT.xlsx` workbook in the Master Price List module where an ADF Button component invokes the `executeSimpleProductQuery` action binding using the search term the end user entered in the ADF Input Text component.

Figure 8–20 ADF Button Component for Simple Search Form


#{res['excel.searchButton.label']}

Edit Component: ADF Button

Edit the properties and press OK to save your changes.

Behavior	
ClickActionSet	
ActionOptions	
Actions	**Action[] Array**
[0]	push the query spec up to the server
Annotation	**push the query spec up to the server**
Method	**UpSync**
[1]	invoke my query actions
[2]	refresh the query spec in case the action altered it
[3]	Download Results
Alert	
Annotation	
Status	

Actions
The collection of actions invoked.

8.6.2 How to Create an Advanced Search Form in an Integrated Excel Workbook

You use the ADF Button component to invoke a page from the Fusion web application that displays a search form to the end user. Configure the action set for the ADF

Button component to invoke the `Download` action for the ADF Table or ADF Read-only Table component so that the search results from the search operation are downloaded to the integrated Excel workbook.

For information about creating a search form in a Fusion web application, see the "Creating ADF Databound Search Forms" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

To invoke an advanced search form in an integrated Excel workbook:

1. Open the integrated Excel workbook.
2. Create an ADF Button component in the Excel worksheet.
3. Set the `Label` property of the ADF Button component so that it displays a string at runtime to indicate to end users that they can start a search operation by clicking the button.
4. Use the Action Collection Editor to configure the array of actions (`Action[]Array`) in the `ClickActionSet` properties of the ADF Button component. Table 8–6 describes the actions to invoke in sequence.

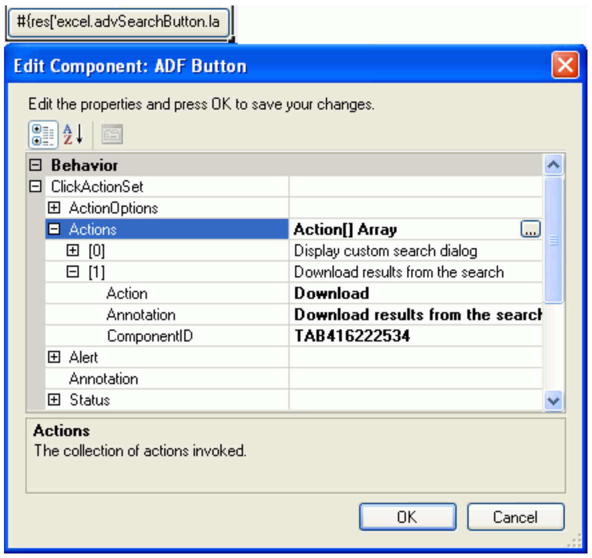
Table 8–6 Actions to Invoke an Advanced Search Form

Add this action...	To...
`Dialog`	Display the page from your Fusion web application that contains the search form. For more information about displaying pages from a Fusion web application, see Section 8.4, "Displaying Web Pages from a Fusion Web Application."
`Component`	Invoke a `Download` action from the ADF Table or ADF Read-only Table components to download the results that the ADF Model action retrieved.

5. Click **OK**.

   Figure 8–21 shows an example from the `EditPriceList-DT.xlsx` workbook in the Master Price List module where an ADF Button component invokes the `Execute` action binding to retrieve the values specified by the end user in the Master Price List's module Search page (`excelAdvSearch.jspx`). The ADF Table component's `Download` action downloads the returned values to the integrated Excel workbook.

Figure 8–21 ADF Button Component for an Advanced Search Form



#{res['excel.advSearchButton.la

Edit Component: ADF Button

Edit the properties and press OK to save your changes.

Behavior	
ClickActionSet	
ActionOptions	
Actions	**Action[] Array**
[0]	Display custom search dialog
[1]	Download results from the search
Action	**Download**
Annotation	**Download results from the searcl**
ComponentID	**TAB416222534**
Alert	
Annotation	
Status	

Actions
The collection of actions invoked.

8.7 Adding a Form to an Integrated Excel Workbook

You can use the ADF Desktop Integration components described in Chapter 6, "Working with ADF Desktop Integration Form-Type Components," to create forms in your integrated Excel workbook. These components can be useful when you want to provide end users with functionality that allows them to view and edit individual fields rather than use the functionality provided by the table-type components to download rows of data from the Fusion web application. Use one or more of the following components to create a form:

- ADF Button

  Use this component to provide end users with a button that can invoke a `ClickActionSet`. Figure 8–22 shows an ADF Button labeled **Search** that invokes a search operation using the search term entered by the end user in the ADF Input Text component.

- ADF Input Text

  Use this component to provide end users with a read/write field where the current value of a binding appears. This component can also be used to input a value, as in the example illustrated in Figure 8–22, where users enter a search term in the ADF Input Text component.

- ADF Output Text

  Use this component to provide end users with a read-only field where the current value of a binding appears.

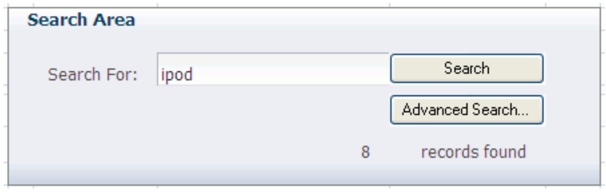
- ADF List of Values

  Use this component to provide end users with a dropdown menu from which a user can select a value from a list binding.

- ADF Label

  Use this component to provide end users with instructions or other information on how to use the form you create. For example, the Master Price List module's `EditPriceList-DT.xlsx` workbook uses ADF Label components to display an instruction to end users and the number of matches for a search term. Figure 8–22 shows the runtime values of these components. The text `Search For:` is a label instructing end uses to enter the search string, and `8 records found` label displays the number of records found matching the search string.

Figure 8–22 Runtime View of a Form in an Integrated Excel Workbook



You use the ADF Desktop Integration task pane to insert the components you require into a worksheet.

To create a form in an integrated Excel workbook:

1. Decide which ADF form components you require for the finalized form and insert them in the Excel worksheet.

   For more information about these components, see Chapter 6, "Working with ADF Desktop Integration Form-Type Components."

2. Configure the layout and appearance of the components you insert.

   For more information about configuring the appearance of components, see Chapter 9, "Configuring the Appearance of an Integrated Excel Workbook."

3. Test your form.

   For more information about testing an integrated Excel workbook, see Chapter 13, "Testing Your Integrated Excel Workbook."

8.8 Creating Dependent Lists of Values in an Integrated Excel Workbook

ADF Desktop Integration provides the following components that you use to create lists of values in an integrated Excel workbook:

- ADF List of Values

  You configure properties for this component when you want to create a list of values in the Excel worksheet.

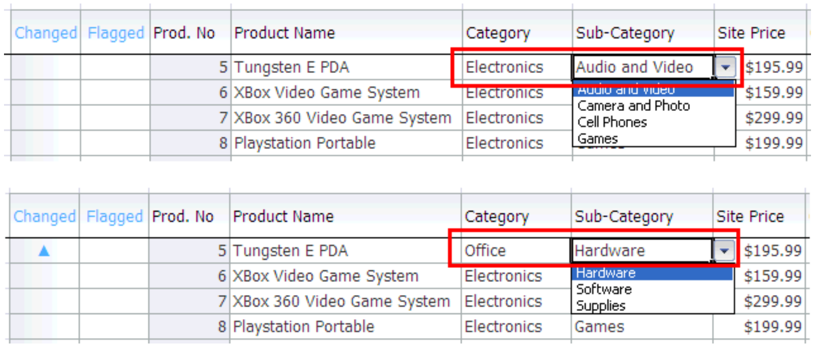
- TreeNodeList subcomponent

  You configure properties for this component when you want to create a list of values in an ADF Table component column.

Using these two components, you can create a dependent list of values in your integrated Excel workbook. A *dependent list of values* is a list of values component (referred to as a *child list of values*) whose values are determined by another list of values component (referred to as a *parent list of values*).

The server-side list bindings must be defined such that when the selected item of the parent list of values is changed, the available child list of values items are updated properly. Figure 8–23 shows an example with two illustrations from the `AdvEditPriceList-DT.xlsx` file of Master Price List module, where the **Sub-Category** column (child list of values) changes when the value in the **Category** column (parent list of values) changes.

Figure 8–23 List of Values and Dependent List of Values in Master Price List Module



Changed	Flagged	Prod. No	Product Name	Category	Sub-Category	Site Price
		5	Tungsten E PDA	Electronics	Audio and Video	$195.99
		6	XBox Video Game System	Electronics		$159.99
		7	XBox 360 Video Game System	Electronics		$299.99
		8	Playstation Portable	Electronics		$199.99

Changed	Flagged	Prod. No	Product Name	Category	Sub-Category	Site Price
▲		5	Tungsten E PDA	Office	Hardware	$195.99
		6	XBox Video Game System	Electronics		$159.99
		7	XBox 360 Video Game System	Electronics		$299.99
		8	Playstation Portable	Electronics	Games	$199.99

Table 8–7 describes the dependent list of values implementations you can create using the previously listed components and the requirements to achieve each implementation.

Some of the implementations described in Table 8–7 require model-driven lists. For information about creating a model-driven list, see the "How to Create a Model-Driven List" section of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Table 8–7 Dependent List of Values Configuration Options

Configuration	Requirements
Render both the parent and child list of values in the Excel worksheet using ADF List of Values components.	Both instances of the ADF List of Values component must reference a list binding. One or both of the list bindings that you reference can be model-driven lists. Both list bindings can reference model-driven lists only if the underlying iterator has at least one row of data. At runtime, if the underlying iterator has zero rows of data and the end user selects a value from the parent list of values (list binding referenced by the ADF List of Values component's `DependsOnListID` property), the child list of values (list binding referenced by the ADF List of Values component's `ListID` property) does not get filtered based on the value the end user selects. To work around this scenario, choose one of the following options: ■ Ensure that the underlying iterator has at least one row of data ■ Use an alternative list binding configuration where you expose multiple iterators and all necessary iterators get refreshed For more information, see Section 8.8.1, "How to Create a Dependent List of Values in an Excel Worksheet."
Render both the parent and child list of values in ADF Table component columns using TreeNodeList subcomponents.	Both the parent and child list of values (TreeNodeList subcomponents) must reference tree binding attributes associated with model-driven lists. For more information, see Section 8.8.3, "How to Create a Dependent List of Values in an ADF Table Component's Columns."
Render the parent list of values in an ADF List of Values component and the child list of values in an ADF Table component column using the TreeNodeList subcomponent.	The child list of values (TreeNodeList subcomponent) must reference a tree binding attribute associated with a model-driven list. The parent list of values (ADF List of Values component) must reference a list binding. For more information, see Section 8.8.5, "Creating a Dependent List of Values in an Excel Worksheet and an ADF Table Component Column."

Note the following points if you plan to create a dependent list of values:

- When the cell value referenced by `DependsOnList` or `DependsOnListID` is changed, ADF Desktop Integration overrides any previous changes to the child component list of values without warning the end user.
- The dependent list of values does not work unless the list specified in the `DependsOnList` (or `DependsOnListID`) property is referenced by a component in the Excel worksheet.
- If a circular dependency is defined (List A depends on List B, and List B depends on List A), the first dependency (List A depends on List B) triggers the expected behavior. ADF Desktop Integration considers other dependencies to be misconfigurations.
- You can create a chain of dependencies as follows:
  - List A depends on List B
  - List B depends on List C

  In this scenario, a change in List C (grandparent list of values) updates both Lists A (grandchild list of values) and B (child list of values). If you create a similar scenario, you must ensure that both the grandchild list of values and the child list of values, get refreshed whenever the parent list of values selection is changed. You can do this by specifying the two bind variables on the grandchild list of values to set up an implicit dependency between the view attributes. Another way is to declare explicit attribute dependencies between each of the view attributes that have model-driven lists configured. For example, specify that attribute A depends on attribute B and attribute C, and attribute B depends on attribute C.
- Caching in a dependent list of values is discussed in Section 15.3, "Caching Lists of Values for Use in Disconnected Mode."
- ADF Desktop Integration caches the values that appear in a dependent list of values. Hence, the dependent list item values for a given parent list selection must remain constant across all rows of an ADF Table component.

8.8.1 How to Create a Dependent List of Values in an Excel Worksheet

Use two instances of the ADF List of Values component to create a dependent list of values in an Excel worksheet.

Specify the list binding referenced by the parent ADF List of Values component as a value for the child ADF List of Values component’s `ListOfValues.DependsOnListID` property.

For more information about ADF List of Values, see Section A.5, "ADF List of Values Component Properties."

To create a dependent list of values in an Excel worksheet:

1. If not present, add the required list bindings to your page definition file.

   For more information about adding bindings to page definition files, see Section 4.3, "Working with Page Definition Files for an Integrated Excel Workbook."
2. Open the integrated Excel workbook.
3. Insert two ADF List of Values components into your integrated Excel workbook, as described in Section 6.6, "Inserting an ADF List of Values Component."

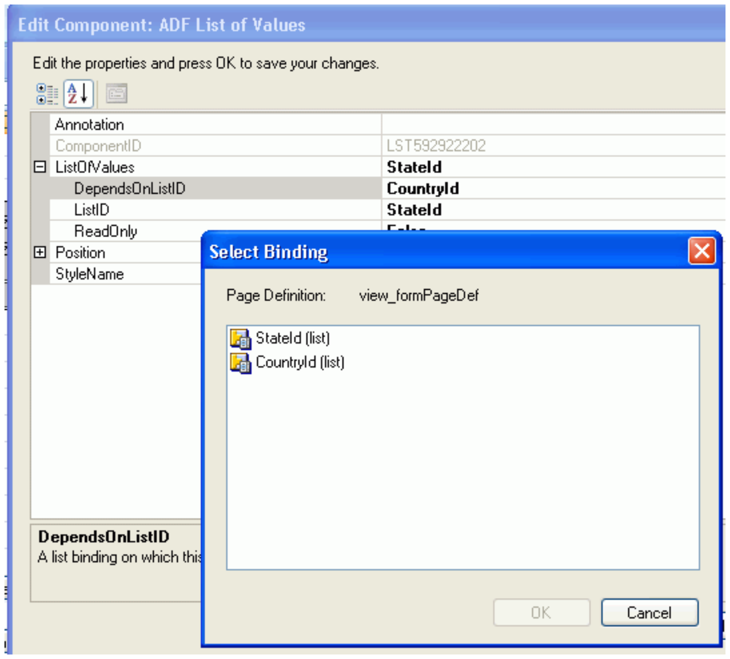
4. Display the property inspector for the ADF List of Values component that is to serve as the parent in the dependent list of values and set the value of the `ListOfValues.ListID` property to the list binding that is the parent.
5. Display the property inspector for the ADF List of Values component that is to serve as the child in the dependent list of values and set its values as follows:
   - `ListOfValues.ListID`

     Specify the list binding that is the child in the dependent list of values.
   - `ListOfValues.DependsOnListID`

     Select the list binding that you specified for the ADF List of Values component that serves as a parent in Step 4.

     Figure 8–24 shows the property inspector for the child ADF List of Values where the `CountryId` list binding is specified as the parent list of values and `StateId` list is the dependent list of values.

Figure 8–24 Design Time Dependent List of Values in an Excel Worksheet


Edit Component: ADF List of Values

Edit the properties and press OK to save your changes.

Annotation	
ComponentID	LST592922202
ListOfValues	**StateId**
DependsOnListID	**CountryId**
ListID	**StateId**
ReadOnly	
Position	
StyleName	

DependsOnListID
A list binding on which this

Select Binding

Page Definition: view_formPageDef

StateId (list)
CountryId (list)

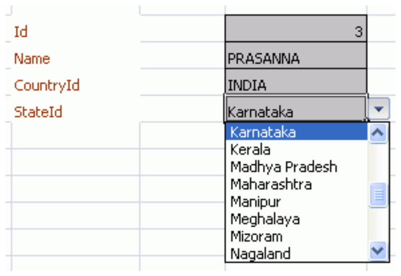
6. Click **OK**.

8.8.2 What Happens at Runtime When a Dependent List of Values Renders in an Excel Worksheet

At runtime, ADF Desktop Integration renders both instances of the ADF List of Values component. When the end user selects a value from the parent list of values, the selected value determines the list of values in the child list.

Figure 8–25 shows an example where **StateID**, a dependent list value, displays only the states from the selected **CountryId** list value.

Figure 8–25 Runtime Dependent List of Values in an Excel Worksheet



8.8.3 How to Create a Dependent List of Values in an ADF Table Component's Columns

Use instances of the TreeNodeList subcomponent to render both lists of values in a dependent list of values in ADF Table component columns at runtime.

Specify a tree binding attribute as a value for the parent TreeNodeList subcomponent's `List` property. You also specify a tree binding attribute as a value for the child TreeNodeList subcomponent's `List` property and the same tree binding attribute referenced by the parent TreeNodeList subcomponent as a value for its `DependsOnList` property.

Ensure that both tree binding attributes are associated with model-driven lists before you add the tree binding to your page definition file. For information about creating a model-driven list, see the "How to Create a Model-Driven List" section of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. For information about adding a tree binding to your page definition file, see Section 4.3, "Working with Page Definition Files for an Integrated Excel Workbook."

For information about the TreeNodeList subcomponent, see Section A.6, "TreeNodeList Subcomponent Properties."

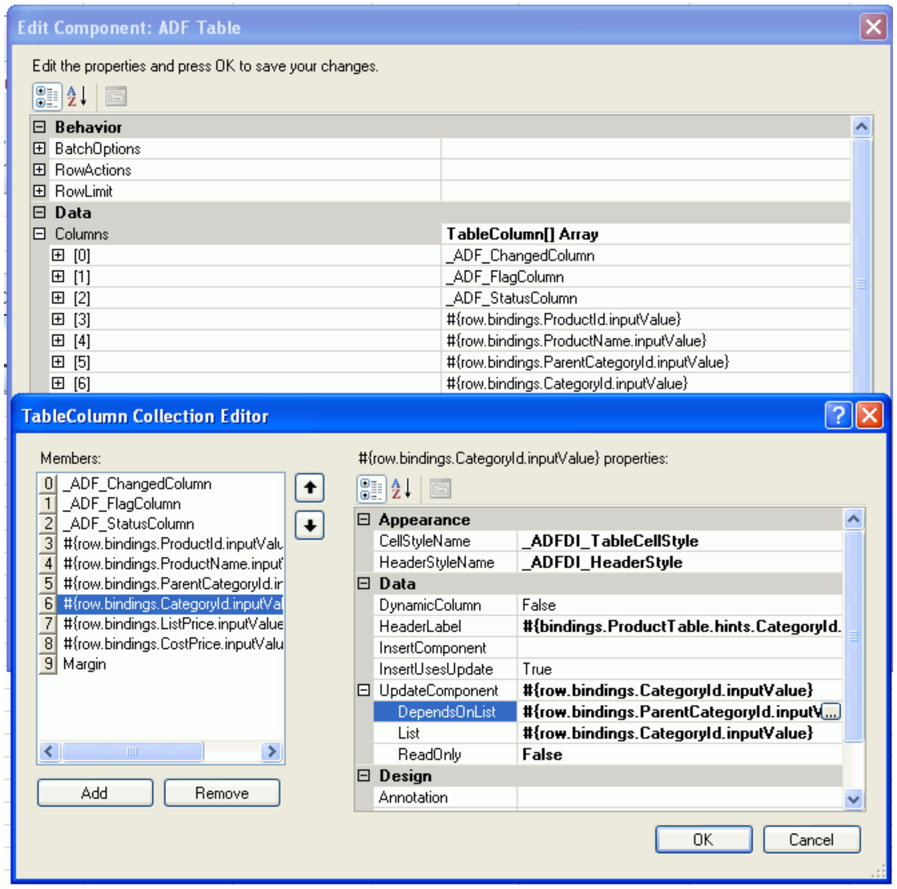
To create a dependent list of values in an ADF Table component:

1. Open the integrated Excel workbook.
2. If not present, insert an ADF Table component.

   For more information, see Section 7.3, "Inserting an ADF Table Component into an Excel Worksheet."
3. Display the property inspector for the ADF Table component and invoke the TableColumn Collection Editor by clicking the ellipsis button (**...**) beside the input field for **TableColumn[] Array**.
4. If not created, click **Add** to add a new column to serve as the parent list of values. For more information about creating a list of values, see Section 7.13, "Creating a List of Values in an ADF Table Component Column."
5. Add a new column to the ADF Table component to serve as the child list of values in the runtime-dependent list of values. For more information about creating a list of values, see Section 7.13, "Creating a List of Values in an ADF Table Component Column."
6. Specify the tree binding attribute of the parent list of values as a value for the `DependsOnList` property.

   Figure 8–26 shows the property inspector for a child ADF Desktop Integration Tree Node component, where the `ParentCategoryId` tree binding attribute is specified as the parent list of values.

Figure 8–26 Design Time Dependent List of Values in an ADF Table Component's Columns



Edit Component: ADF Table

Edit the properties and press OK to save your changes.

Behavior	
BatchOptions	
RowActions	
RowLimit	
Data	
Columns	**TableColumn[] Array**
[0]	_ADF_ChangedColumn
[1]	_ADF_FlagColumn
[2]	_ADF_StatusColumn
[3]	#{row.bindings.ProductId.inputValue}
[4]	#{row.bindings.ProductName.inputValue}
[5]	#{row.bindings.ParentCategoryId.inputValue}
[6]	#{row.bindings.CategoryId.inputValue}

TableColumn Collection Editor

Members:

0 _ADF_ChangedColumn
1 _ADF_FlagColumn
2 _ADF_StatusColumn
3 #{row.bindings.ProductId.inputValu
4 #{row.bindings.ProductName.input
5 #{row.bindings.ParentCategoryId.ir
6 #{row.bindings.CategoryId.inputVal
7 #{row.bindings.ListPrice.inputValue
8 #{row.bindings.CostPrice.inputValu
9 Margin

#{row.bindings.CategoryId.inputValue} properties:

Appearance	
CellStyleName	**_ADFDI_TableCellStyle**
HeaderStyleName	**_ADFDI_HeaderStyle**
Data	
DynamicColumn	False
HeaderLabel	**#{bindings.ProductTable.hints.CategoryId.**
InsertComponent	
InsertUsesUpdate	True
UpdateComponent	**#{row.bindings.CategoryId.inputValue}**
DependsOnList	**#{row.bindings.ParentCategoryId.inputV**
List	**#{row.bindings.CategoryId.inputValue}**
ReadOnly	**False**
Design	
Annotation	

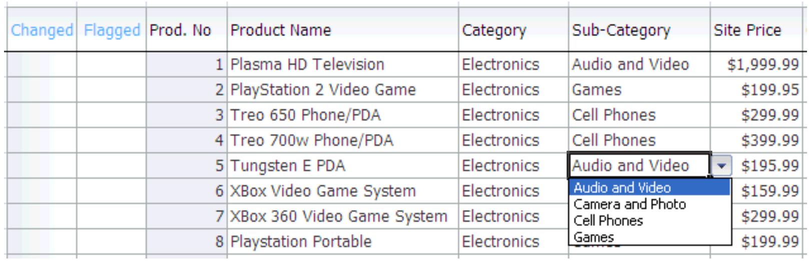
7. Click **OK**.

8.8.4 What Happens at Runtime When a Dependent List of Values Renders in an ADF Table Component’s Columns

At runtime, the ADF Table component renders both instances of the TreeNodeList subcomponent in the columns that you configured to display these instances. When the end user selects a value from the parent list of values, the selected value determines the list of values in the child list.

Figure 8–27 shows an example where the value that the end user selects in the **Category** column list of values results in the corresponding values for sub-category appearing in the **Sub-Category** column list of values.

Figure 8–27 Runtime Dependent List of Values in an ADF Table Component’s Columns



Changed	Flagged	Prod. No	Product Name	Category	Sub-Category	Site Price
		1	Plasma HD Television	Electronics	Audio and Video	$1,999.99
		2	PlayStation 2 Video Game	Electronics	Games	$199.95
		3	Treo 650 Phone/PDA	Electronics	Cell Phones	$299.99
		4	Treo 700w Phone/PDA	Electronics	Cell Phones	$399.99
		5	Tungsten E PDA	Electronics	Audio and Video	$195.99
		6	XBox Video Game System	Electronics		$159.99
		7	XBox 360 Video Game System	Electronics		$299.99
		8	Playstation Portable	Electronics		$199.99

Audio and Video
Camera and Photo
Cell Phones
Games

Note: If the child list and the parent list are bound to columns in the same ADF Table component, the child list items are changed for the current row only, when the end user changes the parent list selection.

8.8.5 Creating a Dependent List of Values in an Excel Worksheet and an ADF Table Component Column

Use an instance of the ADF List of Values component and an instance of the TreeNodeList subcomponent to create a dependent list of values where you render the parent and the child list of values.

- Parent list of values in the Excel worksheet

  An instance of the ADF List of Values component renders the parent list of values in the Excel worksheet.

- Child list of values in an ADF Table component column

  An instance of the TreeNodeList subcomponent renders the child list of values in the ADF Table component column.

Specify a list binding as a value for the parent ADF List of Values component's `ListID` property. You specify a tree binding attribute as a value for the child TreeNodeList subcomponent's `List` property, and the same list binding referenced by the parent ADF List of Values component as a value for its `DependsOnList` property.

Ensure that the tree binding attribute is associated with a model-driven list before you add the tree binding to your page definition file. For information about creating a model-driven list, see the "How to Create a Model-Driven List" section of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. For information about adding a list and tree binding to your page definition file, see Section 4.3, "Working with Page Definition Files for an Integrated Excel Workbook."

For more information about the ADF List of Values component, see Section A.5, "ADF List of Values Component Properties." For information about the TreeNodeList subcomponent, see Section A.6, "TreeNodeList Subcomponent Properties."

To create a dependent list of values in an Excel worksheet and an ADF Table component column:

1. Open the integrated Excel workbook.
2. Insert an ADF List of Values component into your integrated Excel workbook, as described in Section 6.6, "Inserting an ADF List of Values Component."
3. Display the property inspector for the ADF List of Values component and set the value of the `ListID` property to the list binding that is to serve as the parent list of values in the dependent list of values.
4. Click **OK**.
5. Display the property inspector for the ADF Table component and invoke the TableColumn Collection Editor by clicking the ellipsis button (**...**) beside the input field for **TableColumn[] Array**.
6. Click **Add** to add a new column to the ADF Table component to serve as the child list of values in the runtime-dependent list of values.
7. Choose the appropriate option for the newly created column:

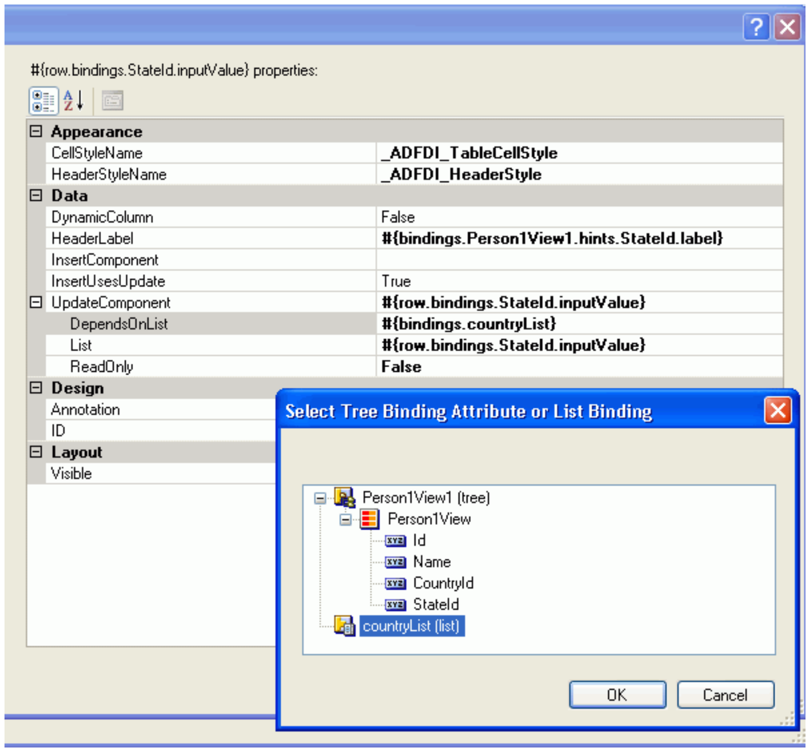
- Click the ellipsis button (**...**) beside the input field for **InsertComponent** to configure the runtime list of values for insert operations.
- Click the ellipsis button (**...**) beside the input field for **UpdateComponent** to configure the runtime list of values for update and download operations.

In both options, the Select subcomponent to create dialog appears.

8. Select **TreeNodeList** and click **OK**.
9. Expand the property that you selected in Step 7 and configure values as follows:
   - Select the same list binding that you specified as a value for the ADF List of Values component's `ListID` property in Step 3 as a value for the `DependsOnList` property.
   - Select a tree binding attribute associated with a model-driven list for the `List` property.
   - Configure the `ReadOnly` property as desired.
10. Click **OK**.

    Figure 8–28 shows the property inspector for a child ADF Desktop Integration Tree Node component where the `countryList` list binding is specified as the parent list of values.

Figure 8–28 Design Time Dependent List of Values in an Excel Worksheet and an ADF Table Component's Column



#{row.bindings.StateId.inputValue} properties:

Appearance	
CellStyleName	_ADFDI_TableCellStyle
HeaderStyleName	_ADFDI_HeaderStyle
Data	
DynamicColumn	False
HeaderLabel	#{bindings.Person1View1.hints.StateId.label}
InsertComponent	
InsertUsesUpdate	True
UpdateComponent	#{row.bindings.StateId.inputValue}
DependsOnList	#{bindings.countryList}
List	#{row.bindings.StateId.inputValue}
ReadOnly	False
Design	
Annotation	
ID	
Layout	
Visible	

Select Tree Binding Attribute or List Binding

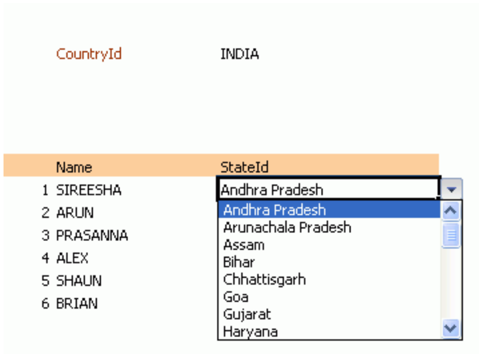
```
Person1View1 (tree)
  Person1View
    Id
    Name
    CountryId
    StateId
countryList (list)
```

8.8.6 What Happens at Runtime When a Dependent List of Values Renders in an Excel Worksheet and an ADF Table Component Column

At runtime, the ADF List of Values component renders the parent list of values and the ADF Table component renders the child list of values in the column that you configured to display the TreeNodeList subcomponent. When the end user selects a value from the parent list of values, the selected value determines the list of values in the child list.

Figure 8–29 shows an example where the value that the end user selects in the **CountryId** list of values determines the list of values that appears in the **StateId** column of the ADF Table component.

Figure 8–29 Runtime-Dependent List of Values in an Excel Worksheet and an ADF Table Component's Column



CountryId INDIA

	Name	StateId
1	SIREESHA	Andhra Pradesh
2	ARUN	
3	PRASANNA	
4	ALEX	
5	SHAUN	
6	BRIAN	

Note: When the parent list is bound to a cell in the worksheet and the child list is bound to an ADF Table Component column, the child list items are updated for all rows in the table when the end user changes the parent list selection.

8.9 Using EL Expression to Generate an Excel Formula

You can use an EL expression to generate an Excel formula as the vlaue of an ADF component. For example, you can use an Excel `HYPERLINK` function in an EL expression. If you use the Excel `HYPERLINK` function in an EL expression, you must enclose the `HYPERLINK` function within an Excel `T` function if you want an Oracle ADF component, such as an ADF Output Text component, to display a hyperlink at runtime.

You enclose the `HYPERLINK` function because ADF Desktop Integration interprets the Excel formula. To work around this, you wrap the `T` function around the `HYERLINK` function so that the value of the `HYPERLINK` function is evaluated by the `T` function. The resulting value is inserted into the Excel cell that the ADF component references. Use the following syntax when writing an EL expression that invokes the `HYPERLINK` Excel function:

```
=T("=HYPERLINK(""link_location"",""friendly_name"")")
```

For example, the following EL expression uses `HYPERLINK` function to navigate to `http://www.oracle.com` when end user clicks the component.

```
=T("=HYPERLINK(""http://www.oracle.com"",
""#{bindings.ProductId.inputValue}"")")
```

If you write an EL expression using the `HYPERLINK` function, it is recommended that you select the **Locked** checkbox in the **Protection** tab of the Format Cells dialog for the custom style that you apply to prevent error messages appearing.

8.9.1 How to Configure a Cell to Display a Hyperlink Using EL Expression

You write an EL expression that uses the Excel `T` function to evaluate the output of the Excel `HYERLINK` function. The following task illustrates how you configure an ADF Output Text component to display a hyperlink that, when clicked, invokes a search operation on the Oracle OTN Discussion Forum for Developer Tools using the value of the `ProductName` binding as the search term.

To configure a cell to display a hyperlink using EL expression:

1. Open the integrated Excel workbook.
2. Insert an ADF Output Text component into the Excel worksheet.
3. Write an EL expression for the `Value` property of the ADF Output Text component.

   The EL expression that you write invokes the Excel `HYPERLINK` function and uses the Excel `T` function to evaluate the output. In our example, we entered the following EL expression for the `Value` property:

   ```
   =T("=HYPERLINK(""http://forums.oracle.com/forums/search.jspa?objID=c19&q=#{bind
   ings.ProductName}"", ""#{bindings.ProductName}"")")
   ```

   > **Note:** Excel requires that you write double quotes (for example, `""#{bindings.ProductName}""`) in the EL expression so that it can evaluate the expression correctly.

4. Click **OK**.

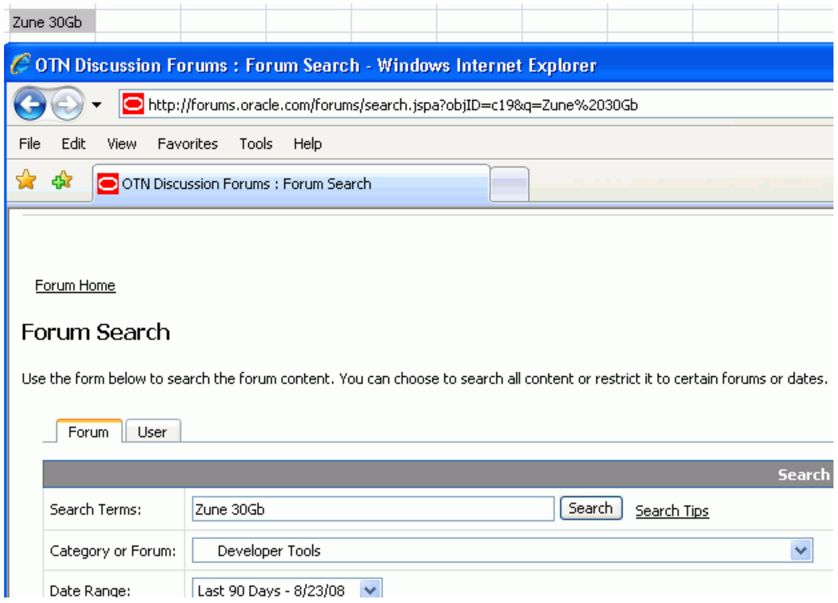
8.9.2 What Happens at Runtime When a Cell Displays a Hyperlink using EL Expression

ADF Desktop Integration evaluates the EL expression that you write at runtime. In the following example, ADF Desktop Integration:

- Retrieves the value of the `ProductName` binding
- Inserts the value of the `ProductName` binding into a URL
- Inserts the result into a hyerlinked cell that a user can click to invoke a search

Figure 8–30 shows the runtime view of the example configured in Section 8.9.1, "How to Configure a Cell to Display a Hyperlink Using EL Expression," where `Zune 30GB` is the retrieved value of the `ProductName` binding. When the end user clicks the cell that hosts the ADF Output Text component, he or she invokes a search operation for `Zune 30GB` on the Oracle OTN Discussion Forum for Developer Tools.

Figure 8–30 ADF Output Text Component Configured to Display a Hyperlink


Zune 30Gb

OTN Discussion Forums : Forum Search - Windows Internet Explorer

http://forums.oracle.com/forums/search.jspa?objID=c19&q=Zune%2030Gb

Forum Search

Use the form below to search the forum content. You can choose to search all content or restrict it to certain forums or dates.

Search Terms: Zune 30Gb

Category or Forum: Developer Tools

Date Range: Last 90 Days - 8/23/08

8.10 Using Calculated Cells in an Integrated Excel Workbook

You can write Excel formulas that perform calculations on values in an integrated Excel workbook. Before you write an Excel formula that calculates values in an integrated Excel workbook, note the following points:

- Formulas can be entered in cells that reference Oracle ADF bindings and cells that do not reference Oracle ADF bindings
- End users of an integrated Excel workbook can enter formulas at runtime
- You (developer of the integrated Excel workbook) can enter formulas at design time
- During invocation, the ADF Table component actions `Upload` and `RowUpSync` send the results of a formula calculation to the Fusion web application and not the formula itself
- Excel recalculates formulas in cells that reference Oracle ADF bindings when these cells are modified by:
  - Invocation of the ADF Table component `RowDownSync` and `Download` actions
  - Rendering of Oracle ADF components
- The ADF Table and ADF Read-only Table components insert or remove rows as they expand or contract to accommodate data downloaded from the Fusion web application. Formulas are replicated according to Excel’s own rules.
- You can enter formulas above or below a cell that references an ADF Table or ADF Read-only Table component. A formula that you enter below one of these components maintains its position relative to the component as the component expands or contracts to accommodate the number of rows displayed.

For more information about Excel functions, see the Function reference section in Excel’s online help documentation.

8.10.1 How to Create a Column That Displays Values Generated by an Excel Formula

You insert a column that displays values calculated by an Excel formula directly into a worksheet using the menu options on Excel's Ribbon. You cannot add a column that displays calculated values using the collection editor that manages columns for an ADF Table or ADF Read-only Table component.

To create a column that displays values generated by an Excel formula:

1. In design mode in the Excel worksheet, select the cell in which you want the column that displays the values generated by the Excel formula to appear at runtime.

   For example, the H13 cell of `EditPriceList-DT.xlsx` contains a formula:

   ```
   =G13-F13
   ```

   Cell `G13` is the design time reference for the ADF Table component column labeled **List Price** at runtime, and `F13` is the design time cell reference for the ADF Table component column labeled **Cost Price** at runtime.

   The `H12` cell marks the header for the formula. It contains an ADF Label component with its `Label` property set to the following EL expression:

   ```
   #{res['excel.difference.label']}
   ```

   The EL expression retrieves the value of the `excel.difference.label` string key at runtime.

   Figure 8–31 shows the design time view of the manually inserted column, with the Excel formula appearing in the formula bar, and the ADF Label component that retrieves the string key value from the resource bundle at runtime.

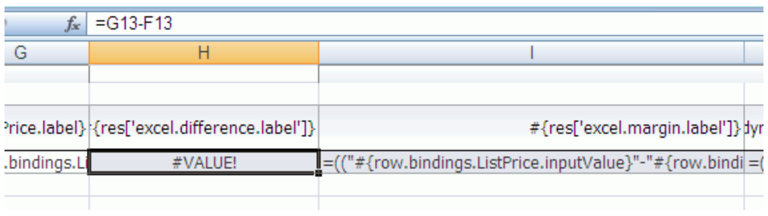


Figure 8–31 Design Time View of Column That Displays Values Generated by an Excel Formula

=G13-F13

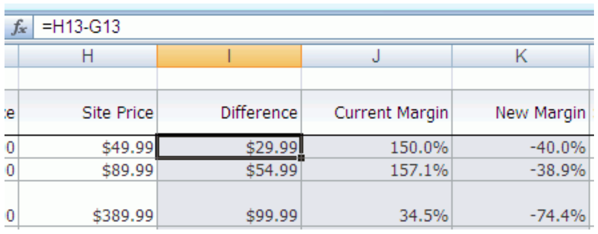
G	H	I
›rice.label}	{res['excel.difference.label']}	#{res['excel.margin.label']}
.bindings.Li	#VALUE!	=(("#{row.bindings.ListPrice.inputValue}"-"#{row.bindi

2. Save your changes using Excel's **Save** button.

8.10.2 What Happens at Runtime When a Column Displays Values Generated by an Excel Formula

At runtime, Excel replicates and adjusts its formula as the ADF Table and ADF Read-only components expand or contract so that the correct value appears in each row of a manually inserted column. Figure 8–32 shows an extract of the runtime view of the example that appears in Figure 8–31 where Excel adjusted the formula so that it evaluates each corresponding row.

Figure 8–32 Runtime View of Column That Displays Values Generated by an Excel Formula



=H13-G13

	H	I	J	K
e	Site Price	Difference	Current Margin	New Margin
0	$49.99	$29.99	150.0%	-40.0%
0	$89.99	$54.99	157.1%	-38.9%
0	$389.99	$99.99	34.5%	-74.4%

8.10.3 How to Calculate the Sum of a Table-Type Component Column

The following task illustrates how you use the Excel functions `SUM` and `OFFSET` to calculate the total of the column labeled **Difference** in the `EditPriceList-DT.xlsx` of the Master Price List module at runtime. You use the `OFFSET` function in an Excel formula that you write where you want to reference a range of cells that expands or contracts based on the number of rows that an ADF Table or ADF Read-only Table component downloads. The `SUM` function calculates the total in a range of Excel cells.

To calculate the sum of a column in an ADF Table component:

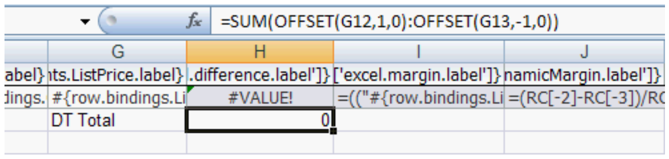
1. In design mode in the Excel worksheet, select the cell in which you want to write the Excel formula. In `EditPriceList-DT.xlsx`, this is the cell with the reference, `H14`.
2. Write the Excel formula that performs a calculation on a range of cells at runtime. For example:

   ```
   =SUM(OFFSET(G12,1,0):OFFSET(G13,-1,0))
   ```

   where `SUM` calculates the total of values in the range of cells currently referenced by `G12` and `G13`.

   Figure 8–33 shows the design time view of the Excel formula in the integrated Excel workbook.

Figure 8–33 Design Time View of Excel Formula in an Integrated Excel Workbook



=SUM(OFFSET(G12,1,0):OFFSET(G13,-1,0))

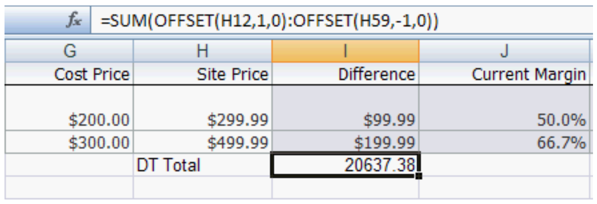
	G	H	I	J
abel'}	its.ListPrice.label'}	.difference.label']}	['excel.margin.label']}	namicMargin.label']}
dings.	#{row.bindings.Li	#VALUE!	=(("#{row.bindings.Li	=(RC[-2]-RC[-3])/RC
	DT Total	0		

3. Save your changes and switch to runtime mode to test that the Excel formula you entered evaluates correctly.

8.10.4 What Happens at Runtime When Excel Calculates the Sum of a Table-Type Component Column

Figure 8–34 shows the runtime view in the integrated Excel workbook when the Excel formula shown in Figure 8–33 is evaluated. The Excel formula calculates the total of the values in the range of cells that you specified in design mode. The cell references that appear in Excel's formula bar at runtime (`H12` and `H59`) differ from those that appear in the formula bar at design time (`G12` and `G13`) because the ADF Table component has moved and expanded to include the rows of data that it downloads.

Figure 8–34 Runtime View of Excel Formula in an Integrated Excel Workbook


=SUM(OFFSET(H12,1,0):OFFSET(H59,-1,0))

G	H	I	J
Cost Price	Site Price	Difference	Current Margin
$200.00	$299.99	$99.99	50.0%
$300.00	$499.99	$199.99	66.7%
	DT Total	20637.38	

8.11 Using Macros in an Integrated Excel Workbook

You can define and execute macros based on Excel events in an integrated Excel workbook.

Note the following points:

- Macros triggered by an Excel event do not get triggered if the Excel event is invoked by ADF Desktop Integration.
- ADF Desktop Integration code invoked by an Excel event is executed when the Excel event is triggered by a macro.

9

Configuring the Appearance of an Integrated Excel Workbook

This chapter describes how you configure the appearance of an integrated Excel workbook using styles that ADF Desktop Integration defined and that you define in Excel. The chapter also discusses how you can use EL expressions to dynamically apply styles to Oracle ADF components in a workbook at runtime.

This chapter includes the following sections:

9.1 Introduction to Configuring the Appearance of an Integrated Excel Workbook

You can configure the appearance of an integrated Excel workbook using both Excel functionality and Oracle ADF functionality. Configuring the appearance of a workbook may make the workbook more usable for end users. For example, applying a particular style to cells that render ADF Output Text components at runtime may indicate to end users that the cell is read-only. You may also want to configure the appearance of an integrated Excel workbook so that it aligns with your company's style sheet or the color scheme of the Fusion web application that the Excel workbook integrates with.

ADF Desktop Integration provides several predefined Excel styles to apply to the ADF Desktop Integration components you configure in a workbook. You may want to define additional styles to meet the needs of your desktop integration project. If you do, familiarize yourself with the formats in an Excel workbook that render differently depending on the locale, region, and language.

Once you have decided what styles to apply to the ADF Desktop Integration components at runtime, you write EL expressions to associate a style with a component. The ADF Desktop Integration component properties that include *StyleName* in their name take an EL expression as a value. The ADF Label component and the `Label` property of other ADF components also support EL expressions. These EL expressions can retrieve the values of string keys defined in resource bundles or the values of attribute control hints defined in your Fusion web application.

Finally, in addition to styles that allow you to configure the appearance of an integrated Excel workbook, ADF Desktop Integration provides a collection of properties (`BrandingItems`) that enable you to brand your integrated Excel workbook with application name, application version details, and copyright information.

9.2 Working with Styles

ADF Desktop Integration provides a mechanism to apply Excel-defined styles to some Oracle ADF components at runtime. The Oracle ADF components that support the application of styles have properties with *StyleName* in their name. For example, the column properties of the ADF Table and ADF Read-only Table components both support the properties `HeaderStyleName` and `CellStyleName` that determine styles to apply at runtime.

Predefined Styles in ADF Desktop Integration

Many properties have default values that are drawn from a predefined list of ADF Desktop Integration module styles. For example, the `HeaderStyleName` property's default value is `_ADFDI_HeaderStyle`, one of the predefined styles in ADF Desktop Integration. ADF Desktop Integration automatically adds these predefined styles to the Excel workbook once when it is enabled for use with ADF Desktop Integration.

The following is the list of predefined styles:

- `_ADFDI_FormBottomStyle`
- `_ADFDI_FormDoubleClickCellStyle`
- `_ADFDI_FormTopStyle`
- `_ADFDI_HeaderStyle`
- `_ADFDI_InputTextStyle`
- `_ADFDI_LabelStyle`
- `_ADFDI_OutputTextStyle`
- `_ADFDI_ReadOnlyTableStyle`
- `_ADFDI_TableCellROStyle`
- `_ADFDI_TableCellStyle`
- `_ADFDI_TableChangedColumnStyle`
- `_ADFDI_TableDoubleClickCellStyle`
- `_ADFDI_TableFlagColumnStyle`
- `_ADFDI_TriangleHeaderStyle`

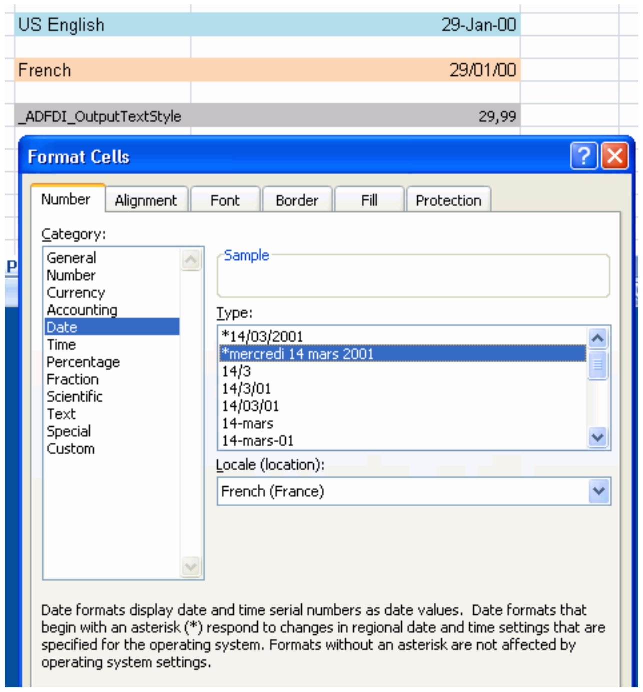
You can merge these styles and other styles that you define yourself from an integrated Excel workbook into another Excel workbook that you intend to integrate with a Fusion web application. You may create additional styles for use in your Excel workbook. For example, to add a date-specific formatting, you can duplicate `_ADFDI_TableCellStyle`, call it `MyTableCellDateStyle`, and add your date-specific formatting.

For more information about creating new styles and merging styles into a workbook, see Excel's documentation.

Excel's Date Formats and Microsoft Windows' Regional and Language Options

Some formats in the **Date** category of the **Number** styles that Excel can apply to cells change if a user changes the locale of the local system using the Regional and Language Options dialog that is accessible from the Microsoft Windows Control Panel. The * character precedes these formats in the **Type** list. Figure 9–1 shows an example of a Date type that formats dates in a cell using French (France) conventions.

Figure 9–1 Date Formats in Excel

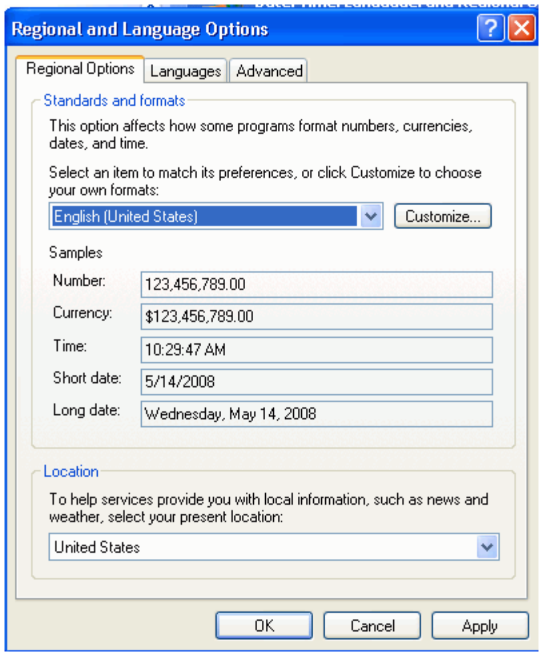


US English	29-Jan-00
French	29/01/00
_ADFDI_OutputTextStyle	29,99

Date formats display date and time serial numbers as date values. Date formats that begin with an asterisk (*) respond to changes in regional date and time settings that are specified for the operating system. Formats without an asterisk are not affected by operating system settings.

If the end user changes the regional options of a system to use **English (United States)**, as illustrated in Figure 9–2, the cells that are formatted with the style in Figure 9–1 use the **English (United States)** conventions.

Figure 9–2 Regional and Language Options in Excel


Regional and Language Options

Standards and formats

This option affects how some programs format numbers, currencies, dates, and time.

Select an item to match its preferences, or click Customize to choose your own formats:

English (United States)

Samples

Number:	123,456,789.00
Currency:	$123,456,789.00
Time:	10:29:47 AM
Short date:	5/14/2008
Long date:	Wednesday, May 14, 2008

Location

To help services provide you with local information, such as news and weather, select your present location:

United States

9.2.1 How to Apply a Style to an Oracle ADF Component

To apply a style to an Oracle ADF component, use the property inspector to set values for properties with *StyleName* in their name.

To apply a style:

1. In the integrated Excel workbook, select the cell that references the Oracle ADF component you want to modify and then click the **Edit Component** button in the **Oracle ADF** tab.

   For example, select a cell that references an ADF Table component.

2. Click **Columns** and then the value for the column in the array of columns where you want to modify the format of cells at runtime.

3. Select the **CellStyleName** property and click the ellipsis button (**...**) to display the Edit Expression dialog.

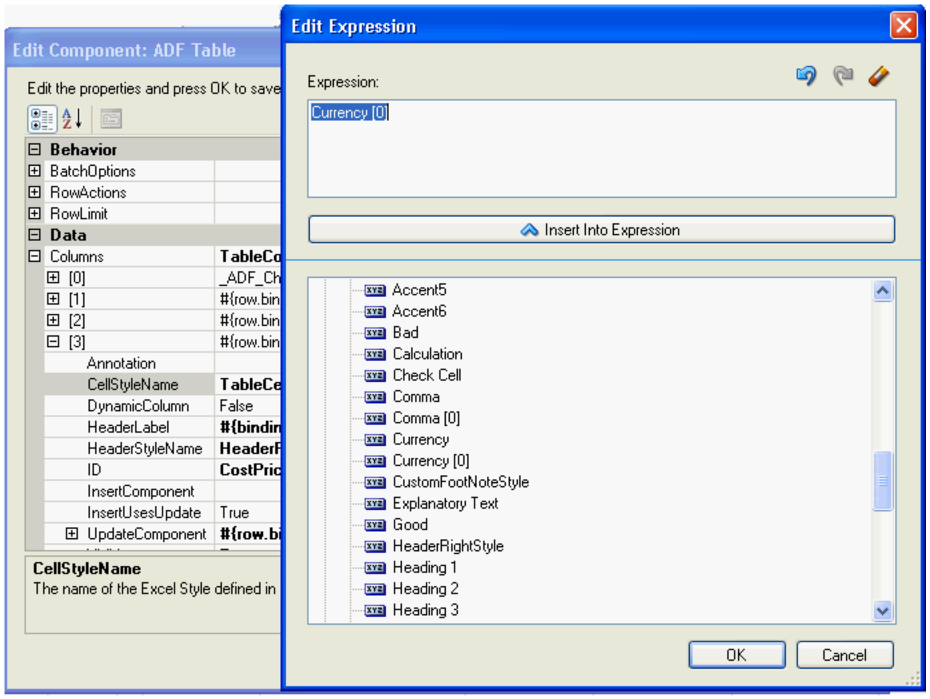
4. Expand the **Styles** node and select the style to apply to cells in the column at runtime.

   For example, apply a currency-based style (`Currency[0]` style) to the Cost Price databound column of Currency type. Applying the `Currency[0]` style rather than a general style to a databound column of Currency type results in runtime data (price values) appearing as values rounded off to zero decimal places rather than a regular value (with two decimal places).

5. Click **Insert Into Expression** to insert `Currency[0]` into the Expression field.

   Figure 9–3 shows the Edit Expression dialog.

Figure 9–3 Edit Expression Dialog Applying a Style



Edit Component: ADF Table

Edit the properties and press OK to save

Behavior	
BatchOptions	
RowActions	
RowLimit	
Data	
Columns	TableCo
[0]	_ADF_Ch
[1]	#{row.bin
[2]	#{row.bin
[3]	#{row.bin
Annotation	
CellStyleName	TableCe
DynamicColumn	False
HeaderLabel	#{bindin
HeaderStyleName	HeaderF
ID	CostPric
InsertComponent	
InsertUsesUpdate	True
UpdateComponent	#{row.bi

CellStyleName
The name of the Excel Style defined in

Edit Expression

Expression:

Currency [0]

Insert Into Expression

Accent5
Accent6
Bad
Calculation
Check Cell
Comma
Comma [0]
Currency
Currency [0]
CustomFootNoteStyle
Explanatory Text
Good
HeaderRightStyle
Heading 1
Heading 2
Heading 3

OK Cancel

6. Click **OK**.

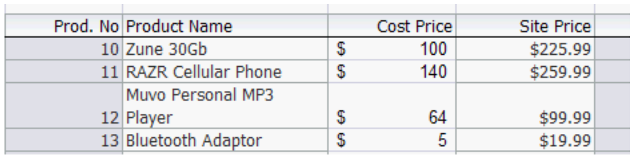
9.2.2 What Happens at Runtime When a Style Is Applied to an Oracle ADF Component

The EL expression that you entered as a value for the property with *StyleName* in its name is evaluated at runtime. If it corresponds to one of the predefined styles or one that you defined, the style is applied to the Oracle ADF component that you set the property for.

If a cell that references an Oracle ADF component has a style applied to it that differs from the style defined in the properties of the Oracle ADF component, the Oracle ADF component overwrites the existing style at runtime and applies the style defined by its properties.

For example, Figure 9–4 shows the runtime values of the Cost Price column after the `Currency[0]` style is applied, overriding the default `TableCellCurrency` style.

Figure 9–4 Runtime Values After Applying Another Style



Prod. No	Product Name	Cost Price		Site Price
10	Zune 30Gb	$	100	$225.99
11	RAZR Cellular Phone	$	140	$259.99
12	Muvo Personal MP3 Player	$	64	$99.99
13	Bluetooth Adaptor	$	5	$19.99

9.3 Applying Styles Dynamically Using EL Expressions

Oracle ADF component properties that include *StyleName* in their name can take an EL expression as a value. The EL expressions that you write can resolve to a named Excel style at runtime that is applied to the Oracle ADF component. The EL expressions that you write are Excel formulas that may include Oracle ADF data binding expressions.

ADF Desktop Integration does not evaluate or apply results when a user navigates between cells or during upload.

The following examples show different contexts where you can use EL expressions to determine the behavior and appearance of Oracle ADF components at runtime. Example 9–1 applies a style dynamically during download. If the status value for binding is `Closed`, apply a read-only style (`MyReadOnlyStyle`). Otherwise apply another style (`MyReadWriteStyle`).

Example 9–1 Applying a Style Dynamically During Download

```
=IF("#{bindings.Status}" = "Closed", "MyReadOnlyStyle", "MyReadWriteStyle")
```

Example 9–2 uses a mixture of Excel formulas and ADF binding expressions to handle errors and type conversion.

Example 9–2 EL Expressions to Handle Errors and Type Conversion

```
=IF(ISERROR(VALUE("#{bindings.DealSize}")), "BlackStyle",
IF(VALUE("#{bindings.DealSize}") > 300, "RedStyle", "BlackStyle"))
```

9.3.1 What Happens at Runtime When an EL Expression Is Evaluated

When evaluating EL expressions at runtime, ADF Desktop Integration determines the value that the EL expression references. It then replaces the EL expression in the Excel formula with the value. In Example 9–1, ADF Desktop Integration first determines that value of the EL expression, `#{bindings.Status}`, in the following Excel formula:

```
=IF("#{bindings.Status}" = "Closed", "MyReadOnlyStyle", "MyReadWriteStyle")
```

It then replaces the EL expression with the runtime value, as in the following example, where the expression evaluated to `Closed`:

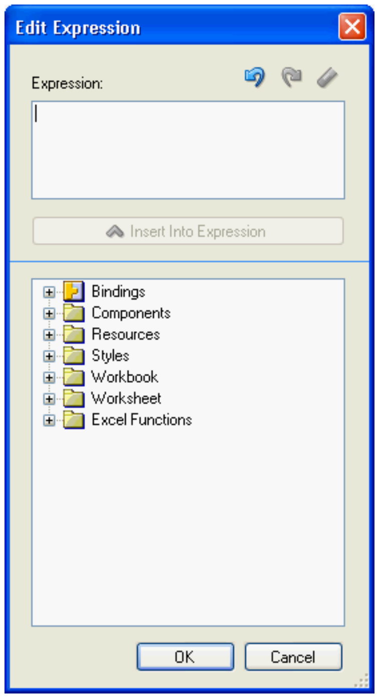
```
=IF("Closed" = "Closed", "MyReadOnlyStyle", "MyReadWriteStyle")
```

Excel evaluates the formula and, in this example, applies the `MyReadOnlyStyle` style.

9.3.2 How to Write an EL Expression That Applies a Style at Runtime

You write EL expressions for the Oracle ADF component properties that support EL expressions in the Edit Expression dialog that is accessible from the Oracle ADF component’s property inspector. Figure 9–5 displays an Edit Expression dialog launched from the property inspector window of an ADF Button component.

Figure 9–5 Edit Expression Dialog



To write an EL expression that applies a style at runtime:

1. Open the integrated Excel workbook.
2. Select a cell in the Excel worksheet that references the Oracle ADF component for which you want to write an EL expression.
3. Click the **Edit Component** button in the **Oracle ADF** tab to display the property inspector.
4. Select the property in the property inspector with which you want to associate an EL expression and click the ellipsis button (**...**) to display the Edit Expression dialog.

   Note: The Edit Expression dialog appears only if the Oracle ADF component that you selected in Step 2 supports EL expressions. Depending on the context, the ellipsis button (**...**) can launch other editors such as the Action Collection Editor.

   The Edit Expression dialog, as illustrated in Figure 9–5, displays a hierarchical list of the Oracle ADF components, bindings, styles, resources, and Excel functions that you can reference in EL expressions. For more information about the syntax of EL expressions that you enter in this dialog, see Appendix B, "ADF Desktop Integration EL Expressions."

9.3.3 What You May Need to Know About EL Expressions That Apply Styles

Note the following points when writing EL expressions that apply styles at runtime.

How the ADF Desktop Integration Applies Styles

EL expressions that evaluate to styles are applied when:

- An ADF Table component invokes its `Download` or `DownloadForInsert` actions
- Rows are inserted into an ADF Table component
- A worksheet invokes its `DownSync` action

EL expressions that evaluate to styles are not applied when:

- An ADF Table component invokes its `RowDownSync` action
- The end user edits the format properties of a cell

  Note also that an EL expression that evaluates to a style is not reevaluated when the end user edits a cell's value.
- The runtime value of an EL expression does not match a style defined in the end user's integrated Excel workbook

  In this scenario the style formats of the targeted cells do not change. Instead, they retain their existing style formats. If you configured client-side logging, ADF Desktop Integration generates an entry in the log file when an EL expression evaluates to a style that is not defined in the end user's integrated Excel workbook. For more information about client-side logging, see Section C.3, "Generating Log Files for an Integrated Excel Workbook."

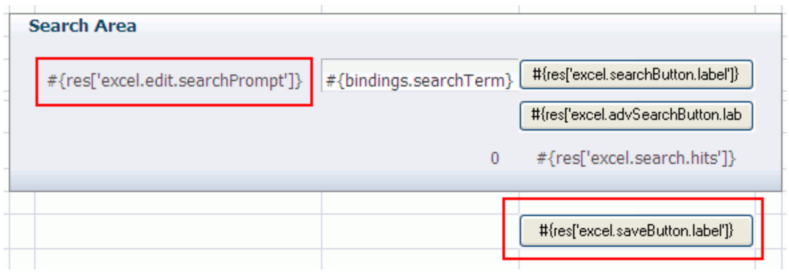
9.4 Using Labels in an Integrated Excel Workbook

Use labels to provide end users with information about how they use the functionality in an integrated Excel workbook. You can write EL expressions that retrieve the value of string keys defined in a resource bundle or that retrieve the values of attribute control hints. An integrated Excel workbook evaluates the value of a `Label` property only when the workbook is initialized.

Retrieving the Values of String Keys from a Resource Bundle

Figure 9–6 shows a portion of the design time view of the `EditPriceList-DT.xlsx` workbook in the Master Price List module. It shows examples of ADF Label components and ADF Button components that have EL expressions specified for their `Label` properties.

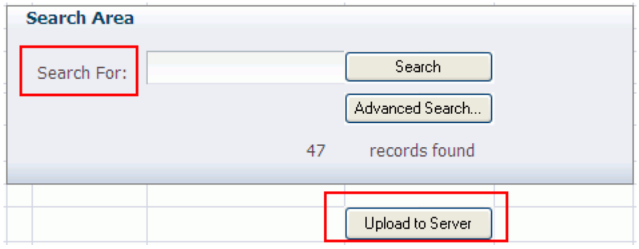
Figure 9–6 Design Time View of an ADF Label Component and an ADF Button Component with Label Property


Search Area
#{res['excel.edit.searchPrompt']}
#{bindings.searchTerm}
#{res['excel.searchButton.label']}
#{res['excel.advSearchButton.lab
0
#{res['excel.search.hits']}
#{res['excel.saveButton.label']}

At runtime, these EL expressions resolve to string keys defined in the `res` resource bundle that is registered with the Master Price List module. You define resource bundles in the workbook properties dialog. For information about referencing string keys from a resource bundle, see Section 10.2, "Using Resource Bundles in an Integrated Excel Workbook."

Figure 9–7 shows the corresponding runtime view of the ADF Label component and ADF Button component illustrated in design mode in Figure 9–6.

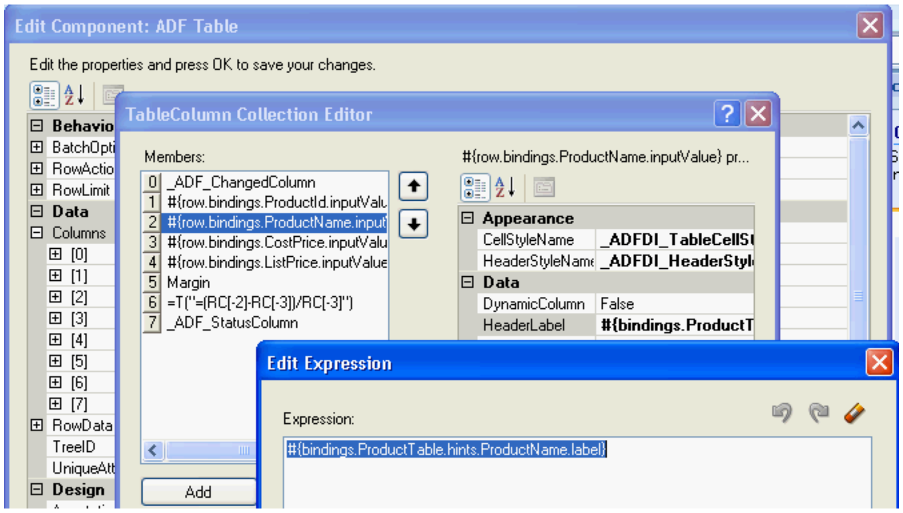
Figure 9–7 Runtime View of an ADF Label Component and an ADF Button Component with Label Property



Retrieving the Values of Attribute Control Hints

In addition to string keys from resource bundles, the ADF Label component and the `Label` property of other ADF components can reference attribute control hints that you define for entity objects and view objects in your JDeveloper project. Figure 9–8 shows the expression builder for the `Product Name` column in the `EditPriceList-DT.xlsx` workbook's ADF Table component. The expression builder contains an EL expression for the `HeaderLabel` property of the `ProductName` column that retrieves the value (`Product Name`) defined for an attribute control hint at runtime.

Figure 9–8 EL Expression That Retrieves the Value of an Attribute Control Hint for a Label Property



0	_ADF_ChangedColumn
1	#{row.bindings.ProductId.inputValu
2	#{row.bindings.ProductName.input
3	#{row.bindings.CostPrice.inputValu
4	#{row.bindings.ListPrice.inputValue
5	Margin
6	=T("=(RC[-2]-RC[-3])/RC[-3]")
7	_ADF_StatusColumn

#{row.bindings.ProductName.inputValue} pr...

Appearance	
CellStyleName	**_ADFDI_TableCellSt**
HeaderStyleName	**_ADFDI_HeaderStyl**
Data	
DynamicColumn	False
HeaderLabel	**#{bindings.ProductT**

```
#{bindings.ProductTable.hints.ProductName.label}
```

Attribute control hints can be configured for both view objects and entity objects. Information about how to add an attribute control hint to an entity object can be found in the "Defining Attribute Control Hints for Entity Objects" section of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. Information about how to add an attribute control hint to a view object can be found in the "Defining Attribute Control Hints for View Objects" section of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

How an Integrated Excel Workbook Evaluates a Label Property

An integrated Excel workbook evaluates the `Label` properties of ADF components when the workbook is initialized after you or the end user opens the workbook for the first time. The integrated Excel workbook saves the retrieved values for the `Label` properties when the workbook itself is saved to a directory on the system.

The retrieved values for the `Label` properties do not get refreshed during invocation of actions such as the worksheet's `DownSync` action or the ADF Table component's `Download` action. You indirectly refresh the retrieved values of the `Label` properties if you invoke the workbook actions `ClearAllData` or `EditOptions` described in Table A–17.

9.5 Using Styles to Improve the User Experience

It is good practice to provide end users of integrated Excel workbooks with information that helps them understand how to use the ADF components that you provide to integrate with a Fusion web application. You can do this by:

- Providing end users with instructions on how to use Oracle ADF components such as ADF Button and ADF Input Text components.

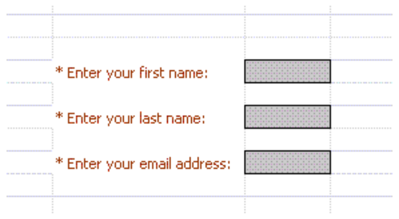
  The ADF Label component and the `Label` property of other ADF components is useful for this task, as you can write labels that instruct the end user on how to use the component.

- Apply styles that indicate if an ADF component is read-only or read-write.

Using ADF Label Components to improve the User Experience

You can use ADF Label components to provide end users of an integrated Excel workbook information about how to use other ADF components in the workbook. For example, many forms, by convention, use an * to indicate to end users that they must enter a value in an input field. Figure 9–9 shows three ADF Input Text components with ADF Label components in adjoining cells. Each ADF Label component references an EL expression that retrieves the value of a string key from a resource bundle at runtime. Each string key includes the * character to indicate to end users that they must supply a value.

Figure 9–9 ADF Label Components Providing End-User Instruction



For information about using resource bundles, see Section 10.2, "Using Resource Bundles in an Integrated Excel Workbook."

About the Read-Only Property in an Integrated Excel Workbook

Note the following points about the read-only property in an integrated Excel workbook:

- ADF Output Text, ADF Label, and ADF Table header row do not have read-only properties. However, these components have implied read-only behavior. In addition, end users can enter values in the cells that host these components and temporarily change the values that appear in these cells. ADF Desktop Integration ignores these changes when uploading from the worksheet and overwrites them when it downloads data from the Fusion web application.

- The ADF Input Text component, ADF List of Values component, and TreeNodeList subcomponent each have a read-only property (`ReadOnly`).

> **Note:** If you specify an Excel formula in the `Value` property of an ADF Input Text component, the component behaves as if its `ReadOnly` property were `True`. The component ignores the actual value of the `ReadOnly` property.

- To protect the values of read-only cells at runtime, set the worksheet protection to automatic. When an attempt is made to edit a read-only cell after enabling worksheet protection, Excel displays a warning message and the edit is blocked. For more information about worksheet protection, see Section 9.7, "Using Worksheet Protection."
- Do not use the Excel's Protect Sheet or Protect Workbook features directly in an integrated Excel workbook. Also, enure that end users do not also use these features.

To prevent end-user confusion, apply styles to components, such as the ADF Output Text component, that indicate to end users whether a component is read-only or can be edited. By default, the ADF Output Text component uses the predefined style, `_ADFDI_OutputTextStyle`. You can define your own styles and apply them to components as described in this chapter.

For more information about the properties that ADF Desktop Integration components support, see Appendix A, "ADF Desktop Integration Component Properties and Actions."

9.6 Branding Your Integrated Excel Workbook

ADF Desktop Integration provides several features that you can configure to brand your integrated Excel workbook with information such as application name, version information, and copyright information. You can use the workbook `BrandingItems` group of properties to associate this information with an integrated Excel workbook. You must configure a Ribbon tab as described in Section 8.3, "Configuring the Runtime Ribbon Tab" so that the end user can view this branding information by clicking a ribbon button that invokes the `ViewAboutDialog` workbook action at runtime. For more information about workbook actions, see Table A–17.

You can also define string keys in a resource bundle to define information, such as titles, in one location that can then be used in multiple locations in an integrated Excel workbook at runtime when EL expressions retrieve the values of these string keys. For information about defining string keys, see Section 10.2, "Using Resource Bundles in an Integrated Excel Workbook."

9.6.1 How to Brand an Integrated Excel Workbook

You define values for the workbook `BrandingItems` group of properties.

To brand an integrated Excel workbook:

1. Open the integrated Excel workbook.
2. In the **Oracle ADF** tab, click **Workbook Properties.**
3. In the Edit Workbook Properties dialog, click the ellipsis button (**...**) beside the input field for **BrandingItems**.

4. In the the NameValuePair Collection Editor, click **Add** and specify values for the new element as follows:

   - `Name`

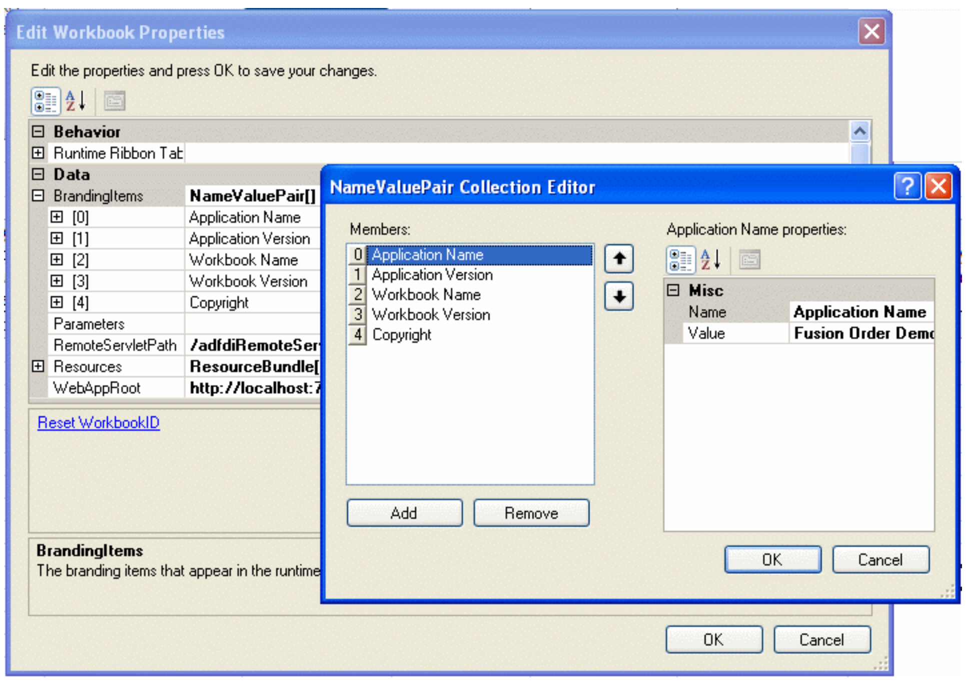
     Specify the name, or the EL expression, of the branding item to define.

   - `Value`

     Specify a literal string or click the ellipsis button (...) to invoke the expression builder and write an EL expression that retrieves a value at runtime. `BrandingItems` must use literal strings or resource expressions, and must not contain any binding expression.

   Figure 9–10 shows the design time view of branding items in the Master Price List module.

Figure 9–10 Design Time View of Branding Items in the Master Price List Module



Edit Workbook Properties

Edit the properties and press OK to save your changes.

⊟ **Behavior**	
⊞ Runtime Ribbon Tab	
⊟ **Data**	
⊟ BrandingItems	**NameValuePair[]**
⊞ [0]	Application Name
⊞ [1]	Application Version
⊞ [2]	Workbook Name
⊞ [3]	Workbook Version
⊞ [4]	Copyright
Parameters	
RemoteServletPath	**/adfdiRemoteSer**
⊞ Resources	**ResourceBundle[**
WebAppRoot	**http://localhost:7**

Reset WorkbookID

BrandingItems
The branding items that appear in the runtime

NameValuePair Collection Editor

Members:

0 Application Name
1 Application Version
2 Workbook Name
3 Workbook Version
4 Copyright

Application Name properties:

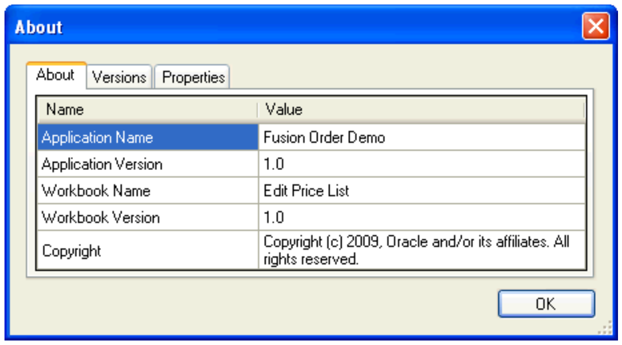
⊟ **Misc**	
Name	**Application Name**
Value	**Fusion Order Demo**

5. Click **OK**.

9.6.2 What Happens at Runtime to the Branding Items in an Integrated Excel Workbook

At runtime, the name-value pairs that you define for the `BrandingItems` group of properties appear in a dialog that the end user invokes from the **About** button of the **Oracle ADF** tab, which you configured to appear, as described in Section 8.3, "Configuring the Runtime Ribbon Tab." Figure 9–11 shows the runtime view of branding items in the Master Price List module's `EditPriceList.xlsx` workbook.

Figure 9–11 Runtime View of Branding Items in the Master Price List Module



Name	Value
Application Name	Fusion Order Demo
Application Version	1.0
Workbook Name	Edit Price List
Workbook Version	1.0
Copyright	Copyright (c) 2009, Oracle and/or its affiliates. All rights reserved.

9.7 Using Worksheet Protection

By default, the end user can edit the values of read-only cells and ADF Desktop Integration read-only components, such as ADF Label and ADF Output Text, at runtime. While uploading data, ADF Desktop Integration ignores these changes and overwrites them when it downloads data from the Fusion web application.

To prevent editing of read-only cells at runtime, you must enable the worksheet protection. Optionally, you can also provide a password to prevent the end user from turning off worksheet protection.

9.7.1 How to Enable Worksheet Protection

Worksheet protection enables true read-only mode for read-only cells and prevents any editing at runtime.

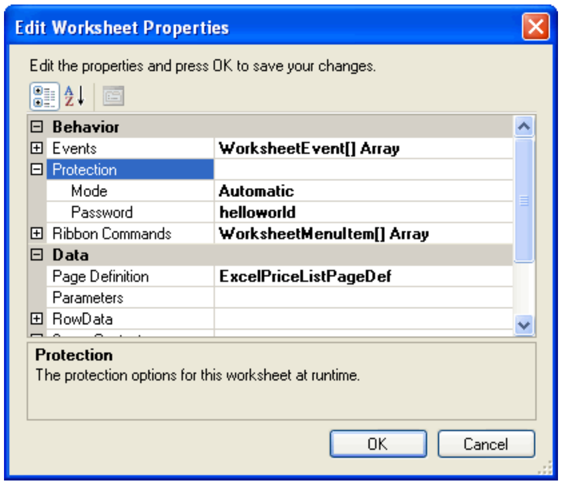
To enable Worksheet Protection

1. Open the integrated Excel workbook.
2. In the **Oracle ADF** tab, click **Worksheet Properties.**
3. In the Edit Worksheet Properties dialog, expand the **Protection** property and configure values as follows:
   - To enable the worksheet protection at runtime, set the **Mode** to **Automatic**.
   - If desired, provide a value in the **Password** field. The end-user cannot turn off sheet protection at runtime without knowing this value.

     Note that the password is not encrypted and the maximum password length allowed by Excel is 255 characters. If you specify a longer password, it will be truncated silently at runtime when sheet protection is toggled.

   Figure 9–12 shows the design time view of worksheet protection in Master Price List module.

Figure 9–12 Design Time View of Worksheet Protection in the Master Price List Module


Edit Worksheet Properties

Edit the properties and press OK to save your changes.

Behavior	
Events	WorksheetEvent[] Array
Protection	
Mode	Automatic
Password	helloworld
Ribbon Commands	WorksheetMenuItem[] Array
Data	
Page Definition	ExcelPriceListPageDef
Parameters	
RowData	

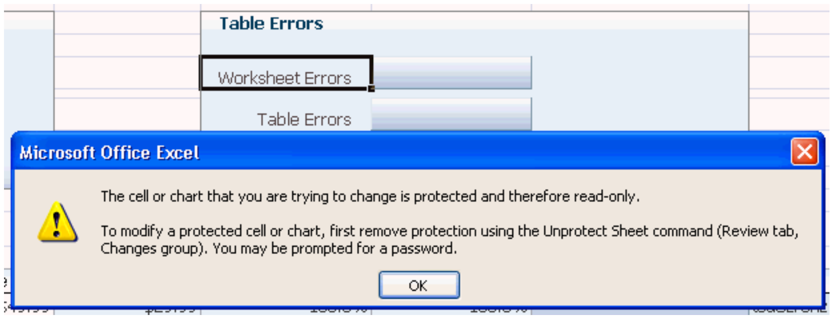
Protection
The protection options for this worksheet at runtime.

4. Click **OK**.

9.7.2 What Happens at Runtime When Worksheet Protection is Enabled

At runtime, if the end user tries to edit a read-only cell or a ADF Desktop Integration read-only component, Excel displays the warning message as shown in Figure 9–13.

Figure 9–13 Worksheet Protection Warning at Runtime



The cell or chart that you are trying to change is protected and therefore read-only.

To modify a protected cell or chart, first remove protection using the Unprotect Sheet command (Review tab, Changes group). You may be prompted for a password.

When worksheet protection is enabled, ADF Desktop Integration controls the **Locked** property for cells that are within the bounds of ADF Desktop Integration components. The **Locked** property of cells outside the bounds of ADF Desktop Integration components is not affected.

At runtime, ADF Desktop Integration evaluates the read-only behavior of its components. Some components such as ADF Labels and ADF Output Text are always read-only, and other components, such as ADF Input Text, have a read-only property. At runtime, the **Locked** property is set to true when read-only for the component evaluates to true. The header labels of ADF Table components are always read-only, but column subcomponents might be ADF Output Text or ADF Input Text. At runtime, each components read-only behavior is evaluated and the corresponding cell’s **Locked** property is set to the appropriate value.

9.7.3 What You May Need to Know About Worksheet Protection

Worksheet protection is not enabled by default, you need to enable it at design time if you want to use it for a particular worksheet. Also, note that after enabling worksheet protection, the **Locked** property for cells is set at runtime, not at design time.

It is important to note that the password used for worksheet protection is itself not encrypted or stored in a safe location. The worksheet protection is used to improve worksheet usability, not to protect sensitive data.

After worksheet protection is enabled, Excel behaves differently. Here are some differences that you can expect:

- The ADF Table components cannot be sorted, as they include read-only cells in the Key column.
- The end user can insert a full row or column. However, once inserted, they cannot be deleted.
- The end user cannot insert partial rows or columns.

10 Internationalizing Your Integrated Excel Workbook

This chapter describes internationalization issues to consider when developing an integrated Excel workbook.

This chapter includes the following sections:

- Section 10.1, "Introduction to Internationalizing Your Integrated Excel Workbook"
- Section 10.2, "Using Resource Bundles in an Integrated Excel Workbook"
- Section 10.3, "Localization in ADF Desktop Integration"

10.1 Introduction to Internationalizing Your Integrated Excel Workbook

ADF Desktop Integration provides several features that allow you to deliver integrated Excel workbooks as part of an internationalized Fusion web application. One of the principal features is the use of resource bundles to manage the localization of user-visible strings that appear in Oracle ADF components at design time. It also uses resource bundles to manage the user-visible strings that appear in these components at runtime. This chapter also describes the use of resource bundles.

Note the following points about internationalization and localization in an integrated Excel workbook:

- Internationalized Data

  ADF Desktop Integration supports both single- and double-byte character sets. It marshals data transmitted between an Excel worksheet and a Fusion web application into XML payloads. These XML payloads use UTF-8 encoding with dates, times, and numbers in canonical formats.

- Locale

  The locale of the system where the Excel workbook is used determines the format for dates, times, and numbers. These settings (formats and the locale of the system) may differ from the settings used by the Fusion web application. ADF Desktop Integration does not attempt to synchronize these settings, but it ensures that the data retains its integrity. ADF Desktop Integration does not provide a mechanism for end users to change the language or display settings of the Oracle ADF components in an integrated Excel workbook at runtime.

  When configuring or applying styles to ADF components in an integrated Excel workbook, configure or choose styles that are locale-sensitive. For more information, see Section 9.2, "Working with Styles."

For information about internationalizing Fusion web applications, see the "Internationalizing and Localizing Pages" chapter in the *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

10.2 Using Resource Bundles in an Integrated Excel Workbook

ADF Desktop Integration uses resource bundles to manage user-visible strings that appear in the ADF components of an integrated Excel workbook at design time and runtime. JDeveloper stores resource bundles in the ADF Desktop Integration project.

You can register up to twenty resource bundles containing string values you define with an integrated Excel workbook. A resource bundle must not exceed 1 megabyte. If you attempt to register more than twenty resource bundles or a resource bundle that exceeds 1 megabyte, ADF Desktop Integration writes a warning to the client-side log file and stops registration of additional resource bundles or resource bundle data after the 1 megabte limit is reached.

For example, if resource bundle *A* measures 2 megabyte and resource bundle *B* measures 1 megabyte, ADF Desktop Integration registers the first megabyte of data in resource bundle *A* and all of the data in resource bundle *B*. For information about client-side logging, see Section C.3.2, "About Client-Side Logging."

The `Resources` workbook property specifies what resource bundles an integrated Excel workbook can use. This property specifies an array of resource bundles (`ResourceBundle[] Array`) in the integrated Excel workbook. Each element in the array has a property that uniquely identifies a resource bundle (`Alias`) and a property that identifies the path to the resource bundle in the JDeveloper desktop integration project (`Class`). For example, `EditPriceList-DT.xlsx` in the Master Price List module references the `res` resource bundle that has the following value for the `Class` property:

```
oracle.fodemo.masterpricelist.resources.UIStrings
```

More information about the `Resources` workbook property can be found in Section A.12, "Workbook Actions and Properties."

By default, ADF Desktop Integration provides a *reserved resource bundle* that supplies string key values used by many component properties at runtime. When you enable an Excel workbook to integrate it with a Fusion web application, the reserved resource bundle is registered by default with the workbook. ADF Desktop Integration uses the value `_ADFDIres` to uniquely identify this resource bundle. Many EL expressions reference string values in this resource bundle. For example, the following EL expression is the default value of the `Label` property for the `Login` button in `WorkbookMenuItems`:

```
#{_ADFDIres['TOOLBAR_MENU_CMD_LOGIN']}
```

where `_ADFDIres` identifies the reserved resource bundle and `TOOLBAR_MENU_CMD_LOGIN` is the key that identifies the string (`Login...`) in the resource bundle to use at runtime for the `Login` button.

If you register another resource bundle, you can replace default string key values assigned from the `_ADFDIres` resource bundle to many of the ADF component properties.

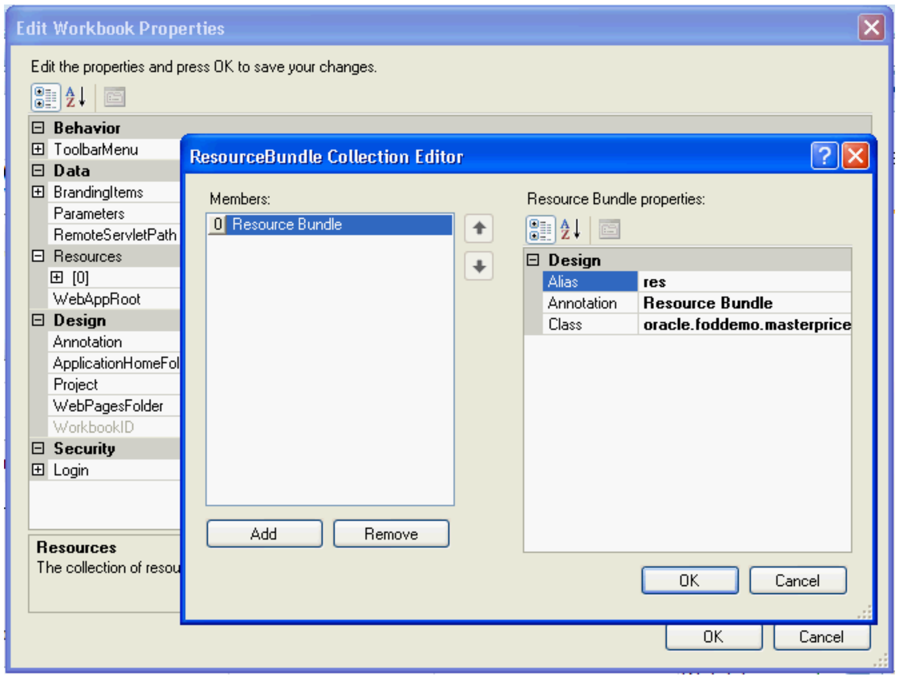
10.2.1 How to Register a Resource Bundle in an Integrated Excel Workbook

You register a resource bundle by adding an element to the `ResourceBundle[] Array` using the ResourceBundle Collection Editor.

To register a resource bundle:

1. In the Excel workbook, click **Workbook Properties** in the ADF Desktop Integration task pane to display the Edit Workbook Properties dialog.
2. Click the ellipsis button (**...**) beside the input field for **Resources** to display the ResourceBundle Collection Editor shown in Figure 10–1.

Figure 10–1 ResourceBundle Collection Editor



Edit Workbook Properties

Edit the properties and press OK to save your changes.

ResourceBundle Collection Editor

Members:
0 Resource Bundle

Resource Bundle properties:

Design	
Alias	res
Annotation	Resource Bundle
Class	oracle.foddemo.masterprice

3. Specify values for the resource bundle and then click **OK**.

   For information about the values to specify for a resource bundle, see the entry for Resources in Table A–18.

10.2.2 How to Replace String Key Values from the Reserved Resource Bundle

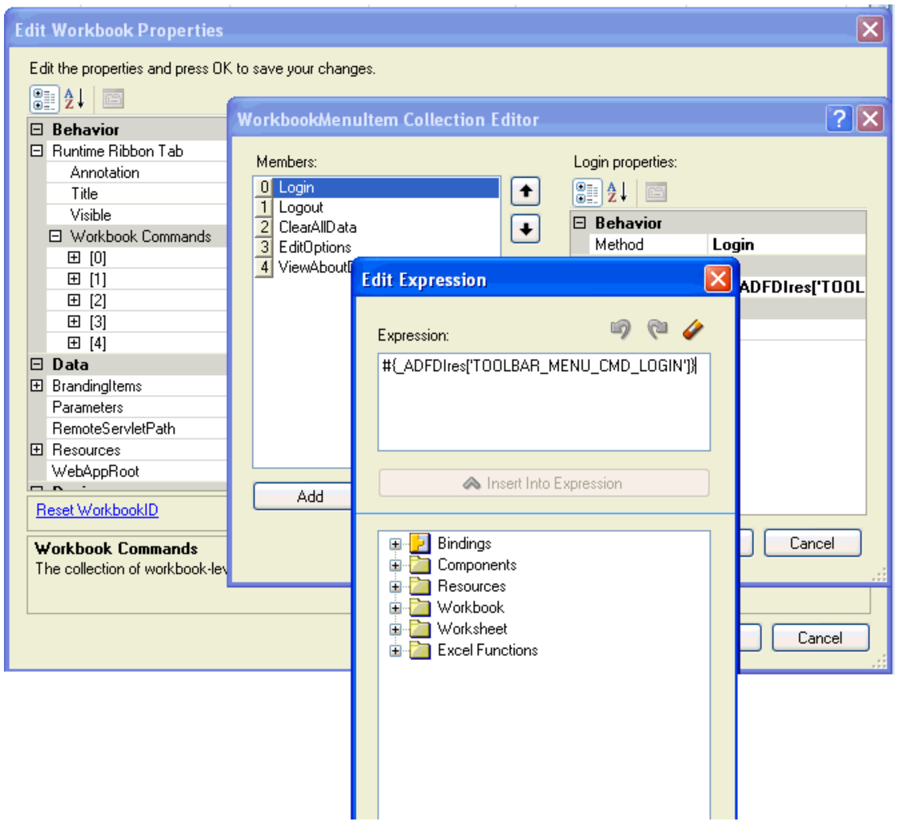
You can replace a string key from the reserved resource bundle by specifying a string key from a resource bundle that you registered with the integrated Excel workbook.

To replace a string key value from the reserved resource bundle:

1. Open the integrated Excel workbook.
2. Click **Workbook Properties** in the **Oracle ADF** tab.
3. Click **Runtime Ribbon Tab** and click the ellipsis button (**...**) beside the input field for **WorkbookMenuItem[] Array**.
4. Click the ellipsis button (**...**) beside the input field for the `Label` property to display the expression builder.
5. Write an EL expression to retrieve the string key value you want to display at runtime.

   Figure 10–2 shows an example where the EL expression references a string key (`key_Login`) defined in a resource bundle (`res`). This EL expression replaces the default EL expression that references a string key in the `_ADFDIres` resource bundle.

Figure 10–2 Expression Builder Replacing a Key String Value



```
#{_ADFDIres['TOOLBAR_MENU_CMD_LOGIN']}
```

6. Click **OK**.

10.2.3 How to Override Resources That Are Not Configurable

The overridable resources contains several user-visible runtime strings that you cannot replace by configuring the properties of the ADF Desktop Integration components. Examples include the strings that appear in the default upload dialog illustrated in Figure 7–5.

To replace these user-visible runtime strings, you create a resource bundle in your Fusion web application that contains the string keys from the overridable resource that ADF Desktop Integration supports. Appendix F, "String Keys in the Overridable Resources" lists these string keys. You define values for the string keys listed in Appendix F to override in the resource bundle you create.

To override resources that are not configurable:

1. Create a resource bundle in your Fusion web application.

   For information about creating a resource bundle, see the "Defining Locales and Resource Bundles" section in the *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

2. Define the string key values you want to appear at runtime in the resource bundle for the string keys listed in Appendix F, "String Keys in the Overridable Resources."

3. Set `_ADFDIres` as the value of the **Alias** property when you register the resource bundle you created in Step 1.

   For information about how to register a resource bundle, see Section 10.2.1, "How to Register a Resource Bundle in an Integrated Excel Workbook."

Table F–1 describes the string keys in the overridable resources that ADF Desktop Integration supports. Supply an alternative value to the value listed in the English value column for each string key in the overridable resource.

10.2.4 What Happens at Runtime When You Override Resources That Are Not Configurable

ADF Desktop Integration retrieves the values of string keys listed in Table F–1 that you defined in the resource bundle you created. It retrieves the values of other string keys that you did not define in the resource bundle you created from the reserved resource bundle.

Note: If you provide override values for `LOGIN_WINDOW_TITLE` or `LOGIN_CONFIRM_CONNECT_2`, the override values will not appear on the first connection to the web application. The default values will be used on the first connection. For more information, see Section 10.2.2, "How to Replace String Key Values from the Reserved Resource Bundle."

10.2.5 What You May Need to Know About Resource Bundles

See the following sections for additional information about resource bundles in an integrated Excel workbook.

Resource Bundle Types

ADF Desktop Integration supports use of the following types of resource bundle:

- Properties bundle (`.properties`)
- List resource bundle (`.rts`)
- Xliff resource bundle (`.xlf`)

For more information about resource bundles, see the "Manually Defining Locales and Resource Bundles" section in the *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

Caching of Resource Bundles in an Integrated Excel Workbook

ADF Desktop Integration caches the values of string keys from the resource bundles that an integrated Excel workbook retrieves when it first connects to the Fusion web application. If you change a string key value in a resource bundle after an integrated Excel workbook has cached the previous value, the modified value does not appear in the workbook unless the `ClearAllData` workbook action is invoked and the end user closes and reopens the workbook so that it retrieves the modified value from the Fusion web application. For more information about the `ClearAllData` workbook action, see Table A–17.

EL Expression Syntax for Resource Bundles

ADF Desktop Integration requires that you enclose the string key name in EL expressions using the `[]` characters, as in the following example:

```
#{res['StringKey']}
```

Note that ADF Desktop Integration does not support the following syntax:

```
#{res.StringKey}
```

10.3 Localization in ADF Desktop Integration

ADF Desktop Integration integrates several diverse sets of technologies. Each of these technologies provides various options for controlling the choice of natural human language when you localize your Fusion web application.

When the end user interacts with an integrated Excel workbook, various elements are involved. Each of these elements has its own set of supported languages and resource translations. In such a scenario, the translation of language is the responsibility of the respective publisher.

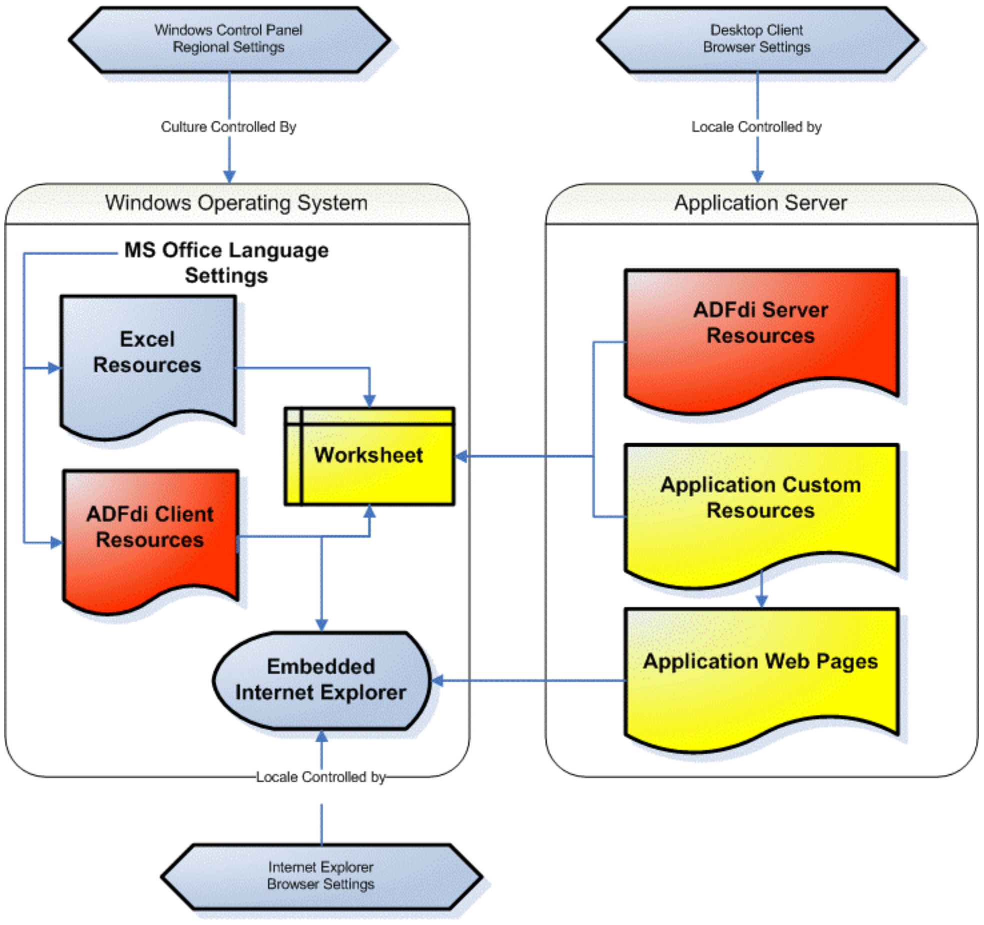
Table 10–1 presents a summary of elements involved and their role in translation:

Table 10–1 Summary of Localization

Area subject to localization	Determination of language to use
Microsoft operating system	Operating system language settings. You can choose the language through Regional Settings on Control Panel.
Microsoft Office	Microsoft Office language settings
Web pages displayed in ADF Desktop Integration Dialog actions	Usually controlled by Microsoft Internet Explorer's Language Preferences.
ADF Desktop Integration client resources	Microsoft Office language settings
ADF Desktop Integration server resources	Microsoft Internet Explorer language preferences
ADF Desktop Integration custom resource bundles	Microsoft Internet Explorer language preferences

Figure 10–3 illustrates how various elements involved in a Fusion web application play their role in translation.

Figure 10–3 Localization in ADF Desktop Integration



For more information about localization in ADF Desktop Integration, see the "*Oracle ADF 11g Desktop Integration Localization whitepaper*" on OTN at:

`http://www.oracle.com/technetwork/developer-tools/jdev/adfdi-localization-129807.pdf`

11 Securing Your Integrated Excel Workbook

This chapter provides information about how to secure your integrated Excel workbook with a Fusion web application by providing authentication and authorization capabilities in an authenticated session. The chapter describes how to access integrated Excel workbooks in an authenticated and non authenticated session. It also describes issues that you should be aware of so that you can secure your Excel workbook.

It is recommended that you configure ADF Security for the Fusion web application before you deploy the integrated Excel workbook; however, it is not required. For information about ADF Security, see the "Enabling ADF Security in a Fusion Web Application" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

This chapter includes the following sections:

- Section 11.1, "Introduction to Security In Your Integrated Excel Workbook"
- Section 11.2, "Authenticating the Excel Workbook User"
- Section 11.3, "Checking the Integrity of an Integrated Excel Workbook's Metadata"
- Section 11.4, "What You May Need to Know About Securing an Integrated Excel Workbook"

11.1 Introduction to Security In Your Integrated Excel Workbook

If you are using a Fusion web application that does not enforce authentication, then the integrated Excel workbook verifies and creates a valid user session before connecting to the Fusion web application and downloading data. The session that is established is used for each and every data transfer between the integrated Excel workbook and Fusion web application. The session is also used for web pages referenced from the integrated Excel workbook.

In a Fusion web application that is enforcing authentication, the integrated Excel workbook ensures that a valid, authenticated user session is established before transferring data to or from the web application.

For both authenticated and non-authenticated Fusion web applications, ADF Desktop Integration relies on the establishment of cookie-based sessions. With no authentication mechanism in place, your integrated Excel workbooks are not completely safe. Hence, it is recommended that you enable ADF Security in your Fusion web application before you deploy your web application with integrated Excel workbooks.

11.2 Authenticating the Excel Workbook User

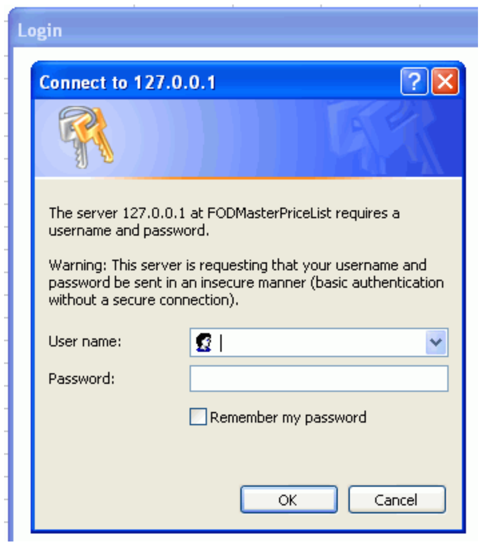
The integration of an Excel workbook with a secure Fusion web application requires an authenticated web session established between the integrated Excel workbook and the server that hosts the Fusion web application. ADF Security determines the mechanism used to authenticate the user.

If the end user opens an Excel workbook without a valid authenticated session, a login mechanism is invoked to authenticate the end user.

11.2.1 What Happens at Runtime When the Login Method Is Invoked

After the login method is invoked, a new session between the integrated Excel workbook and the Fusion web application is created, and a modal dialog appears that contains a web browser control. This web browser control displays whatever login mechanism the Fusion web application uses. For example, if the Fusion web application uses HTTP Basic Authentication, the web browser control displays the simple dialog shown in Figure 11–1.

Figure 11–1 Dialog That Appears When a Fusion Web Application Uses Basic Authentication



The server 127.0.0.1 at FODMasterPriceList requires a username and password.

Warning: This server is requesting that your username and password be sent in an insecure manner (basic authentication without a secure connection).

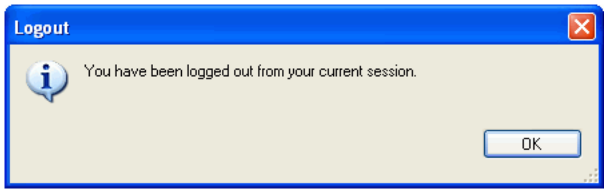
The end user enters user credentials and, assuming these are valid, an authentication session is created and assigned to the client (the web browser control hosted by the Excel workbook).

Note: If the `Login` method is invoked when a session has already been established, it first invokes the `Logout` action internally to free that session.

11.2.2 What Happens at Runtime When the Logout Method Is Invoked

After the logout method is invoked, a dialog appears informing users that they have logged out of the current session. The user is automatically logged out when the workbook is closed, or when the **Clear All Data** option is selected from the runtime custom tab in Excel ribbon.

Figure 11–2 Dialog That Appears When a User Logs Out



You have been logged out from your current session.

After logging out, the user must log in again to upload or download data.

If two or more workbooks are open (in test or runtime mode) with same credentials, closing one workbook does not initiate the logout mechanism. The user continues to stay logged in and may continue to work on remaining open workbooks, and can open the closed workbook without being asked for credentials again. The user is logged out when all workbooks, requiring same credentials, are closed.

11.3 Checking the Integrity of an Integrated Excel Workbook's Metadata

ADF Desktop Integration provides a mechanism to verify that the metadata it uses to integrate an Excel workbook with a Fusion web application is not tampered with after you publish the Excel workbook for end users. It generates a hash code value and inserts the value into the ADF Desktop Integration client registry file (`adfdi-client-registry.xml`) that it also creates when you publish the integrated Excel workbook as described in Section 14.3, "Publishing Your Integrated Excel Workbook." ADF Desktop Integration stores the `adfdi-client-registry.xml` file in the `WEB-INF` directory of the Fusion web application.

If you republish the integrated Excel workbook, ADF Desktop Integration generates a new hash code value and replaces the value in the `adfdi-client-registry.xml` file. ADF Desktop Integration creates the `adfdi-client-registry.xml` file if it does not exist.

The `ApplicationHomeFolder` and `WebPagesFolder` workbook properties allow the integrated Excel workbook to identify the location of the Fusion web application's `WEB-INF` directory. You must set valid values for these properties before you can publish the integrated Excel workbook and ADF Desktop Integration can generate a hash code value.

ADF Desktop Integration generates the hash code value using most of the elements in the metadata for the workbook and the value of the `WorkbookID` workbook property. The `WorkbookID` workbook property is read-only and uniquely identifies the integrated Excel workbook. You must reset the `WorkbookID` workbook property if you create a new integrated Excel workbook by copying an existing integrated Excel workbook. ADF Desktop Integration excludes the `WebAppRoot` property from the hash code calculation since its value is expected to change at runtime.

For more information about the workbook properties discussed here, see Table A–18.

11.3.1 How to Reset the Workbook ID

The value of the `WorkbookID` workbook property is unique to each workbook and cannot be modified by you. You can, however, reset the `WorkbookID` workbook property. You must do this when you create a new integrated Excel workbook by copying an existing integrated Excel workbook.

To reset a workbook ID:

1. Open the integrated Excel workbook.
2. Click **Workbook Properties** in the **Oracle ADF** tab.
3. In the Edit workbook Properties dialog, click the **Reset WorkbookID** link.
4. Click **Yes** to confirm and reset the `WorkbookID` workbook property in the dialog that appears.
5. Click **OK**.

11.3.2 How to Disable the Metadata Tamper-Check in the Fusion Web Application

By default, ADF Desktop Integration verifies that the metadata it uses to integrate an Excel workbook with a Fusion web application is not tampered with after you publish the Excel workbook for end users. You can disable the metadata tamper-check by configuring a parameter in the deployment descriptor file (`web.xml`) of the Fusion web application, although this is not recommended.

To disable the metadata tamper-check in the Fusion web application:

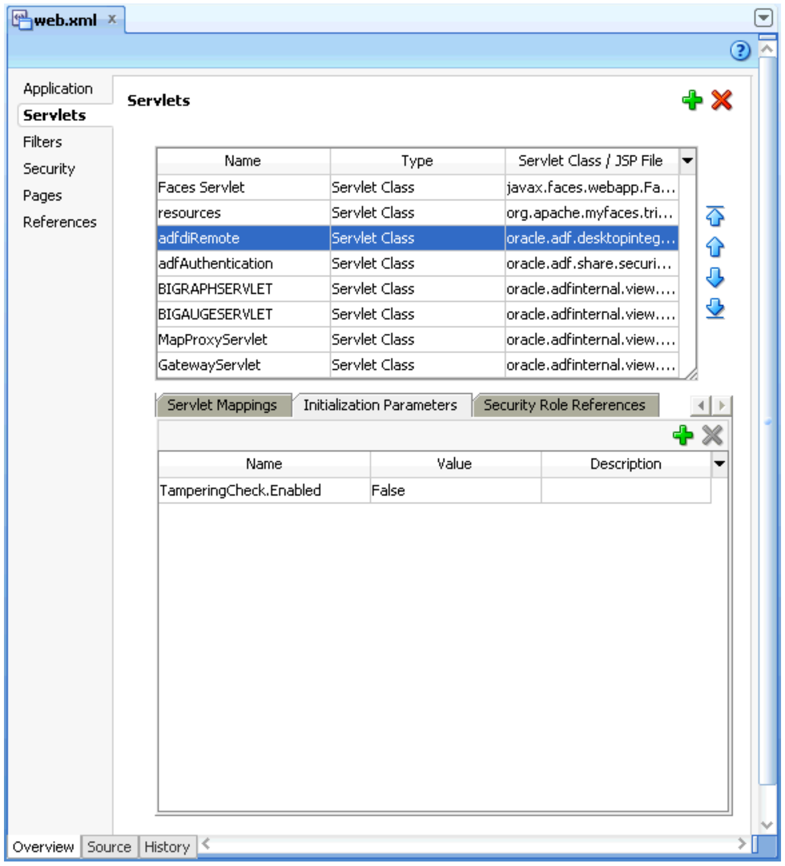
1. Stop your Fusion web application.
2. Open the `web.xml` file of your Fusion web application.
3. Add an initialization parameter to the `adfdiRemote` servlet to disable the metadata tamper-check, as described in Table 11–1.

Table 11–1 Disabling Metadata Tamper-Check

Property	Value
`Name`	Enter the name of the initialization parameter as follows: `TamperingCheck.Enabled`
`Value`	Set the value of `TamperingCheck.Enabled` to `False`.

Figure 11–3 shows the `web.xml` editor in JDeveloper.

Figure 11–3 Disabling the Metadata Tamper Check In JDeveloper



Name	Type	Servlet Class / JSP File
Faces Servlet	Servlet Class	javax.faces.webapp.Fa...
resources	Servlet Class	org.apache.myfaces.tri...
adfdiRemote	Servlet Class	oracle.adf.desktopinteg...
adfAuthentication	Servlet Class	oracle.adf.share.securi...
BIGRAPHSERVLET	Servlet Class	oracle.adfinternal.view....
BIGAUGESERVLET	Servlet Class	oracle.adfinternal.view....
MapProxyServlet	Servlet Class	oracle.adfinternal.view....
GatewayServlet	Servlet Class	oracle.adfinternal.view....

Name	Value	Description
TamperingCheck.Enabled	False	

4. Save the `web.xml` file.

The `web.xml` file of your Fusion web application has the following entries:

```
<servlet>
        <servlet-name>adfdiRemote</servlet-name>
        <servlet-class>...</servlet-class>
        <init-param>
            <param-name>TamperingCheck.Enabled</param-name>
            <param-value>False</param-value>
        </init-param>
</servlet>
```

5. Rebuild and restart your Fusion web application.

For more information about the `web.xml` file, see Appendix E, "ADF Desktop Integration Settings in the Web Application Deployment Descriptor."

11.3.3 How to Allow Missing Entries in the ADF Desktop Integration Client Registry

You can configure the metadata tamper-check so that a missing entry for the `WorkbookID` workbook property is allowed in the `adfdi-client-registry.xml` file. To do this, you configure a parameter in the deployment descriptor file (`web.xml`) of the Fusion web application, although this is not recommended.

To allow missing entries in the metadata of the Fusion web application:

1. Stop your Fusion web application.

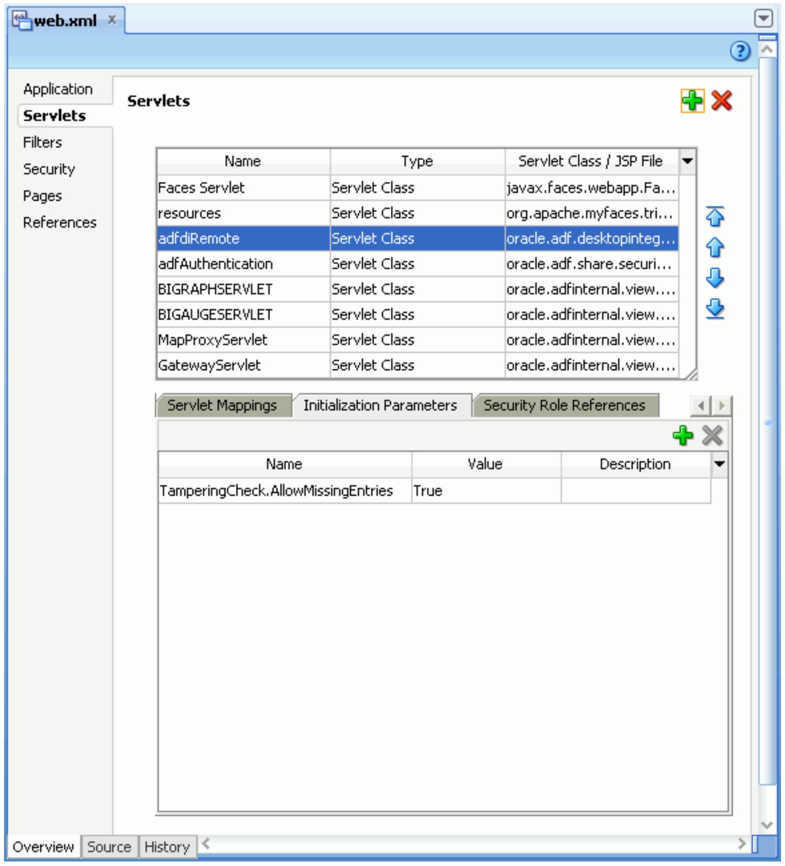
2. Open the `web.xml` file of your Fusion web application.
3. Add an initialization parameter to the `adfdiRemote` servlet to allow missing entries in the metadata as described in Table 11–2.

Table 11–2 Allowing Missing Entries in the Metadata

Property	Value
`Name`	Enter the name of the initialization parameter as follows: `TamperingCheck.AllowMissingEntries`
`Value`	Set the value of `TamperingCheck.AllowMissingEntries` to `True`.

Figure 11–4 shows the `web.xml` editor in JDeveloper.

Figure 11–4 Enabling Missing Metadata Entries In JDeveloper



Name	Type	Servlet Class / JSP File
Faces Servlet	Servlet Class	javax.faces.webapp.Fa...
resources	Servlet Class	org.apache.myfaces.tri...
adfdiRemote	Servlet Class	oracle.adf.desktopinteg...
adfAuthentication	Servlet Class	oracle.adf.share.securi...
BIGRAPHSERVLET	Servlet Class	oracle.adfinternal.view....
BIGAUGESERVLET	Servlet Class	oracle.adfinternal.view....
MapProxyServlet	Servlet Class	oracle.adfinternal.view....
GatewayServlet	Servlet Class	oracle.adfinternal.view....

Name	Value	Description
TamperingCheck.AllowMissingEntries	True	

4. Save the `web.xml` file.

   The `web.xml` file of your Fusion web application has the following entries:

```
<servlet>
        <servlet-name>adfdiRemote</servlet-name>
        <servlet-class>...</servlet-class>
        <init-param>
            <param-name>TamperingCheck.AllowMissingEntries</param-name>
            <param-value>True</param-value>
        </init-param>
</servlet>
```

5. Rebuild and restart your Fusion web application.

For more information about the `web.xml` file, see Appendix E, "ADF Desktop Integration Settings in the Web Application Deployment Descriptor."

11.3.4 What Happens When the Metadata Tamper-Check is Performed

At runtime, the integrated Excel workbook regenerates the metadata hash code and provides it to the Fusion web application with the first server request. If the Fusion web application cannot get a match on this hash code, it returns an error to the integrated Excel workbook. On receiving an error from the tamper check process, the integrated Excel workbook reports this failure to the end user and closes the integration framework.

11.4 What You May Need to Know About Securing an Integrated Excel Workbook

Note the following points about securing an integrated Excel workbook with a Fusion web application:

- Data security

  If you save an Excel workbook containing data downloaded from a Fusion web application to a location, such as a network directory, where other users can access the Excel workbook, the data stored in the Excel workbook is accessible to other users.

- Security and protection in Microsoft Excel

  Certain security and protection features that Microsoft Excel provides do not work for workbooks or worksheets that are integrated with a Fusion web application. For example, you cannot use the worksheet protection feature for a worksheet that you integrate with a Fusion web application. You can, however, use Excel's functionality to set a password on an integrated Excel workbook to prevent unauthorized users from opening or modifying it. For more information about Excel security features, see Excel's documentation.

- Integrated Excel workbooks can be configured to cache data, as described in Section 15.2, "Restore Server Data Context Between Sessions." Make sure that you do not cache sensitive data in the integrated Excel workbook.

- If the Fusion web application is running on the `https` protocol and you have not installed the security certificate on the client, the integrated Excel workbook gives an error on login and the connection is not established. To establish a connection, you must install the security certificate.

- For applications running in an environment using Oracle Access Manager, the system administrator should ensure that the URL for the ADF Desktop Integration Remote servlet is configured as a protected resource for Oracle Access Manager. For more information, see the *Oracle Access Manager Access System Administration Guide*.

- For applications running in an environment using WebGate, set the user-defined parameter `filterOAMAuthnCookie` to `False`. For more information, see the chapter on registering partners (agents and applications) remotely in the *Oracle Access Manager Access System Administration Guide*.

- System administrators should ensure that applications using ADF Desktop Integration have a security constraint configured in `web.xml` that protects the ADF Desktop Integration remote servlet.

  The following code extract from `web.xml` shows a security constraint protecting the remote servlet:

  ```
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>adfdiRemote</web-resource-name>
      <url-pattern>/adfdiRemoteServlet</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>valid-users</role-name>
    </auth-constraint>
  </security-constraint>
  ```

12 Adding Validation to an Integrated Excel Workbook

This chapter describes how to provide validation for your integrated Excel workbook

This chapter includes the following sections:

12.1 Introduction to Adding Validation to Integrated Excel Workbook

You configure server-side and client-side validation for the Fusion web application and the integrated Excel workbook to make use of the validation options offered by the ADF Model Layer and Microsoft Excel. In addition to these validation options, you can make use of components in ADF Desktop Integration to return error messages from the Fusion web application, to provide status on the results of component actions, and to manage errors that may occur when data is changed in an integrated Excel workbook conflict with data hosted by the Fusion web application.

12.2 Providing Server-Side Validation for an Integrated Excel Workbook

ADF Desktop Integration uses the validation rules that the ADF Model Layer sets for a binding's attributes. Data that the end user enters or edits in one of the ADF Desktop Integration components, such as the ADF Table component, can be validated against set rules and conditions in the Fusion web application. For general information about defining validation rules in Oracle ADF, see the "Defining Validation and Business Rules Declaratively" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

For information about adding ADF Model layer validation, see the "Adding ADF Model Layer Validation" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

12.3 Providing Client-Side Validation for an Integrated Excel Workbook

ADF Desktop Integration does not provide client-side validation. However, you can use Excel's data validation features to control the type of data or the values that end users enter into a cell. These features allow you to restrict data entry to a certain range of dates, limit choices by using a list, or ensure that only positive whole numbers are entered in a cell. For example, you could configure the `Product Number` column in the `EditPriceList-DT.xlsx` workbook so that users can enter only whole numbers in the cells of this column.

If you apply custom validation to columns within an ADF Table component, the validation is propagated when the ADF Table component's columns are populated at runtime. Note, however, that ADF Desktop Integration overwrites at runtime any custom validation applied to columns that reference the `TreeNodeList` subcomponent at design time. This is because ADF Desktop Integration applies its own list-constraint validation, which is invoked at runtime.

Note:

- Excel displays error messages when a validation fails; these error messages cannot be localized.
- ADF Desktop Integration does not support server-side validation warnings. Validation warnings, set for rules defined in the Fusion web application, are not displayed by the integrated Excel workbook.

For more information about data validation in Excel, see Excel's documentation.

12.4 Error Reporting in an Integrated Excel Workbook

The server that hosts the Fusion web application you integrate your Excel workbook with can return error messages to end users that provide feedback on the results of operations. The error messages returned can be one many types: validation failures, conflict errors, deleted records, and so on.

12.4.1 Error Reporting Using EL Expressions

To return error message summaries to end users, you must set an EL expression for the `Value` property of an ADF Output Text component. At runtime, the ADF Output Text component displays the error message summary to the end user if an error occurs.

The type of EL expression that you set for the `Value` property of the ADF Output Text component depends on whether you want to return error message summaries generated by action sets invoked on a worksheet, or by actions invoked by other components such as the ADF Table and ADF Read-only Table components. The following EL expression displays error message summaries which are returned during the invocation of an action set on a worksheet:

```
#{worksheet.errors}
```

At runtime, the previous error message summary is cleared (if one existed) when the action set starts the invocation. If no errors occur during invocation, error message remains blank. If an error occurs, the ADF Output Text component displays the error message summary.

An alternative approach to returning error message summaries generated by action sets invoked on a worksheet is to set `#{worksheet.errors}` as the value for an action set's `Alert.FailureMessage` property. This approach displays the generated error message summary in a dialog.

Components such as the ADF Table and ADF Read-only Table components that have actions which interact with the Fusion web application can also return error message summaries. Set the following EL expression for the `Value` property of the ADF Output Text component or for an action set's `Alert.FailureMessage` property:

```
#{components.componentID.errors}
```

where `componentID` refers to the ID of the component (ADF Table or ADF Read-only Table component) that invokes the action.

The `EditPriceList-DT.xlsx` file in the Master Price List module of the Fusion Order Demo application demonstrates how to return error message summaries generated by action sets invoked on a worksheet and by the actions of an ADF Table component. Figure 12–1 shows these EL expressions in design mode.

Figure 12–1 EL Expressions to Return Error Messages in an ADF Output Text Component

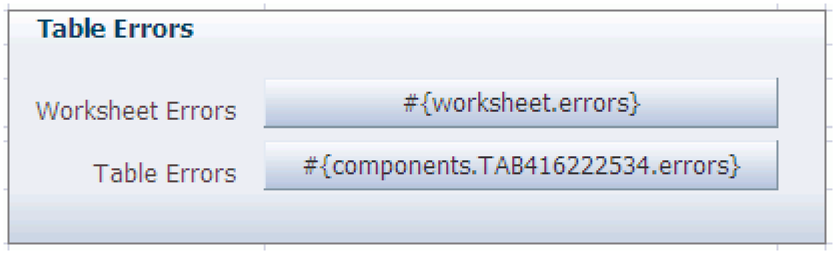


Table Errors

Worksheet Errors	#{worksheet.errors}
Table Errors	#{components.TAB416222534.errors}

12.4.2 Error Reporting Using Component Actions

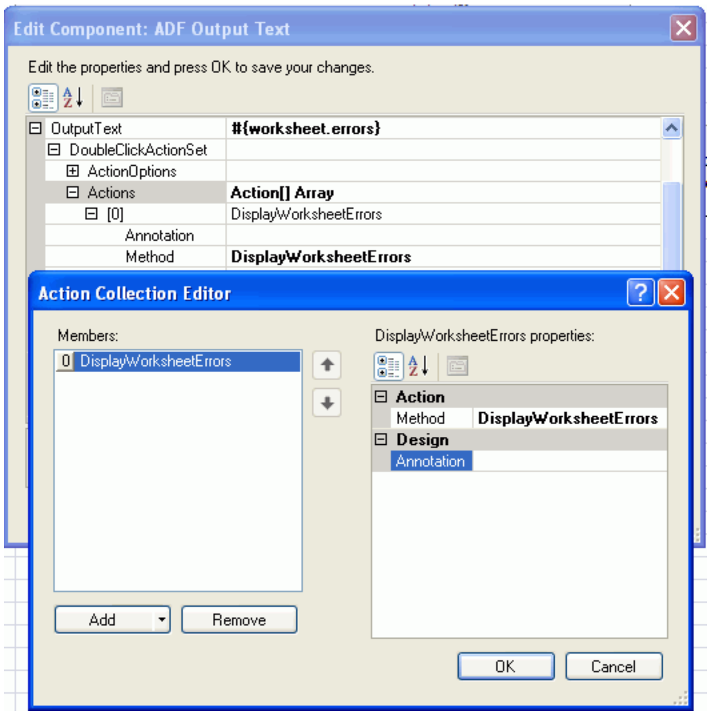
ADF Desktop Integration provides actions that display error details generated by an ADF Table component or an integrated Excel worksheet.

The action set in which you invoke one of these actions must include only one action. In general, action sets clear error labels and message lists when invoked. An action set that invokes one of the following actions returns error labels and message lists to the end user:

- Worksheet's `DisplayWorksheetErrors` action

  To display a worksheet's error messages, configure the action set of a component on the worksheet or the worksheet ribbon button to invoke this action. For example, Figure 12–2 shows the Action Collection Editor dialog configuring the `DisplayWorksheetErrors` action as a `DoubleClickActionSet` item for an ADF Output Text component on the worksheet.

Figure 12–2 DisplayWorksheetErrors Action



Edit Component: ADF Output Text

Edit the properties and press OK to save your changes.

OutputText	#{worksheet.errors}
DoubleClickActionSet	
ActionOptions	
Actions	Action[] Array
[0]	DisplayWorksheetErrors
Annotation	
Method	DisplayWorksheetErrors

Action Collection Editor

Members:

0 DisplayWorksheetErrors

DisplayWorksheetErrors properties:

Action	
Method	DisplayWorksheetErrors
Design	
Annotation	

At runtime, double-clicking the ADF OutputText component invokes the `DisplayWorksheetErrors` action as shown in Figure 12–3.

Figure 12–3 Runtime View of DisplayWorksheetErrors action

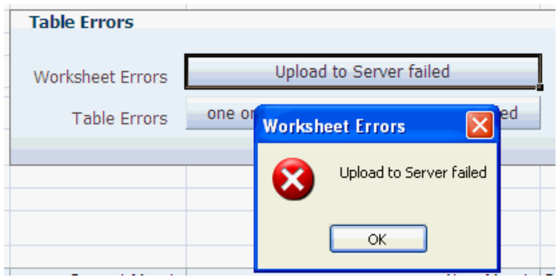


Table Errors

Worksheet Errors	Upload to Server failed
Table Errors	one or ... ed

Worksheet Errors

Upload to Server failed

For more information about the Worksheet's `DisplayWorksheetErrors` action, see Section A.13, "Worksheet Actions and Properties."

- ADF Table component's `DisplayRowErrors` action

  To display row-level failures that occur in an ADF Table component, invoke this action. Row-level failures occur when end user invokes the following actions:

  – `Upload`

  – `DeleteFlaggedRows`

  – `DoubleClickActionSet` invoked from an ADF Table component column

  For more information about using this action, see Section 12.5, "Providing a Row-by-Row Status on an ADF Table Component."

- ADF Table component's `DisplayTableErrors` action

To display table-level failures that occur in an ADF Table component, invoke this action. It is not intended that an ADF Table component column's `DoubleClickActionSet` invoke this action. Instead add this action to an action set that returns error messages to end users when failures occur during invocation of the action binding specified by an ADF Table component's `BatchOptions.CommitBatchActionID` property.

At runtime, double-clicking the ADF OutputText component invokes the `DisplayTableErrors` action as shown in Figure 12–4.

Figure 12–4 Runtime View of DisplayTableErrors action

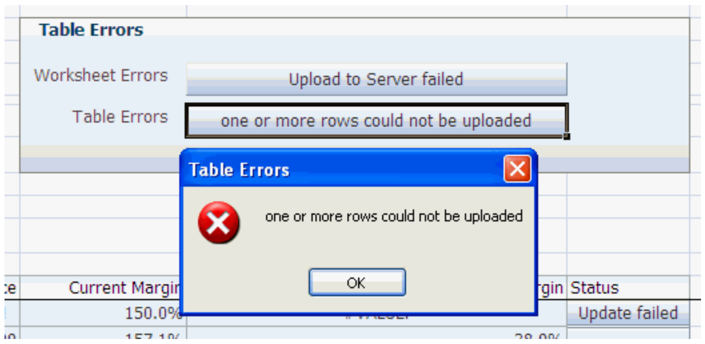


Table Errors

Worksheet Errors: Upload to Server failed

Table Errors: one or more rows could not be uploaded

Table Errors

one or more rows could not be uploaded

OK

For more information about ADF Table component actions, see Section A.9, "ADF Table Component Properties and Actions."

12.5 Providing a Row-by-Row Status on an ADF Table Component

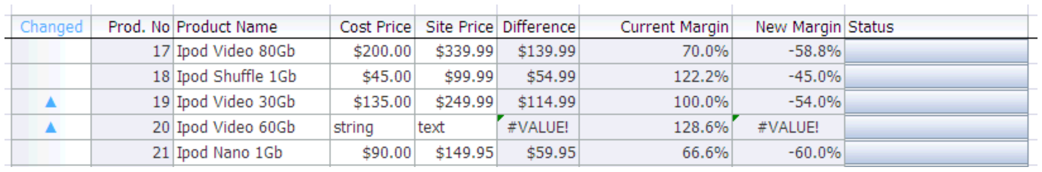
The ADF Table component provides a mechanism to indicate to end users whether rows from the ADF Table component have been processed successfully or not after invocation of following ADF Table component actions:

- `DeleteFlaggedRows`
- `Upload`
- `DoubleClickActionSet` invoked from an ADF Table component's column

The ADF Table component populates the `_ADF_StatusColumn` column with the status for each row following the invocation of the ADF Table component action. For example, it populates the `_ADF_StatusColumn` column with the upload status for each row following the invocation of the ADF Table component's `Upload` action.

Figure 12–5 shows rows in an ADF Table component where the values in those rows have been changed, as indicated by the upward pointing arrows in the **Changed** column. In the **CostPrice** column, two string values have been entered where a number value is expected.

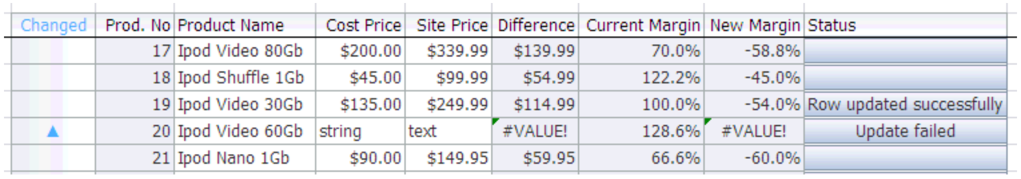
Figure 12–5 ADF Table Component with Changed Rows Before Upload



Changed	Prod. No	Product Name	Cost Price	Site Price	Difference	Current Margin	New Margin	Status
	17	Ipod Video 80Gb	$200.00	$339.99	$139.99	70.0%	-58.8%	
	18	Ipod Shuffle 1Gb	$45.00	$99.99	$54.99	122.2%	-45.0%	
▲	19	Ipod Video 30Gb	$135.00	$249.99	$114.99	100.0%	-54.0%	
▲	20	Ipod Video 60Gb	string	text	#VALUE!	128.6%	#VALUE!	
	21	Ipod Nano 1Gb	$90.00	$149.95	$59.95	66.6%	-60.0%	

Figure 12–6 shows the same rows in the ADF Table component after invocation of the ADF Table component's `Upload` action. The ADF Table component populates the `_ADF_StatusColumn` column (labeled **Status** in this example at runtime) with a message indicating whether the row updated successfully or not.

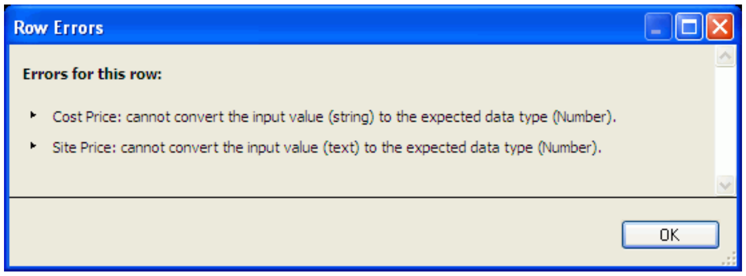
Figure 12–6 ADF Table Component with Changed Rows After Upload



Changed	Prod. No	Product Name	Cost Price	Site Price	Difference	Current Margin	New Margin	Status
	17	Ipod Video 80Gb	$200.00	$339.99	$139.99	70.0%	-58.8%	
	18	Ipod Shuffle 1Gb	$45.00	$99.99	$54.99	122.2%	-45.0%	
	19	Ipod Video 30Gb	$135.00	$249.99	$114.99	100.0%	-54.0%	Row updated successfully
▲	20	Ipod Video 60Gb	string	text	#VALUE!	128.6%	#VALUE!	Update failed
	21	Ipod Nano 1Gb	$90.00	$149.95	$59.95	66.6%	-60.0%	

By default, the `_ADF_StatusColumn` column's `DoubleClickActionSet` is configured to invoke the ADF Table component's `DisplayRowErrors` action. When end users double-click a row in this column at runtime, the ADF Table component invokes the `DisplayRowErrors` action. This action displays a dialog with a list of errors for that row if errors exist. If no errors exist, the dialog displays a message to indicate that no errors occurred. Figure 12–7 shows the dialog that appears if the end user double-clicks the cell in Figure 12–6 that displays `Update failed` in the **Status** column.

Figure 12–7 Dialog Displaying Row Error Message



Errors for this row:

- Cost Price: cannot convert the input value (string) to the expected data type (Number).
- Site Price: cannot convert the input value (text) to the expected data type (Number).

For more information about the `_ADF_StatusColumn` column, see Section 7.11, "Special Columns in the ADF Table Component."

12.6 Adding Detail to Error Messages in an Integrated Excel Workbook

You can configure your Fusion web application to report errors using a custom error handler to provide more detail to the error messages displayed to end users in an integrated Excel workbook.

To implement this functionality, the custom error handler must override the `getDetailedDisplayMessage` method to return a `DCErrorMessage` object. At runtime, ADF Desktop Integration detects the custom error handler and invokes the `getHtmlText` method on the `DCErrorMessage` object. ADF Desktop Integration includes the HTML returned by the `getHtmlText` method in the error message list as detail.

For more information about creating a custom error handler, see the "Customizing Error Handling" section of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

12.7 Handling Data Conflicts When Uploading Data from a Workbook

If one of your end users (John) makes changes to a row of data that he downloaded from a Fusion web application to an Excel workbook and another end user (Jane) in a different session modifies the same row in the Fusion web application after John downloads the row, John may encounter an error when he attempts to upload the modified row, as his changes conflict with those that Jane made. Depending on the configuration of your Fusion web application, John may receive `RowInconsistentException` type error messages. For information about how to configure your Fusion web application to protect your data, see the "How to Protect Against Losing Simultaneously Updated Data" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

To resolve this conflict in the integrated Excel workbook, John needs to download the most recent version of data from the Fusion web application. However, invoking the ADF Table component's `Download` action causes the component to refresh all data that the component hosts in the Excel workbook. This may overwrite other changes that John made that do not generate conflict error messages. To resolve this scenario, you can expose the ADF Table component's `DownloadFlaggedRows` action. When invoked, this action downloads data only for the rows that the end user flags for download. Using this action, John can resolve the conflict issues and upload his modified data.

Chapter 15, "Using an Integrated Excel Workbook Across Multiple Web Sessions and in Disconnected Mode" provides information about using an integrated Excel workbook across multiple sessions. For information about flagging rows, see Section 7.10.2, "Row Flagging in an ADF Table Component." For information about invoking component actions, see Section 8.2.2, "How to Invoke Component Actions in an Action Set." For more information about the components that the ADF Table component supports, see Section A.9, "ADF Table Component Properties and Actions."

12.7.1 How to Configure a Workbook to Handle Data Conflicts When Uploading Data

You specify a row-specific attribute of the tree binding for the `RowData.ChangeIndicatorAttribute` property to determine whether a row has been modified by another user since the row was last downloaded by the ADF Table component.

To configure a workbook to handle data conflicts:

1. Open the integrated Excel workbook.
2. Select the cell in the Excel worksheet that references the ADF Table component and click **Edit Properties** in the **Oracle ADF** tab to display the Edit Component: ADF Table dialog.
3. For the `RowData.ChangeIndicatorAttribute` property, specify the row-specific attribute of the tree binding that you use to determine whether a row has been modified by another user since the row was last downloaded by the ADF Table component in your integrated Excel workbook.
4. Click **OK**.

12.7.2 What Happens at Runtime When You Configure a Workbook to Handle Data Conflicts

The ADF Table component caches the original value of the row-specific attribute of the tree binding that you specified as a value for

`RowData.ChangeIndicatorAttribute` when it invokes the `RowDownSync` action. When the ADF Table component invokes the `RowUpSync` action, it checks if the value of the binding hosted by the Fusion web application and the original value cached by the ADF Table component differ. If they differ, it indicates data conflict, as changes have been made to the value of the binding hosted by the Fusion web application since the ADF Table component downloaded the value of the binding.

13 Testing Your Integrated Excel Workbook

This chapter describes features in ADF Desktop Integration that help you test your integrated Excel workbook as you configure it. It includes the following sections:

- Section 13.1, "Introduction to Testing Your Integrated Excel Workbook"
- Section 13.2, "Testing Your Fusion Web Application"
- Section 13.3, "Testing Your Integrated Excel Workbook"

13.1 Introduction to Testing Your Integrated Excel Workbook

Testing an integrated Excel workbook before you publish and deploy it to your end users enables you to verify that the functionality you configure behaves as you intend. Before you test your integrated Excel workbook, test the Fusion web application with which you integrate the Excel workbook. Once your Fusion web application functions as you intend, use the test mode provided by ADF Desktop Integration to test the functionality in your integrated Excel workbook.

13.2 Testing Your Fusion Web Application

Test the Fusion web application that you integrate your Excel workbook with before you start testing the integrated Excel workbook. For information about testing a Fusion web application, see the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. Verify that the Fusion web application you want to integrate an Excel workbook with, supports ADF Desktop Integration by carrying out the procedure described in Section C.1, "Verifying That Your Fusion Web Application Supports ADF Desktop Integration."

There are some differences between the test mode and the runtime mode when you run the integrated Excel workbook. Table 13–1 lists these differences.

Table 13–1 Differences between Test mode and Runtime mode

Test mode	Runtime mode
Does not perform tamper check	Performs tamper check
Does not display the connection confirmation dialog	Displays the connection confirmation dialog
Displays the Oracle ADF ribbon tab	Does not display Oracle ADF tab
Allows you to switch back to design mode	Does not allow you to switch back to design mode

Before you run the Fusion web application in JDeveloper, ensure that you have closed all integrated Excel workbooks and the Excel application. The application deployment may fail if it encounters locked files as Excel locks the files that it opens.

> **Tip:** If you plan to test integrated Excel workbooks that you downloaded from web pages of the Fusion web application, you should republish them before redeploying the application. Republishing the workbooks ensures that you have their latest versions.

If you make changes to the Fusion web application to resolve problems identified by testing the application, you need to:

- Close Excel and all integrated Excel workbooks. The application deployment may fail if it encounters locked files, as Excel locks the files that it opens.
- Rebuild the JDeveloper project where you develop the Fusion web application.
- Run the Fusion web application.
- Reload the page definition files that are associated with the integrated Excel workbook. Click the **Refresh Bindings** button in **Oracle ADF** tab of the integrated Excel workbook to reload the page definition files.

These steps make sure that the changes in the Fusion web application are available to the integrated Excel workbook. For information about how to reload a page definition file, see Section 4.3.3, "Reloading a Page Definition File in an Excel Workbook."

Server Ping Test

The server ping test enables you to check the version of ADF Desktop Integration Remote Servlet in a running system. It also helps to confirm that the remote servlet is loaded and responding.

After running the Fusion web application and logging in as a valid user, open a URL in the following format to verify whether the remote servlet is running:

```
http://<hostname>:<portnumber>/<context-root>/adfdiRemoteServlet
```

For example, if you run the Master Price List Fusion web application, open the following URL:

```
http://127.0.0.1:7101/FODMasterPriceList/adfdiRemoteServlet
```

The following response verifies that the remote servlet is running:

```
Oracle ADF 11g Desktop Integration (11.1.1.55.30) [1738]

Response from
oracle.adf.desktopintegration.servlet.DIRemoteServlet: OK.
```

In the above example, the remote servlet version is `11.1.1.55.30` and the ADF Desktop Integration version corresponding to the remote servlet is `1738`.

> **Note:** A valid user session is required to run the server ping test. If authentication is enabled for the web application, you will be prompted for valid credentials to log in.

13.3 Testing Your Integrated Excel Workbook

As you configure your Excel workbook to integrate with a Fusion web application, you can switch to test mode from design mode to test the functionality that you add to the workbook. You use the **Oracle ADF** tab to switch to test mode from design mode and from design mode to test mode.

Test mode enables you to test the functionality of your integrated Excel workbook as you configure it incrementally. It also enables you to view the integrated Excel workbook from the end user's perspective, as test mode corresponds to what end users see when they view and execute the published integrated Excel workbook. The difference between an integrated Excel workbook in test mode and a published integrated Excel workbook is that the ADF Desktop Integration task pane is not available to users of the published integrated Excel workbook.

For more information about test mode and design mode, see Section 5.1, "Introduction to Development Tools."

ADF Desktop Integration can generate log files that capture information based on events triggered by an integrated Excel workbook. For more information about these log files, see Appendix C, "Troubleshooting an Integrated Excel Workbook."

Note: Before you start testing the integrated Excel workbook, ensure that:

- The Fusion web application is running
- The ping to server is successful, and the server is configured for ADF Desktop Integration

---

To run an integrated Excel workbook in test mode:

- To test and run an integrated Excel workbook, click the **Run** button on the **Oracle ADF** tab.

  The integrated Excel workbook switches to test mode from design mode.

To stop test mode and return the integrated Excel workbook to design mode:

- In the integrated Excel workbook that you are testing, click the **Stop** button on the Oracle ADF tab.

  The integrated Excel workbook switches to design mode from test mode.

Note: When the end user tries to close the integrated Excel workbook, Microsoft Excel prompts a dialog to save the workbook even if the end user has not modified it after opening it. This behavior is expected because ADF Desktop Integration modifies an integrated Excel workbook each time the end user opens it.

---

14 Deploying Your Integrated Excel Workbook

This chapter describes how to deploy a workbook that you have integrated with a Fusion web application to your end users after you have finalized the integration.

This chapter includes the following sections:

- Section 14.1, "Introduction to Deploying Your Integrated Excel Workbook"
- Section 14.2, "Making ADF Desktop Integration Available to End Users"
- Section 14.3, "Publishing Your Integrated Excel Workbook"
- Section 14.4, "Deploying a Published Workbook with Your Fusion Web Application"
- Section 14.5, "Passing Parameter Values from a Fusion Web Application Page to a Workbook"

14.1 Introduction to Deploying Your Integrated Excel Workbook

After you finish development of your integrated Excel workbook, you make the final integrated Excel workbook available to end users by deploying the resulting Fusion web application to an application server. Before you deploy a finalized Excel workbook that integrates with the Fusion web application, you must publish it as described in Section 14.3, "Publishing Your Integrated Excel Workbook." After you have published the Excel workbook, you can deploy it using one of the methods outlined in the "Deploying Fusion Web Applications" chapter of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

The end users that you deploy an integrated Excel workbook to must do the following:

- Set up ADF Desktop Integration on their machines.

  Make the ADF Desktop Integration `setup.exe` tool available to end users from, for example, a directory on your network. For more information, see Section 14.2, "Making ADF Desktop Integration Available to End Users."

- If required, configure the security settings for their Excel application.

14.2 Making ADF Desktop Integration Available to End Users

End users who want to use the functionality that you configure in an integrated Excel workbook must install the Runtime edition of ADF Desktop Integration. The `setup.exe` tool is located in the `adfdi-excel-runtime-client-installer.zip` file available in *`MW_HOME`*`\oracle_common\modules\oracle.adf.desktopintegration_11.1.1` directory, where *`MW_HOME`* is the Middleware Home directory.

For information about using the `setup.exe` tool, see Section I.1, "Installing the Runtime Edition of ADF Desktop Integration."

For more information about Microsoft ClickOnce installer, see the following:

`http://msdn.microsoft.com/en-us/library/71baz9ah.aspx`

14.3 Publishing Your Integrated Excel Workbook

After you finish configuring the Excel workbook with Oracle ADF functionality, you must publish it. Publishing a workbook makes it available to the end users for whom you configured the integrated Excel workbook.

ADF Desktop Integration also provides you with two methods to publish your workbook. You can publish your integrated Excel workbook directly from Excel, or you can use the publish tool available in JDeveloper to publish the workbook from the command line. The command-line publish tool enables you to use ANT build scripts to publish an integrated Excel workbook from your Fusion web application.

14.3.1 How to Publish an Integrated Excel Workbook from Excel

You publish a workbook by clicking a button on the **Oracle ADF** tab and specifying values in the dialogs that appear, or by using the command-line publish tool. You can use the command line publish tool to publish a workbook from your Fusion web application.

To publish a workbook from Excel:

1. Open the integrated Excel workbook.
2. Ensure that the `ApplicationHomeFolder` and `WebPagesFolder` properties in the Edit Workbook Properties dialog are correct. If these properties are not set, ADF Desktop Integration prompts you to set them when you publish the integrated Excel workbook.

   For more information, see Section 4.4.3, "How to Configure a New Integrated Excel Workbook."
3. In the **Oracle ADF** tab, click the **Publish** button.
4. Specify the directory and file name for the published workbook in the Publish Workbook dialog that appears. The directory and file name that you specify for the published workbook must be different from the directory and file name for the design time workbook.
5. Click **Save** to save changes.

14.3.2 How to Publish an Integrated Excel Workbook Using the Command Line Publish Tool

The publish tool is run from the command line, and is available in the *`MW_HOME`*`\jdeveloper\adfdi\bin\excel\tools\publish` directory as `publish-workbook.exe`. Before you run the publish tool, open the source integrated Excel workbook and ensure that the `ApplicationHomeFolder` and `WebPagesFolder` properties in the Edit Workbook Properties dialog are correct.

Now, run the publish tool using the following syntax:

```
publish-workbook -workbook (-w) <source-workbook-path> -out (-o)
<destination-workbook-path>
```

where `source-workbook-path` is the path of sthe ource workbook, and `destination-workbook-path` is the path where the published workbook is saved. Note that the destination workbook cannot have the same name as the source, even if the directory locations are different.

For example:

```
publish-workbook -workbook
D:\Application1\Project1\ViewController\src\oracle\foddemo\maste
rpricelist\excel\workbook-src.xlsx -out
D:\Application1\Project1\ViewController\public_
html\excel\published\workbook.xlsx
```

> **Tip:** For more information about the arguments required by the publish tool, run the following command:
>
> ```
> publish-workbook -help (-h)
> ```

After publishing the integrated Excel workbook successfully, the publish tool displays a success message. If there is any error while publishing the workbook, the publish tool aborts the process and the error messages are displayed on the command line console.

If you are using the command line publish tool, note that by default the publish tool logs messages to the command line console at information level.

Using the Publish Tool with ANT

You can create ANT scripts to run the publish tool from JDeveloper when you build your Fusion web application. You can use either of the following methods to run the utility using ANT:

- Generate an ANT build script for the project and add a target to run the workbook command line publish tool
- Generate or create a separate ANT build script for running the workbook command line publish tool

A sample ANT build script (`publish-workbook.xml`) to run the publish tool is available in the `*MW_HOME*\jdeveloper\adfdi\bin\excel\samples` directory. The sample ANT script demonstrates the invocation of the command-line workbook publishing tool.

14.3.3 What Happens When You Publish an Integrated Excel Workbook

When you click the **Publish** button in design mode, ADF Desktop Integration performs the following actions:

- Removes binding expressions that are visible in the worksheet while the workbook is in design mode.
- Changes the workbook mode to runtime mode.
- Clears the `ApplicationHomeFolder`, `WebAppRoot`, and `WebPagesFolder` properties from the workbook settings of the published workbook.
- Creates the published workbook with the file name you specified in the directory that you specified.
- Updates the client registry. For more information, see Section 11.3, "Checking the Integrity of an Integrated Excel Workbook's Metadata."

14.4 Deploying a Published Workbook with Your Fusion Web Application

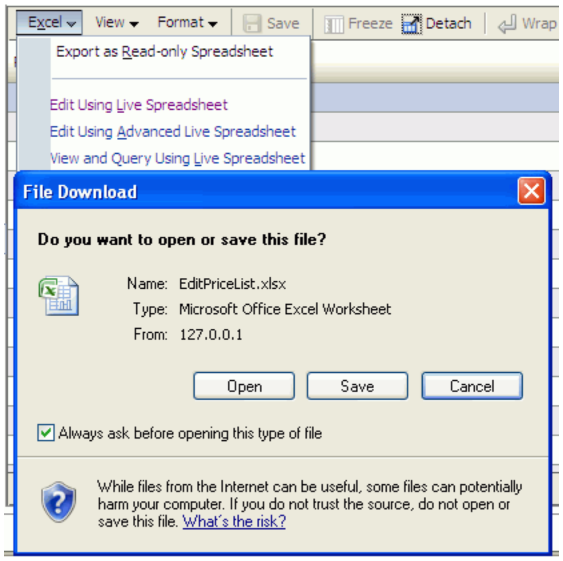
Add the integrated Excel workbook to the JDeveloper project for your Fusion web application if it is not packaged with the other files that constitute your JDeveloper project. This makes sure that the Excel workbooks you integrate with your Fusion web application get deployed when you deploy your finalized Fusion web application. For example, the Master Price List module stores the Excel workbooks that it integrates at the following location:

```
FOD_
HOME\MasterPriceList\ViewController\src\oracle\foddemo\masterpri
celist\excel\excel
```

where `FOD_HOME` is the installation directory for the Fusion Order Demo application.

After you decide on a location to store your integrated Excel workbooks, you can configure web pages in your Fusion web application allowing end users to access the integrated Excel workbooks. For example, Figure 14–1 shows Internet Explorer's File Download dialog, which was invoked by clicking the **Excel** > **Edit Using Live Spreadsheet** menu options on the `PriceListSummary.jspx` page displayed by the Master Price List module.

Figure 14–1 Invoking an Integrated Excel Workbook from a Fusion Web Application


Excel
View
Format
Save
Freeze
Detach
Wrap
Export as Read-only Spreadsheet
Edit Using Live Spreadsheet
Edit Using Advanced Live Spreadsheet
View and Query Using Live Spreadsheet
File Download
Open
Save
Cancel

Do you want to open or save this file?

Name: EditPriceList.xlsx
Type: Microsoft Office Excel Worksheet
From: 127.0.0.1

Always ask before opening this type of file

While files from the Internet can be useful, some files can potentially harm your computer. If you do not trust the source, do not open or save this file. What's the risk?

To enable the functionality illustrated in Figure 14–1, the HTTP filter parameters for your Fusion web application must be configured to recognize Excel workbooks. JDeveloper automatically configures these parameters for you when you add ADF Desktop Integration to the technology scope of your Fusion web application, as explained in Section 4.2, "Adding ADF Desktop Integration to a Fusion Web Application." If you want to manually configure the HTTP filter parameters, see Appendix E, "ADF Desktop Integration Settings in the Web Application Deployment Descriptor."

After you have configured the HTTP filter for your Fusion web application, you configure the web pages that the Fusion web application displays to end users to allow them to invoke Excel workbooks. A basic method of invoking an Excel workbook that you have integrated with a Fusion web application is to provide a hyperlink that invokes the workbook. For example, you could write the following HTML in a web page:

```
<a href="/excel/EditPriceList.xlsx">Open the Master Price List
in Excel</a>
```

where `excel` is a subdirectory of the directory specified by the `WebPagesFolder` workbook property and `EditPriceList.xlsx` is the Excel workbook that the end user invokes.

You can provide functionality that allows end users to invoke Excel workbooks from buttons, lists and ribbon command buttons. The following list provides some examples:

- Button

  Display a button on the web page that, when clicked, invokes the integrated Excel workbook.

- Selection list

  Use the ADF Faces `selectOneChoice` component with a button to invoke an integrated Excel workbook.

- Menu

  Use the ADF Faces `goMenuItem` component.

  The **View and Query Using Live Spreadsheet** menu, as illustrated in Figure 14–1, uses the `goMenuItem` component. The `PriceListSummary.jspx` page displays this menu. The following entry appears in the `PriceListSummary.jspx` page of the Master Price List module and demonstrates the `goMenuItem` component:

  ```
  <af:goMenuItem id="goReadOnly"
  textAndAccessKey="#{res['pls.productList.menu.lsr.label']}"
  destination="/excel/published/ReadOnlyPriceList.xlsx"/>
  ```

For more information about creating web pages for a Fusion web application, see the *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

14.5 Passing Parameter Values from a Fusion Web Application Page to a Workbook

You can configure a page in your Fusion web application to pass parameter values to an integrated Excel workbook when the end user downloads the workbook from the page. For example, if the end user attempts to download a workbook from a page that displays a list of products, the list of products that appears in the workbook corresponds to the list of products displayed in the page when the end user invoked the download. Subsequent changes that the end user makes to data in one location (the worksheet or the Fusion web application's page) do not affect data in the other location.

To configure this functionality, you must:

- Verify that the HTTP filter is configured to allow end users to download integrated Excel workbooks from the Fusion web application. By default, JDeveloper configures the HTTP filter with appropriate values when you add ADF Desktop Integration to the technology scope of your Oracle ADF Desktop Integration project. To verify the parameter values of the HTTP filter, see Section E.2, "Configuring the ADF Desktop Integration Excel Download Filter."
- Configure the page in your Fusion web application from which the end user downloads the integrated Excel workbook so that it passes its parameters through

URL arguments to the integrated Excel workbook when the end user downloads it.

- Configure the page definition file associated with the worksheet in the integrated Excel workbook so that the worksheet is initialized with the parameters from the page in the Fusion web application from which the end user downloads the workbook.
- Configure workbook and worksheet properties in the integrated Excel workbook that end users download so that the workbook contains the parameters from the page in the Fusion web application from which the end invokes download.

14.5.1 How to Configure the Fusion Web Application's Page to Pass Parameters

You insert an `<af:goLink>` tag and specify property values for it that reference the integrated Excel workbook the end user downloads and the values to download. You also specify the commands on the page that, when invoked, require the Fusion web application to refresh the values referenced by the `<af:goLink>` tag and its property values.

To configure the page in the Fusion web application:

1. In JDeveloper, insert the `af:goLink` tag into the page from which the end user downloads the integrated Excel workbook.
2. In the Structure window, right-click the **af:goLink** node and choose **Go to Properties**.
3. Expand the **Common** section and set values for the properties, as described in Table 14–1.

Table 14–1 Properties for af:goLink Tag

Property	Value
`Text`	Write the text that appears to end users at runtime. For example, write text such as the following to appear at runtime: `Download to Excel`

Table 14–1 (Cont.) Properties for af:goLink Tag

Property	Value
`Destination`	Invoke the expression builder to write an EL expression that specifies the integrated Excel workbook and the values to download as a URL argument: For example, write an EL expression such as the following: `"/excel/workbook.xlsx?productName=#{bindings.productName.attributeValue}"` Note that the runtime URL-encoded value of the EL expression to the right of `?` must be less than 2048 bytes. If the runtime value exceeds 2048 bytes, the integrated Excel workbook downloads the URL arguments in the first 2048 bytes. Subsequent URL arguments do not get downloaded to the integrated Excel workbook. Instead, the Fusion web application writes log entries for these URL arguments identifying them as having not been downloaded. For example, the runtime URL-encoded value of `productName=#{bindings.productName.attributeValue}` must be less than 2048 bytes. Also note that if the URL contains more than 256 characters, an exception is raised when the end user downloads and opens the integrated Excel workbook without saving it. To resolve this problem, you must limit your URL length to 256 characters, or instruct the end user to save the workbook before opening it.

4. Optionally, expand the **Behavior** section and specify component IDs for the `partialTriggers` property that, when invoked, update the values of the `af:goLink` tag and its `Destination` property.

   For example, if you have navigation buttons with the IDs `NextButton`, `PreviousButton`, `FirstButton`, and `LastButton`, specify them as follows:

   `:NextButton :PreviousButton :FirstButton :LastButton`

5. Save the page.

   The following example shows the entries that JDeveloper generates in a JSF page using the examples in this procedure:

```
<af:goLink text="Download to Excel"
destination="/excel/workbook.xlsx?productName=#{bindings.productName.attributeV
alue}"
partialTriggers=":NextButton :PreviousButton :FirstButton :LastButton"/>
```

14.5.2 How to Configure the Page Definition File for the Worksheet to Receive Parameters

You configure the page definition file associated with the worksheet in the integrated Excel workbook as follows:

- Add one or more `parameter` elements that initialize the worksheet with the values specified by the workbook `Parameters` property that you configure in Section 14.5.3, "How to Configure Parameters Properties in the Integrated Excel Workbook."

  The following example shows a parameter element in a page definition file that is associated with a worksheet in an integrated Excel workbook:

```
<parameters>
```

```
    <parameter id="ProductNameParam" />
</parameters>
```

- Add an `invokeAction` and a method action binding so that the page definition file associated with the worksheet initializes correctly.

  The following example shows the `initializeProductTable` `invokeAction` invoking the `filterByProductName` method action binding. The `invokeAction` is refreshed only when a value for `ProductNameParam` is supplied.

```
<executables>
    <invokeAction Binds="filterByProductName" id="initializeProductTable"
                  Refresh="deferred"
                  RefreshCondition="${bindings.ProductNameParam != null}"/>
...
</executables>
```

  The method action binding invokes a view object method (`filterByProductName`). The view object method takes a single String argument (`ProductNameArg`) that references the value of `ProductNameParam`.

```
<bindings>
    <methodAction id="filterByProductName" RequiresUpdateModel="true"
                  Action="invokeMethod" MethodName="filterByProductName"
                  IsViewObjectMethod="true" DataControl="AppModuleDataControl"
                  InstanceName="AppModuleDataControl.ProductVO1">
      <NamedData NDName="ProductNameArg" NDValue="${bindings.ProductNameParam}"
                  NDType="java.lang.String"/>
    </methodAction>
. . .
</bindings>
```

For more information about configuring a page definition file, see Section 4.3, "Working with Page Definition Files for an Integrated Excel Workbook."

14.5.3 How to Configure Parameters Properties in the Integrated Excel Workbook

You configure the workbook `Parameters` property and the worksheet `Parameters` property so that the integrated Excel workbook that the end user downloads from the Fusion web application receives parameter values included in the query string of the workbook download URL.

To configure the workbook Parameters property:

1. Open the integrated Excel workbook.
2. Click **Workbook Properties** in the **Oracle ADF** tab.
3. Click the ellipsis button (**...**) beside the input field for **Parameters** to invoke WorkbookParameter Collection Editor.
4. Click **Add** to add a new workbook initialization parameter and configure its properties as follows:
   - (Optional) In the **Annotation** field, enter a description of the workbook initialization parameter.
   - In the **Parameter** field, specify the name of the URL argument that you configured for the `af:goLink` tag's `Destination` property as described in Section 14.5.1, "How to Configure the Fusion Web Application's Page to Pass Parameters."

5. Repeat Step 4 as necessary to add other workbook initialization parameters.
6. Click **OK**.

   For more information about the workbook `Parameters` property, see Table A–18.

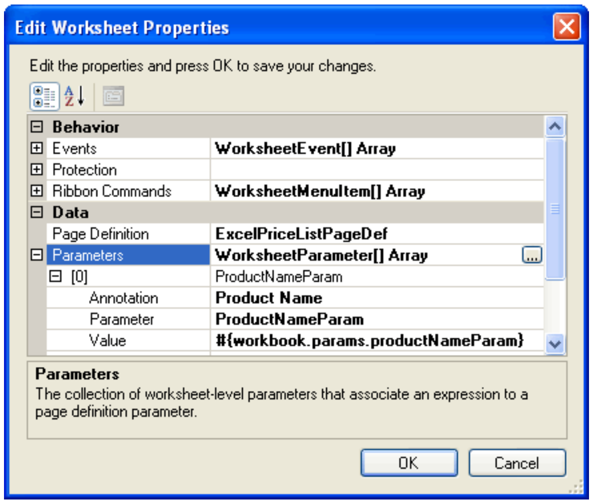
To configure the worksheet Parameters property:

1. Open the integrated Excel workbook.
2. Click **Worksheet Properties** in the **Oracle ADF** tab.
3. Click the ellipsis button (**...**) beside the input field for **Parameters** to invoke the WorksheetParameter Collection Editor.
4. Click **Add** to add a new worksheet parameter and configure it as in Figure 14–2:
   - (Optional) In the **Annotation** field, enter a description of the worksheet parameter.
   - In the **Parameter** field, specify a parameter element that you added to the page definition file associated with the worksheet, as described in Section 14.5.2, "How to Configure the Page Definition File for the Worksheet to Receive Parameters."
   - In the **Value** field, write an EL expression that references the value of the `Parameter` property you specified for the workbook initialization parameter (workbook `Parameters` array). Use the following syntax when writing the EL expression:

     `#{workbook.params.`*`productName`*`}`

     where *`productName`* references the value of the `Parameter` property you specified for the workbook initialization parameter.

Figure 14–2 Worksheet Parameters



Edit the properties and press OK to save your changes.

⊟ **Behavior**	
⊞ Events	**WorksheetEvent[] Array**
⊞ Protection	
⊞ Ribbon Commands	**WorksheetMenuItem[] Array**
⊟ **Data**	
Page Definition	**ExcelPriceListPageDef**
⊟ Parameters	**WorksheetParameter[] Array**
⊟ [0]	ProductNameParam
Annotation	**Product Name**
Parameter	**ProductNameParam**
Value	**#{workbook.params.productNameParam}**

Parameters
The collection of worksheet-level parameters that associate an expression to a page definition parameter.

5. Repeat Step 4 as necessary to add other workbook initialization parameters.
6. Click **OK**.

   For more information about the worksheet `Parameters` property, see Table A–19.

By default, the workbook parameters are not sent every time the workbook connects to the server to request metadata, the end user logs out, or the session expires. If required, you can configure the workbook to send the initialization parameters by configuring the `SendParameters` property.

To configure the worksheet SendParameters property:

1. Open the integrated Excel workbook.
2. Click **Worksheet Properties** in the **Oracle ADF** tab.
3. In the property inspector, set the value of `SendParameters` as shown in the following table and Figure 14–2:

Set this property to...	This value...
`SendParameters`	`True` to send workbook parameters when the workbook connects to the server to request metadata or data. When set to `True`, parameters are sent everytime when the metadata is requested and the first time when data is requested, during each user session. `False` is the default value. For more information, see Section 15.2, "Restore Server Data Context Between Sessions."

4. Click **OK**.

14.5.4 What Happens at Runtime When a Fusion Web Application Page Passes Parameters to an Integrated Excel Workbook

When the end user downloads the integrated Excel workbook from the Fusion web application, the `af:goLink` tag is evaluated and the current product name is captured and included on the URL. The `adfdiExcelDownload` filter embeds the names and values of all the parameters from the URL into the downloaded integrated Excel workbook.

After downloading the workbook, when the end user opens it for the first time, the active worksheet of the integrated Excel workbook is initialized. The initialization process includes fetching metadata from the web application. As part of retrieving the worksheet metadata, the stored workbook parameters (if any) are sent to the ADF Desktop Integration remote servlet and are available for application logic such as `<invokeAction>` executables. Specifically, the parameters are set into `BindingContainer DCParameters` before the binding container is refreshed. The action set in the worksheet `Startup` event is also executed during initialization. After initialization, the initialization status for each worksheet is recorded when the integrated Excel workbook is saved to disk.

After the integrated Excel workbook has been saved, closed, and reopened , the first-time initialization is skipped for any worksheets that were previously initialized. If workbook parameters were captured when the integrated Excel workbook was first downloaded, and those parameters are required to set up server context, then the `Worksheet.ServerContext.SendParameters` property should be set to `True`. When the `SendParameters` property is `True`, workbook parameters are sent on every request for metadata, and also on the first request for data in each user session.

To reset the initialization state for all worksheets in the workbook, invoke the `ClearAllData` action. For more information about the `ClearAllData` action, see Table A–17.

Note: Parameter values passed to the server might reset when a web dialog is invoked in an action set where the `ShareFrame` property is `True`. Custom code, which uses the parameters and requires that values be maintained across the invocation of a web dialog, should ensure that the values in the user session data structures are saved.

15 Using an Integrated Excel Workbook Across Multiple Web Sessions and in Disconnected Mode

This chapter describes the functionality that your end users can use when they are not connected to a Fusion web application. It also describes how to restore server data context when the end user connects to a Fusion web application through an integrated Excel workbook after having previously been disconnected from the application.

This chapter includes the following sections:

15.1 Introduction to Disconnected Workbooks

End users can open an integrated Excel workbook and log on to a Fusion web application from the workbook ribbon command button that you configure. The Fusion web application assigns a session to the user. After a connection to the Fusion web application is established and a valid session assigned, end users can download data from the Fusion web application to the workbook. They can then log off from the Fusion web application using the workbook ribbon command button or otherwise disconnect from the Fusion web application by, for example, disconnecting from the network that hosts the Fusion web application.

How the Fusion web application terminates the session assigned to the user depends on how the user disconnects from the Fusion web application. If the user logs off from the Fusion web application using a workbook command, the Fusion web application terminates the session immediately. If the user disconnects from the Fusion web application by some other means (for example, closing the workbook), the Fusion web application terminates the session assigned to the user after session timeout expires.

Functionality Available to End Users in an Integrated Excel Workbook When Disconnected from a Fusion Web Application

When end users are disconnected from the Fusion web application, they can perform the following actions:

- Modify data downloaded from the Fusion web application
- Insert new data into the appropriate ADF Table component contained in the workbook

- Save changes to data and close and reopen the workbook without having to upload data to the Fusion web application
- Track and update changes in the ADF Table component

Caching of Static Information in an Integrated Excel Workbook

Certain types of relatively static data are cached in the integrated Excel workbook to allow end users to use the workbook while disconnected from the Fusion web application. Table 15–1 describes the types of data that an integrated Excel workbook caches. It also describes when the integrated Excel workbook refreshes the data.

Table 15–1 Types of Data an Integrated Excel Workbook Caches

This type of data...	Is cached when...	And refreshed when...
Page definition metadata that is not runtime specific such as control binding types, IDs, and labels.	An integrated Excel worksheet bound to a page definition file is activated and no cache of the page definition file's metadata exists.	The page definition metadata is not refreshed unless you download a new copy of the integrated Excel workbook or invoke the workbook actions `ClearAllData` and `EditOptions` described in Table A–17.
ADF List of Values component list items	The ADF List of Values component first downloads the list items from the Fusion web application.	The values of the list items hosted by the Fusion web application differ from those cached by the integrated Excel workbook. The cached list items are refreshed only once per workbook session and only if a workbook session exists. Invoking the workbook actions `ClearAllData` and `EditOptions` described in Table A–17 also clears cached list items.
Resource bundle strings	The integrated Excel workbook is first initialized. A workbook is initialized when it is opened for the first time after conversion, or after `ClearAllData` is invoked.	The cache of resource bundle strings is not refreshed unless you download a new copy of the integrated Excel workbook or invoke the workbook actions `ClearAllData` and `EditOptions` described in Table A–17.

15.2 Restore Server Data Context Between Sessions

You must configure the page definition file so that the correct view object state is restored if the Fusion web application assigns the end user a new session after one of the following events occurs:

- The end user makes changes to data in a workbook, saves and closes the workbook, reopens the workbook at a later time, and attempts to upload the changes he or she made before saving and closing the workbook.
- The time between invocation of an ADF Table component's `Download` and `Upload` actions (or some other ADF Table component action that contacts the Fusion web application) exceeds the session timeout value specified for a Fusion web application session.

Both the scenarios described in the previous list involve two sessions. The first session is assigned when the end user opens an integrated Excel workbook and logs on to the Fusion web application. The Fusion web application terminates this session when the end user logs off from the Fusion web application or when the session expires. The Fusion web application assigns a second session when the end user reopens the integrated Excel workbook or invokes an action that interacts with the Fusion web application.

In addition to configuring the page definition file, configure the functionality in an integrated Excel workbook so that pending changes are not lost if the end user logs off from the Fusion web application or a session expires before changes are committed to the Fusion web application. For example, you configure the worksheet `Startup` event to invoke a `CreateInsert` action binding and a worksheet `DownSync` action. You also configure an ADF Button component labeled **Save** to invoke the worksheet `UpSync` action and the `Commit` action binding. If the end user's session ends, no record is saved even if the end user clicks the **Save** button after the Fusion web application assigns a new session. To prevent this scenario occurring, it is better to invoke the `CreateInsert` action binding from the ADF Button component labeled **Save**.

Another example is the behavior of the ADF Table component's `DownloadForInsert` action. If you create a custom method in the Fusion web application that creates temporary records to support the invocation by the ADF Table component of the `DownloadForInsert` action, make sure to remove these temporary records after successful invocation of the `DownloadForInsert` action. For more information about the use of the `DownloadForInsert` action, see Section 7.7, "Configuring a Worksheet to Download Pre-Insert Data to an ADF Table Component."

15.2.1 How to Configure an Integrated Excel Workbook to Restore Server Data Context

You specify the attribute bindings that you want to cache in an integrated Excel workbook between sessions as values for the worksheet's `ServerContext` group of properties. This group of properties also enables you to specify the action binding that uses the attribute binding data to restore server-side context when a Fusion web application assigns a new session to the integrated Excel workbook.

Before you can specify values for the `ServerContext` group of properties, the page definition file that is associated with the worksheet must expose the attribute bindings and action bindings for which you want to restore server context. For information about adding attribute bindings and action bindings to a page definition file, see Section 4.3, "Working with Page Definition Files for an Integrated Excel Workbook." For information about the `ServerContext` group of properties, see the entry for `ServerContext` in Table A–19.

To configure an integrated Excel workbook to restore server data context:

1. In the integrated Excel workbook, click **Worksheet Properties**.
2. In the property inspector that appears, configure values for the `ServerContext` group of properties as described by the following table:

Table 15–2 ServerContext Properties to Restore Server Data Context

For this property...	Enter or select this value...
`CacheDataContexts`	Add an element to the collection of `CacheDataContexts`. Configure the element you add as follows: ■ `RestoreDataContextActionID` Specify the action binding (for example, the `Execute` action binding) that connects to the Fusion web application to restore the data specified by `CachedServerContexts`. ■ `CachedServerContexts` An array that identifies the attribute binding values to cache and set before the action binding specified by `RestoreDataContextActionID` is invoked. Each element in the array (`CachedServerContext`) supports the `CachedAttributeID` and `RestoredAttributeID` properties. For more information about the `CacheDataContexts` property and its subproperties, see Section A–19, " Worksheet Properties."
`IDAttributeID`	Specify the attribute binding that uniquely identifies the row displayed in the current worksheet. At runtime, the value that this property references is used to determine if the server data context has been correctly restored. For more information about this property and its subproperties, see Section A–19, " Worksheet Properties."

If your integrated Excel workbook uses parameters and you have deployed it by downloading it from your Fusion web application, see Section 14.5.3, "How to Configure Parameters Properties in the Integrated Excel Workbook."

3. Click **OK**.

Note: For integrated Excel workbooks that use `Parameters` and `<invokeAction>` executables, you may not need to configure `RestoreDataContextActionID` and `CachedServerContexts`, if `Parameters` and `<invokeAction>` can restore server data context when a new session is created.

15.2.2 What Happens at Runtime When an Integrated Excel Workbook Restores Server Data Context

During the session that is assigned the initial session (for example, `session ID 1`), the worksheet caches data using the `ServerContext` group of properties. In a later session with a different session ID (for example, `session ID 2`), where the ADF Table component's `Upload` action is invoked, the data cached in the `ServerContext` group of properties is sent to the Fusion web application.

15.3 Caching Lists of Values for Use in Disconnected Mode

ADF Desktop Integration caches the values referenced by the ADF List of Values and the TreeNodeList subcomponents that you use to create lists of values and dependent lists of values so that these components do not send a request to the Fusion web application when the end user selects a value at runtime. For more information about using these components to create lists of values, see the following sections:

- Section 6.6, "Inserting an ADF List of Values Component"

- Section 7.13, "Creating a List of Values in an ADF Table Component Column"
- Section 8.8, "Creating Dependent Lists of Values in an Integrated Excel Workbook"

ADF Desktop Integration caches up to two hundred and fifty values for each component. If a component references a list of values with more than two hundred and fifty values, ADF Desktop Integration caches the first two hundred and fifty values and writes a warning message to the client-side log file for subsequent values. Consider configuring your integrated Excel workbook to invoke a pick dialog from a page in your Fusion web application where a list of values references more than two hundred and fifty values. For more information about client-side log files, see Section C.3, "Generating Log Files for an Integrated Excel Workbook." For more information about invoking a pick dialog from a Fusion web application page, see Section 8.4, "Displaying Web Pages from a Fusion Web Application" and Section 8.5, "Inserting Values in ADF Table Columns from a Web Page Pick Dialog."

Cached list of values in an integrated Excel workbook get refreshed once per workbook session. This refresh occurs after the user reestablishes a web session with the Fusion web application and if the values referenced by the Fusion web application have changed since the integrated Excel workbook last cached the list of values.

The upload of a selected value from a list of values causes the upload to fail if the selected value no longer exists in the Fusion web application. This may occur if, for example, one end user deletes the value in the Fusion web application while another end user modifies the selected value in the cached list of values of an integrated Excel workbook and attempts to upload the modified value to the Fusion web application. For more information about handling data conflict, see Section 12.7, "Handling Data Conflicts When Uploading Data from a Workbook."

Note that if you change the Fusion web application configuration after you have deployed the Fusion web application and the end users have started using the published integrated Excel workbooks, you must inform the end users to download a fresh copy of the integrated Excel workbook, or run the `ClearAllData` command. For more information about the `ClearAllData` action, see Table A–17

The changes in your Fusion web application might include changing the definitions of the list bindings associated with the ADF List of Values and TreeNodeList subcomponents exposed in the worksheet. Changing list binding metadata can cause unexpected exceptions in workbooks that have been downloaded and run prior to the change.

A
ADF Desktop Integration Component Properties and Actions

This appendix lists and describes the properties of ADF Desktop Integration components. It also describes the actions that certain components in this module expose.

This appendix includes the following sections:

A.1 Frequently Used Properties in the ADF Desktop Integration

Table A–1 lists alphabetically properties in ADF Desktop Integration that many components reference.

Table A–1 Frequently Used Properties in ADF Desktop Integration

Name	Type	EL	Description
`ActionSet`		N	For information about action sets, see Section A.11, "Action Set Properties."
`Annotation`	String	N	Use this field to enter a comment about the component's use in the worksheet. Comments you enter have no effect on the behavior of the workbook. They are the equivalent of code comments.
`ComponentID`	String	N	ADF Desktop Integration generates this string to uniquely identify each instance of an ADF component in an integrated Excel workbook.
`Label`	String	Y	Specify an EL expression that is evaluated at runtime. For information about EL expressions in ADF Desktop Integration, see Appendix B, "ADF Desktop Integration EL Expressions." For information about using labels, see Section 9.4, "Using Labels in an Integrated Excel Workbook."
`Position`		N	This property defines the upper-left corner of the Oracle ADF component in the integrated Excel workbook.
`ReadOnly`	Boolean	Y	Set this property to `TRUE` so that ADF Desktop Integration ignores changes a user makes to a cell that references a component which uses this property. This property is independent of Excel's workbook and worksheet protection functionality. Setting `ReadOnly` to `TRUE` does not prevent a user from modifying a cell. When `TRUE`, the behavior for cells that reference Oracle ADF components is as follows: ■ ADF Desktop Integration overwrites changes without warning when a worksheet is refreshed. ■ No changes are sent to the Fusion web application when the integrated Excel workbook is synchronized with the Fusion web application. To avoid end user confusion, apply styles to the cells where you set `ReadOnly` to `TRUE` that provide a visual clue to users that they cannot modify the cell's contents. For information about applying styles, see Section 9.2, "Working with Styles."
`RowLimit`			This group of properties allows you configure the number of rows that the ADF Table component or ADF Read-only Table component download and display. For more information, see Section 7.17, "Limiting the Number of Rows Your Table-Type Component Downloads."
`RowLimit.Enabled`	Boolean	N	Set to `TRUE` to limit the number of rows downloaded to the value specified by `RowLimit.MaxRows`. `TRUE` is the default value. A value for this property is required.

Table A–1 (Cont.) Frequently Used Properties in ADF Desktop Integration

Name	Type	EL	Description
`RowLimit.MaxRows`	Integer	Y	Specify an EL expression that evaluates to the maximum number of rows to download. The component evaluates the EL expression when it invokes its `Download` action. The default value is `500`. If `MaxRows` is not a positive integer, the component attempts to download as many rows as possible. An invalid expression such as "`ABC`" is interpreted as `-1` (negative integer). As a result, the component attempts to download as many rows as possible. Note that setting the value of `MaxRows` to `0` results in a message where the user is asked if they want to download the first `0` rows. To avoid this, set `MaxRows` to a positive integer other than `0`.

Table A–1 (Cont.) Frequently Used Properties in ADF Desktop Integration

Name	Type	EL	Description

`RowLimit.WarningMessage`	String	Y	Write an EL expression to generate a message to display to the end user if the number of rows available to download exceeds the number specified by `RowLimit.MaxRows`. The component evaluates this EL expression each time it invokes its `Download` action. The maximum number of rows that a Excel 2007, or a higher version, worksheet can contain is approximately 1 million.
			
			The default value for `RowLimit.WarningMessage` is:
			
			`#{_ADFDIres['ROWLIMIT_WARNINGS_MESSAGE_1']}`
			
			You can specify a string key from a custom resource bundle to use instead of the default value. Write a value similar to the following for a string key in your resource bundle if you want the warning message to let the end user know how many rows he or she can download:
			
			`Too many rows available. Do you want to download the first {0} rows?`
			
			where `{0}` is a placeholder that references the value of `RowLimit.MaxRows` at runtime.
			
			Write an EL expression similar to the following for `RowLimit.WarningMessage`:
			
			`#{res[‘stringkey’]}`
			
			where `res` refers to the custom resource bundle and `stringkey` refers to the string key that you defined in the custom resource bundle. For more information about resource bundles, see Section 10.2, "Using Resource Bundles in an Integrated Excel Workbook."
			
			If the value for this property is null, the `Download` action downloads the number of rows specified by `RowLimit.MaxRows` without displaying a message to the end user.

`StyleName`	String	Y	Specifies the style in the current Excel workbook to apply when the Oracle ADF component is rendered. For more information, see Section 9.2, "Working with Styles."
`Value`	Varies	Y	This property references an EL expression that is evaluated after the invocation of the ADF Table component’s `RowDownSync` action or a worksheet’s `DownSync` action. The resulting value is typically the primary value seen in the selected component.

A.2 ADF Input Text Component Properties

Table A–2 lists alphabetically the properties of the ADF Input Text component.

Table A–2 ADF Input Text Component Properties

Name	Description
`Annotation`	For information about this property, see Table A–1.
`ComponentID`	For information about this property, see Table A–1.
`InputText.DoubleClickActionSet`	Specifies the action set invoked when a user double-clicks the cell. For information about action sets, see Section A.11, "Action Set Properties."
`InputText.ReadOnly`	For information about this property, see Table A–1.
`InputText.Value`	For information about this property, see Table A–1.
`Position`	For information about this property, see Table A–1.
`StyleName`	For information about this property, see Table A–1.

A.3 ADF Output Text Component Properties

Table A–3 lists alphabetically the properties of the ADF Output Text component.

Table A–3 ADF Output Text Component Properties

Name	Description
`Annotation`	For information about this property, see Table A–1.
`ComponentID`	For information about this property, see Table A–1.
`OutputText.DoubleClickActionSet`	Specifies the action set invoked when a user double-clicks the cell. For information about action sets, see Section A.11, "Action Set Properties."
`OutputText.Value`	For information about this property, see Table A–1.
`Position`	For information about this property, see Table A–1.
`StyleName`	For information about this property, see Table A–1.

A.4 ADF Label Component Properties

The ADF Label component displays a static string value at runtime. ADF Desktop Integration generates the value when the EL expression that the `Label` property references is evaluated. For information about using labels, see Section 9.4, "Using Labels in an Integrated Excel Workbook."

Table A–4 lists alphabetically the properties that the ADF Label component references.

Table A–4 ADF Label Component Properties

Name	Description
`Annotation`	For information about this property, see Table A–1.
`ComponentID`	For information about this property, see Table A–1.
`Label`	For information about this property, see Table A–1.
`Position`	For information about this property, see Table A–1.
`StyleName`	For information about this property, see Table A–1.

A.5 ADF List of Values Component Properties

Table A–5 lists the properties of the ADF List of Values component. For information about creating an ADF List of Values component, see Section 6.6, "Inserting an ADF List of Values Component."

Table A–5 ADF List of Values Component Properties

Name	Type	EL	Description
`Annotation`			For information about this property, see Table A–1.
`ComponentID`			For information about this property, see Table A–1.
`ListOfValues.DependsOnListID`	List binding	N	Select the list binding whose value at runtime determines the choices available in the dependent list of values at runtime. The list binding that you select can be a model-driven list. For more information about dependent list of values, see Section 8.8, "Creating Dependent Lists of Values in an Integrated Excel Workbook."
`ListOfValues.ListID`	List binding	N	Select the list binding that defines the values available in the list of values. The list binding that you select can be a model-driven list.
`ListOfValues.ReadOnly`	Boolean	N	For information about this property, see Table A–1.
`Position`			For information about this property, see Table A–1.
`StyleName`			For information about this property, see Table A–1.

A.6 TreeNodeList Subcomponent Properties

The TreeNodeList is an ADF Table subcomponent that renders dropdown menus in columns of the ADF Table component at runtime. It provides the same functionality to end users as the ADF List of Values component.

The TreeNodeList subcomponent does not appear in the components palette of the ADF Desktop Integration task pane. Instead, you configure properties for this subcomponent when you specify `TreeNodeList` as the subcomponent to invoke for the ADF Table component's `UpdateComponent` or `InsertComponent` table column properties described in Section A.9.2, "ADF Table Component Column Properties."

Table A–6 describes the properties that you configure for the TreeNodeList subcomponent.

Table A–6 TreeNodeList Subcomponent Properties

Name	Type	EL	Description
`DependsOnList`	Tree binding attribute or List binding	Y	Specify the tree binding attribute or list binding that serves as the parent list of values in a dependent list of values. Note that the tree binding attribute you specify must be associated with a model-driven list. For more information about dependent list of values, see Section 8.8, "Creating Dependent Lists of Values in an Integrated Excel Workbook."
`List`	Tree binding attribute	Y	Specify the tree binding attribute associated with a model-driven list that defines the values available in the runtime dropdown menu to appear in the ADF Table component's column.
`ReadOnly`	Boolean	Y	For information about this property, see Table A–1.

A.7 ModelDrivenColumnComponent Subcomponent Properties

The ModelDrivenColumnComponent subcomponent, like the TreeNodeList subcomponent, does not appear in the components palette of the ADF Desktop Integration task pane. Instead, you configure properties for this subcomponent when you specify `ModelDrivenColumnComponent` as the subcomponent to invoke for the ADF Table component's `UpdateComponent` or `InsertComponent` table column properties described in Section A.9.2, "ADF Table Component Column Properties."

Table A–7 describes the properties that you configure for the ModelDrivenColumnComponent subcomponent.

Table A–7 ModelDrivenColumnComponent Subcomponent Properties

Name	Type	EL	Description
`DoubleClickActionSet`			Specifies the action set invoked when a user double-clicks the cell. For information about action sets, see Section A.11, "Action Set Properties."
`ReadOnly`	Boolean	Y	For information about this property, see Table A–1.
`Value`	Varies	Y	For information about this property, see Table A–1.

A.8 ADF Button Component Properties

Table A–8 lists alphabetically the properties of the ADF Button component.

Table A–8 ADF Button Component Properties

Name	Description
`Annotation`	For information about this property, see Table A–1.
`ClickActionSet`	Specify the action set to invoke when a user clicks the button. For information about action sets, see Section A.11, "Action Set Properties."
`ComponentID`	For information about this property, see Table A–1.
`Label`	For information about this property, see Table A–1.
`LowerRightCorner`	This property is an Excel cell reference. Used with `Position`, it specifies the area that the button occupies on the Excel worksheet.
`Position`	For information about this property, see Table A–1.

A.9 ADF Table Component Properties and Actions

The ADF Table component uses the properties and component actions listed here.

A.9.1 ADF Table Component Properties

Table A–9 lists alphabetically the properties the ADF Table component uses.

Table A–9 ADF Table Component Properties

Name	Type	EL	Description
`Annotation`			For information about this property, see Table A–1.
`BatchOptions`			This group of properties enables you to configure batch options for the ADF Table component. For more information about how you use these properties, see Section 7.10, "Batch Processing in an ADF Table Component."
`BatchOptions.BatchSize`	Integer	N	Specifies how many rows to process before an ADF Table component action (`Upload` or `DeleteFlaggedRows`) invokes `CommitBatchActionID`. Any value other than a positive integer results in all rows being processed in a single batch. The default value is 100 rows. A value for this property is required.
`BatchOptions.CommitBatchActionID`	Action binding	N	Specify an action binding to invoke when the number of rows specified by `BatchSize` have been processed. The action binding is expected to be a commit-type action.
`BatchOptions.LimitBatchSize`	Boolean	N	Set this property to `TRUE` to process rows in batches where each batch contains the number of rows specified by `BatchSize`. If set to `FALSE`, all rows are processed in a single batch.
`BatchOptions.StartBatchActionID`	Action binding	N	Specify an action binding to invoke at the beginning of each batch. For example, this property might be used for an operation like "start transaction", if required by a particular database. A value for this property is optional.
`Columns`			An array of columns. For information about the properties that each column in the array supports, see Section A.9.2, "ADF Table Component Column Properties."
`ComponentID`			For information about this property, see Table A–1.
`Position`			For information about this property, see Table A–1.
`RowActions`			This group of properties allows you specify which actions are enabled and can be invoked.
`RowActions.DeleteRowActionID`	Action binding	N	Specify an action binding to invoke for each row flagged for deletion. A value for this property is optional.
`RowActions.DeleteRowEnabled`	Boolean	N	Set to `TRUE` to allow a user to delete existing rows. `FALSE` is the default value. A value for this property is required.
`RowActions.FailureActionID`	Action binding	N	Specify an action binding to invoke if failures occur during the processing of rows. A value for this property is optional.
`RowActions.InsertAfterRowActionID`	Action binding	N	Specify an action binding to invoke for each row inserted using the ADF Table component `Upload` action. The action binding is invoked after the attributes are set. Use of this property is suitable with a custom action where a variable iterator is employed along with the main iterator. A value for this property is optional.

Table A–9 (Cont.) ADF Table Component Properties

Name	Type	EL	Description
`RowActions.InsertBeforeRowActionID`	Action binding	N	Specify an action binding to invoke for each row inserted using the `Upload` component action. The action binding specified is invoked before the attributes are set. A value for this property is optional.
`RowActions.InsertRowEnabled`	Boolean	N	Set to `TRUE` to allow the end user insert new rows in the ADF Table component. `FALSE` is the default value. If you set this property to `TRUE`, you must specify values for one or both of the following properties: ■ `RowActions.InsertAfterRowActionID` ■ `RowActions.InsertBeforeRowActionID` Which property (`InsertAfterRowActionID` or `InsertBeforeRowActionID`) you specify a value for depends on how your Fusion web application creates new rows. Typically, a Fusion web application uses the `CreateInsert` action binding to create and insert a new row. In this scenario, you specify the `CreateInsert` action binding as the value for `InsertBeforeRowActionID`. For more information about inserting rows in an ADF Table component, see Section 7.5, "Configuring an ADF Table Component to Insert Data."
`RowActions.InsertRowsAfterUploadEnabled`	Boolean	N	Set to `TRUE` to allow the end user to reinsert changed rows regardless of whether they have been previously uploaded. `FALSE` is the default value. The property is ignored if `InsertRowEnabled` is set to `FALSE`.
`RowActions.UpdateRowActionID`	Action binding	N	Specify an action binding to invoke for each row updated. A value for this property is optional.
`RowActions.UpdateRowEnabled`	Boolean	N	Set to `TRUE` to allow a user update an existing row. `TRUE` is the default value. A value for this property is required.
`RowData`			Set values for the `CachedAttributes` property when you want to cache data in an integrated Excel workbook across multiple sessions with the Fusion web application. Set a value for the `ChangeIndicatorAttributeID` property to determine whether a row has been modified by another user since you downloaded it from the Fusion web application.

Table A–9 (Cont.) ADF Table Component Properties

Name	Type	EL	Description
`RowData.CachedAttributes`	Array	N	Specify values for the properties in this array to determine the attributes for which data is cached. Each `CachedTreeAttribute` element in this array supports the following properties: ■ `Value` Select the tree binding attribute for which data is to be cached. ■ `Annotation` For more information about this property, see Table A–1. Do not configure a component (for example, an ADF Table component's column or an ADF Input Text component) so the end user can view or edit an attribute binding that you have also specified for an element in the `RowData.CachedAttributes` array. The `RowData.CachedAttributes` array caches the values retrieved by the worksheet `DownSync` action. The worksheet `UpSync` action sends the values cached by the `RowData.CachedAttributes` array to the Fusion web application. This may override edits the end user makes to an attribute binding exposed through a component in the worksheet. For information about using the `RowData.CachedAttributes` array to cache data in an ADF Table component, see Section 8.5, "Inserting Values in ADF Table Columns from a Web Page Pick Dialog."
`RowData.ChangeIndicatorAttributeID`	Binding	N	Specify the row-specific attribute of the tree binding used to determine if a row has been modified by another user since the row was last downloaded by to your integrated Excel workbook. For more information, see Section 12.7, "Handling Data Conflicts When Uploading Data from a Workbook."

Table A–9 (Cont.) ADF Table Component Properties

Name	Type	EL	Description
`RowLimit`			For information about this group of properties, see Table A–1.
`TreeID`	Binding	N	Specify a tree binding from the current worksheet's page definition file. You must specify a value for this property so that row downloads and uploads function properly. For more information about the page definition requirements for an integrated Excel workbook, see Table 4–1.
`UniqueAttribute`	Attribute binding	Y	Write an EL expression to specify an attribute of the tree binding that you specified as the value for `TreeID`. The value of this attribute is cached in the integrated Excel workbook during invocation of the ADF Table component's `Download` action. ADF Desktop Integration uses this value to ensure that the tree binding's iterator is positioned correctly before setting or getting data from the current row. A value for this property is optional if the: ▪ ADF Table component is configured to be insert-only (`RowActions.InsertRowEnabled` is set to `True` and `RowActions.UpdateRowEnabled` is set `False`) ▪ Underlying tree binding exposes a `rowKey` (in which case the `rowKey` is used for positioning) A value is required if the tree binding's iterator does not expose a `rowKey`.

A.9.2 ADF Table Component Column Properties

Table A–10 describes the properties that a column in the TableColumn array can use.

Table A–10 ADF Table Component Column Properties

Name	Type	EL	Description
`Annotation`			For information about this property, see Table A–1.
`CellStyleName`	String	Y	Write an EL expression that resolves to an Excel style name that is applied to each cell in the column.
`DynamicColumn`	Boolean	N	Set to `True` to make a column dynamic. `False` is the default value. For more information about dynamic columns, see Section 7.15, "Adding a Dynamic Column to Your ADF Table Component."
`HeaderLabel`	String	Y	Write an EL expression that, when evaluated at runtime, displays a label in the column header.
`HeaderStyleName`	String	Y	Write an EL expression that resolves to an Excel style name that is applied to each cell in the column header.

Table A–10 (Cont.) ADF Table Component Column Properties

Name	Type	EL	Description
`ID`	String	N	Assign a name to the column to identify it and its purpose. The value that you assign for this property has no functional impact. However, you must specify a value and the value that you specify must be unique within the list of columns. It serves to help you keep track of columns in the ADF Table component. The following IDs are reserved to the three default columns in the ADF Table component: ■ `_ADF_ChangedColumn` ■ `_ADF_FlagColumn` ■ `_ADF_StatusColumn` For more information about these columns, see Section 7.11, "Special Columns in the ADF Table Component."
`InsertComponent`	ADF component	N	Specifies the properties of the component that represents the binding for insert operations. This component can be one of the following: ■ InputText component For information about the properties that this component supports, see Section A.2, "ADF Input Text Component Properties." ■ OutputText component For information about the properties that this component supports, see Section A.3, "ADF Output Text Component Properties." ■ TreeNodeList component For information about the properties that this component supports, see Section A.6, "TreeNodeList Subcomponent Properties." ■ ModelDrivenColumnComponent For information about the properties that this component supports, see Section A.7, "ModelDrivenColumnComponent Subcomponent Properties." When `InsertUsesUpdate` is set to `True`, the ADF Table component ignores the value of the `InsertComponent` property.

Table A–10 (Cont.) ADF Table Component Column Properties

Name	Type	EL	Description
`InsertUsesUpdate`	Boolean	N	Set to `True` if insert and update operations use the same component type. When `True`, the ADF Table component ignores the values of the `InsertComponent` property and reads the value of the `UpdateComponent` property. The default value is `True`.
`UpdateComponent`	ADF component	N	Specifies the properties of the component that represents the binding for update and download operations. This component can be one of the following: ■ InputText component For information about the properties that this component supports, see Section A.2, "ADF Input Text Component Properties." ■ OutputText component For information about the properties that this component supports, see Section A.3, "ADF Output Text Component Properties." ■ TreeNodeList component For information about the properties that this component supports, see Section A.6, "TreeNodeList Subcomponent Properties." ■ ModelDrivenColumnComponent For information about the properties that this component supports, see Section A.7, "ModelDrivenColumnComponent Subcomponent Properties."
`Visible`	Boolean	Y	Write an EL expression that resolves to `True` or `False`. If `True`, the column appears in the ADF Table component. If `False`, the column does not appear. `True` is the default value. If you make a column dynamic, the ADF Table component ignores the value of the `Visible` property. For more information about dynamic columns, see Section 7.15, "Adding a Dynamic Column to Your ADF Table Component."

A.9.3 ADF Table Component Actions

Table A–11 describes the component actions available for use with the ADF Table component.

Table A–11 ADF Table Component Actions

Component Action	Description
`ClearCachedRowAttributes`	Clears the values of cached attributes for the current row of the ADF Table component. Only a `DoubleClickActionSet` in an ADF Table component's column should invoke this action.
`DeleteFlaggedRows`	Invokes a specified action on each of a set of flagged rows in the ADF Table component and then removes these rows from the ADF Table component. For more information, see Section 7.9, "Configuring an ADF Table Component to Delete Rows in the Fusion Web Application."
`DisplayRowErrors`	Displays error details for the current row in the ADF Table component if error details are available. This action should only be invoked from a column's action set in an ADF Table component. By default, the `_ADF_StatusColumn` described in Table 7.11 is configured with an action set that invokes this action.

Table A–11 (Cont.) ADF Table Component Actions

Component Action	Description
`DisplayTableErrors`	Displays a detailed list of errors in a message dialog for the ADF Table component if any errors are available. Do not invoke this action from a column's action set in an ADF Table component. Instead configure an action set for an ADF Button, ADF Output Text component, or worksheet ribbon button to invoke this action.
`Download`	Download the rows corresponding to the current state of `TreeID`. For information about `TreeID`, see Section A.9.1, "ADF Table Component Properties."
`DownloadFlaggedRows`	Downloads the flagged rows corresponding to the current set of items available within `TreeID`. For information about `TreeID`, see Table A–9.
`DownloadForInsert`	Invoke this action to download rows to the ADF Table component from the Fusion web application and treat each row as a pending insert. Do not specify `Download` and `DownloadForInsert` as actions within the same action set. The last of these actions that the action set invokes determines what data appears in the ADF Table component. Specify the `MarkAllRowsChanged` component action as the next component action to invoke in an action set where you want all rows that the `DownloadForInsert` action downloads to be marked as changed. The `DownloadForInsert` action is ignored if it is invoked from an action set while the ADF Table component's `RowActions.InsertRowEnabled` property is set to `False`. Set `RowActions.InsertRowEnabled` to `True` to correctly invoke the `DownloadForInsert` action. For more information, see Section 7.7, "Configuring a Worksheet to Download Pre-Insert Data to an ADF Table Component" and Section 15.2, "Restore Server Data Context Between Sessions."
`FlagAllRows`	Sets the flag for all rows. Invoke this action to set a flag character in all rows of the `_ADF_FlagColumn` column. The flag character has the following properties: `Character Code 25CF, Unicode(hex)` It appears as a solid circle. For more information about the `_ADF_FlagColumn` column, see Section 7.10.2, "Row Flagging in an ADF Table Component" and Section 7.11, "Special Columns in the ADF Table Component."
`Initialize`	This action performs the following actions: ■ Removes all rows of data from the ADF Table component ■ Clears the values of cached attributes from rows in the ADF Table component ■ Creates the placeholder row ■ Recalculates how many dynamic columns to render in the ADF Table component ■ Redraws column headers If the ADF Table component contains pending changes that have not been saved in the integrated Excel workbook, a dialog appears to the end user that allows cancellation of invocation of this action.
`MarkAllRowsChanged`	After an action set invokes a `DownloadForInsert` component action, specify the `MarkAllRowsChanged` component action as the component action to invoke if you want all rows downloaded by the `DownloadForInsert` component action marked as changed in the `_ADF_ChangedColumn` column.
`MarkAllRowsUnchanged`	Specify this component action to clear all flags from the `_ADF_ChangedColumn` column.

Table A–11 (Cont.) ADF Table Component Actions

Component Action	Description
`RowDownSync`	Synchronizes data from the row in the ADF Table component's iterator in the Fusion web application that corresponds to the current worksheet row to the worksheet. As this action acts upon the current worksheet row, only a `DoubleClickActionSet` associated with a column in the ADF Table component should invoke this action. The ADF Table component does not evaluate or apply the value of a column's `Visible` property when invoking `RowDownSync`. The ADF Table component evaluates and applies the value of a column's `CellStyleName` property when invoking `RowDownSync`. For more information about column properties, see Section A.9.2, "ADF Table Component Column Properties."
`RowUpSync`	Synchronizes any pending changes in the current worksheet row that the ADF Table component references to the Fusion web application. `RowUpSync` acts upon the current worksheet row so only a `DoubleClickActionSet` associated with a column in the ADF Table component should invoke this action. The `DoubleClickActionSet` that invokes `RowUpSync` also changes the position of the ADF Table component's iterator on the Fusion web application to the current worksheet row (assuming it exists in the Fusion web application).
`UnflagAllRows`	Removes flags from cells in the `_ADF_FlagColumn` column. For more information about the `_ADF_FlagColumn`, see Section 7.10.2, "Row Flagging in an ADF Table Component" and Section 7.11, "Special Columns in the ADF Table Component."
`Upload`	Uploads pending changes to the Fusion web application. For more information, see Section 7.8, "Configuring an Oracle ADF Component to Upload Changes from an ADF Table Component." For more information about resolving data conflict between the Excel workbook and the Fusion web application, see Section 12.7, "Handling Data Conflicts When Uploading Data from a Workbook".

A.10 ADF Read-only Table Component Properties and Actions

The ADF Read-only Table component exposes one action, `Download`. This action downloads the current rows in the table identified by the ADF Read-only Table property, `TreeID`. Table A–12 describes `TreeID` and the other properties that the ADF Read-only Table component supports.

Table A–12 ADF Read-only Table Component Properties

Name	Type	EL	Description
`Annotation`			For information about this property, see Table A–1.
`Columns`	Array	N	References an array of read-only columns. For information about the properties that a column in this array can support, see Table A–13.
`ComponentID`			For information about this property, see Table A–1.
`Position`			For information about this property, see Table A–1.
`RowLimit`			For information about this group of properties, see Table A–1.
`TreeID`	Tree binding	N	References a tree binding ID from the page definition file associated with the current worksheet if the ADF Read-only Table component was created by inserting a tree binding into the worksheet.

Table A–13 lists alphabetically the properties that a column in the `ReadOnlyColumn` array can use.

Table A–13 ADF Read-only Table Component Column Properties

Name	Type	EL	Description
`Annotation`			For information about this property, see Table A–1.
`CellStyleName`	String	Y	Write an EL expression that resolves to an Excel style name that is applied to each cell in the column.
`HeaderLabel`	String	Y	Write an EL expression that resolves to a label for the column header.
`HeaderStyleName`	String	Y	Write an EL expression that resolves to an Excel style name that is applied to each cell in the column header.
`ID`	String	N	Assign a name to the column to identify it and its purpose. The value that you assign for this property has no functional impact. However, you must specify a value and the value that you specify must be unique within the list of columns. It serves to help you keep track of columns in the ADF Read-only Table component.
`OutputText`	ADF Component		For information about the properties that this component supports, see Section A.3, "ADF Output Text Component Properties."

A.11 Action Set Properties

Table A–14 lists alphabetically the properties that you can configure for an action set.

Table A–14 Action Set Properties

Name	Type	EL	Description
`ActionOptions`			This group of properties specifies options for invoking local and remote actions.
`ActionOptions.AbortOnFailure`	Boolean	N	When set to `TRUE`, the remaining actions in the array are not invoked if an action fails. If `FALSE`, all actions are invoked regardless of the success or failure of previous actions. The default value is `TRUE`.
`ActionOptions.FailureActionID`	Action binding	N	Specify the action binding to invoke if an action set does not complete successfully. For example, you could specify an action binding that rolls back changes made during the unsuccessful invocation of the action set.
`ActionOptions.SuccessActionID`	Action binding	N	Specify an action binding to invoke if an action set completes successfully. For example, you could specify an action binding that executes a `commit` action. A value for this property is optional.

Table A–14 (Cont.) Action Set Properties

Name	Type	EL	Description
`Actions`	Array	N	Specifies an ordered array of actions. An action can be one of the following: ■ `ADFmAction` Invokes an action binding or method action binding in the underlying page definition file. The `ADFmAction.ActionID` property identifies the action binding or method action binding to invoke. For information about page definition files, see Section 4.3, "Working with Page Definition Files for an Integrated Excel Workbook." ■ `ComponentAction` Invokes an action that a component on the worksheet exposes. `ComponentAction.ComponentID` identifies the component that exposes the action while `ComponentAction.Method` identifies the action to invoke. The ADF Table and ADF Read-only Table components are the only components in ADF Desktop Integration that expose actions. For information about these actions, see Section A.9, "ADF Table Component Properties and Actions" and Section A.10, "ADF Read-only Table Component Properties and Actions." For information about invoking component actions, see Section 8.2.2, "How to Invoke Component Actions in an Action Set." ■ `WorksheetMethod` Invokes a worksheet action. For information about worksheet actions, see Section A.13, "Worksheet Actions and Properties." ■ `Confirmation` Invokes a confirmation dialog. For more information about the properties that this action uses, see Section A.11.1, "Confirmation Action Properties." ■ `Dialog` Invokes a web page in a popup dialog or Excel's task pane. For more information, see Section 8.4, "Displaying Web Pages from a Fusion Web Application."
`Alert`			This group of properties determines if and how an alert-style dialog appears to the user to indicate that the requested action is complete. The dialog that appears contains one button that allows the user to acknowledge the message and dismiss the dialog. For information about how to display an alert message, see Section 8.2.7, "How to Provide an Alert After the Invocation of an Action Set." Many properties in this group make use of EL expressions to retrieve string values from resource bundles. For more information about using EL expressions, see Section 10.2, "Using Resource Bundles in an Integrated Excel Workbook."
`Alert.Enabled`	Boolean	N	Set to `TRUE` to display an alert message to end users that notifies them when an action set operation completes successfully or includes one or more failures. For more information, see Section 8.2.7, "How to Provide an Alert After the Invocation of an Action Set."

Table A–14 (Cont.) Action Set Properties

Name	Type	EL	Description
`Alert.FailureMessage`	String	Y	Specify an EL expression that evaluates to a message to appear in the dialog if errors occur during execution of the action set. The default EL expression is: `#{_ADFDIres['DIALOGS_ACTION_ALERT_FAILURE_PROMPT']}`
`Alert.OKButtonLabel`	String	Y	Specify an EL expression that evaluates to a message to appear in the OK button of the dialog. The default EL expression is: `#{_ADFDIres['DIALOGS_OK_BUTTON_LABEL']}`
`Alert.SuccessMessage`	String	Y	Specify an EL expression that evaluates to a message to appear in the dialog if no errors occur during the execution of the action set. The default EL expression is: `#{_ADFDIres['DIALOGS_ACTION_ALERT_SUCCESS_PROMPT']}`
`Alert.Title`	String	Y	Specify an EL expression that evaluates to a message to appear in the title area of the dialog. The default EL expression is: `#{_ADFDIres['DIALOGS_ACTION_TITLE']}`
`Annotation`			For information about `Annotation`, see Table A–1.
`Status`			This group of properties determines if and how a status message appears during the execution of an action set. For information about how to display a status message, see Section 8.2.5, "How to Display a Status Message While an Action Set Executes." Many properties in this group make use of EL expressions that reference string keys defined in resource bundles. For more information, see Section 10.2, "Using Resource Bundles in an Integrated Excel Workbook."
`Status.Enabled`	Boolean	N	If `TRUE` (default), a status window appears during the execution of the action set. If `FALSE`, no status window appears.
`Status.Message`	String	Y	Specify an EL expression to evaluate and display in the status window while the action set executes. The default value is: `#{_ADFDIres['STATUS_MESSAGE_PROMPT']}`
`Status.Title`	String	Y	Specify an EL expression to evaluate and display in the title area of the status window while the action set executes. The default value is: `#{_ADFDIres['DIALOGS_ACTION_TITLE']}`

A.11.1 Confirmation Action Properties

Table A–15 lists alphabetically the properties that the `Confirmation` action in the array of `Actions` of an action set supports. For information about the other properties the array of `Actions` and action sets use, see Table A–14.

Table A–15 Confirmation Action Properties

Name	Type	EL	Description
`Annotation`			For information about `Annotation`, see Table A–1.
`CancelButtonLabel`	String	Y	An EL expression that is evaluated and displayed in the Cancel button at runtime. The default value is: `#{_ADFDIres['DIALOGS_CANCEL_BUTTON_LABEL']}`
`OKButtonLabel`	String	Y	An EL expression that is evaluated and displayed in the OK button at runtime. The default value is: `#{_ADFDIres['DIALOGS_OK_BUTTON_LABEL']}`
`Prompt`	String	Y	An EL expression that is evaluated and displayed in the main area of the confirmation dialog at runtime. The default value is: `#{_ADFDIres['DIALOGS_ACTION_CONFIRM_PROMPT']}`
`Title`	String	Y	An EL expression that is evaluated and displayed in the title area of the confirmation dialog at runtime. The default value is: `#{_ADFDIres['DIALOGS_ACTION_TITLE']}`

A.11.2 Dialog Action Properties

Table A–16 describes the properties that the `Dialog` action in the array of `Actions` of an action set supports. For information about the other properties the array of `Actions` and action sets use, see Table A–14.

For information about how to use the properties in Table A–16 to invoke a web page from a Fusion web application, see Section 8.4, "Displaying Web Pages from a Fusion Web Application."

Table A–16 Dialog Action Properties

Name	Type	EL	Description
`Annotation`	String	N	For information about this property, see Table A–1.
`Page`	String	N	Specify the web page that the action invokes. Relative and absolute URLs are valid values.
`ShareFrame`	Boolean	N	Set to `TRUE` (default) to execute the web page specified by the `Dialog.Page` property in the same data control frame as the Excel worksheet. If you specify an absolute URL, ADF Desktop Integration ignores the value of the `Dialog.ShareFrame` property.
`Target`	List	N	Specifies how the web page the action invokes is rendered. Select: ■ `Popup` to render the web page in a modal dialog within an embedded web browser. ■ `TaskPane` to render the web page in runtime task pane.
`Title`	String	Y	Write an EL expression that resolves to the title of the `Dialog` at runtime or write a literal string.
`WindowSize`	Integer	N	Specify the initial size in pixels of the dialog that appears to the user. Valid values range from `0` to `2147483647`. Values will be revised upwards or downwards as appropriate at runtime if the specified values are too large or too small. The default value for `Height` is `625` and `600` for `Width`.

A.12 Workbook Actions and Properties

Table A–17 describes the actions that a workbook can invoke. For information about configuring ribbon buttons to invoke these actions, see Section 8.3.1, "How to Define a Workbook Command Button for the Runtime Ribbon Tab."

Table A–17 Workbook Actions

Action	Description
`Login`	When invoked, this action creates a new session between the integrated Excel workbook and the Fusion web application. If invoked when a session has already been established, it first invokes the `Logout` action internally to free that session. For a workbook running against a web application that is enforcing authentication, the `Login` action prompts the end user to provide valid user credentials.
`Logout`	When invoked, ADF Desktop Integration sends a request to the Fusion web application to invalidate the session between the integrated Excel workbook and the Fusion web application. After invoking this action, the end user must be authenticated the next time the Excel workbook accesses the Fusion web application.

Table A–17 (Cont.) Workbook Actions

Action	Description
`ClearAllData`	When invoked, this action clears all data entered by the user from cells that reference Oracle ADF bindings. Tables, such as those created by the ADF Table and ADF Read-only Table components, will be truncated so that they only display header rows with labels cleared. Values in cells that reference the Input Text or Output Text components are cleared. Column headers and labels are cleared as well. References to all resource bundles that the integrated Excel workbook uses are cleared. Worksheets that do not contain bindings or reference a page definition file remain unchanged. A dialog prompts the end user to confirm invocation of this action. Once the end user confirms invocation, ADF Desktop Integration executes the following events after invocation of the action: ■ Invokes the integrated Excel workbook's `Logout` action ■ Terminates the runtime session and clears all data from the integrated Excel workbook and all caches ■ Reinitializes the integrated Excel workbook and invokes the workbook's `Login` action Invocation of the `ClearAllData` action does not change data hosted by the Fusion web application. One or more of the following actions must be invoked to change data hosted by the Fusion web application: ■ A worksheet's `UpSync` action This action synchronizes all data referenced by non-table type components. For more information, see Section A.13, "Worksheet Actions and Properties." ■ An ADF Table component's `RowUpSync` action can be used to synchronize any pending changes in a row to the Fusion web application. The ADF Table component's `DeleteFlaggedRows` action can be invoked to delete flagged rows. For more information about ADF Table component actions, see Section A.9.3, "ADF Table Component Actions."
`EditOptions`	When invoked, this action launches a dialog that shows the current value of the `WebAppRoot` property and allows the end user to enter a new value. If the end user chooses to change the value of `WebAppRoot`, a confirmation dialog appears after the end user clicks **OK**. Once the change is confirmed, the following events occur: ■ Workbook `ClearAllData` action is invoked ■ Workbook `Logout` action is invoked ■ All data referenced by bindings in the workbook is removed ■ References to `WebAppRoot` are updated in the Excel workbook's metadata ■ Workbook `Login` action is invoked to authenticate the user with the Fusion web application that is specified as the value for `WebAppRoot` The `ClearAllData` workbook action clears all resource bundles referenced by the integrated Excel workbook. After `WebAppRoot` is changed, the integrated Excel workbook attempts to retrieve resource bundles from the Fusion web application as part of the reinitialization process. This request to the Fusion web application triggers the authentication process.
`ViewAboutDialog`	When invoked, this action launches a dialog called **About** that displays information defined in the `BrandingItems` workbook property and other information such as the versions of supporting software.

Table A–18 lists alphabetically the ADF Desktop Integration properties that an Excel workbook can use.

Table A–18 Workbook Properties

Name	Type	EL	Description
`ApplicationHomeFolder`	String	N	Specify the absolute path to the directory that is the root for the JDeveloper application workspace (`.jws`) where you developed the desktop integration project. For example, the value of this property in a workbook integrated with the Master Price List module could be something similar to the following: `C:\FusionOrderDemo\MasterPriceList` ADF Desktop Integration prompts you to specify a value for this property the first time that you open an integrated Excel workbook. If you click **Cancel** in the dialog that prompts you for a value, ADF Desktop Integration sets the value of `ApplicationHomeFolder` to the directory that contains the Excel workbook. For more information, see Section 4.4.3, "How to Configure a New Integrated Excel Workbook."
`BrandingItems`	Array	N	An array of name-value pairs that resolve to resource bundle references (for example, `#{res['myAppName']}`) or a literal string. Each pair in the array consists of a name and a value. Each name and value can reference a literal string or an EL expression. For information about branding your integrated Excel workbook, see Section 9.6, "Branding Your Integrated Excel Workbook."
`Login.WindowSize`	Integer	N	Specify the initial size in pixels of the login dialog that appears to the user. Valid values range from `0` to screen width or height. Values will be revised upwards or downwards as appropriate at runtime if the specified values are too large or too small. The default value for `Height` is `625` and `Width` is `600`.
`Parameters`	Array	N	An array of workbook initialization parameters that you configure to pass the parameters from a page in a Fusion web application to an integrated Excel workbook. You can define multiple workbook initialization parameters in the Fusion web application's page. Each workbook initialization parameter (parameter that references a URL argument) that you define in a page must be specified in a `Parameter` property of this array, otherwise it is ignored. Each element in the array supports the following properties: ■ `Annotation` For more information about this property, see Table A–1. ■ `Parameter` You specify the name of the workbook initialization parameter you defined in the page of the Fusion web application from which the end user downloads the integrated Excel workbook. For information about using this property, see Section 14.5, "Passing Parameter Values from a Fusion Web Application Page to a Workbook."

Table A–18 (Cont.) Workbook Properties

Name	Type	EL	Description
`Project`	String	N	Specify the name of a JDeveloper project in the current JDeveloper workspace. ADF Desktop Integration attempts to load the `.jpr` file that corresponds to the project that you specify. An error appears if the `.jpr` file is not available or is not in the expected format. When you open an integrated Excel workbook for the first time in design mode, ADF Desktop Integration searches for a `.jpr` file in the parent folder hierarchy. If it finds a `.jpr` file, it sets the value of `Project` to the name of the project that corresponds to the `.jpr` file. ADF Desktop Integration loads the names of the available projects from the *application_name.jws* file specified by `ApplicationHomeFolder`.
`RemoteServletPath`	String	N	Specify the path to the ADF Desktop Integration remote servlet. This path must be relative to the value specified for `WebAppRoot`. Note that the value you specify for `RemoteServletPath` must match the value that is specified in the web application's deployment descriptor file (`web.xml`). The default value for this property is: `/adfdiRemoteServlet`
`Resources`	Array	N	Specifies an array of resource bundles to register with the workbook. Each element in the array supports the following properties: ■ `Alias` Specify a string value that is unique within `Workbook.Resources`. EL expressions use this string to reference the resource bundle. ■ `Annotation` For more information about this property, see Table A–1. ■ `Class` Specify a fully qualified class name. The class name that you specify is expected to be a Java resource bundle class that the Fusion web application you integrate your workbook with uses. For example, the `EditPriceList-DT.xlsx` workbook in the Master Price List module references the following resource bundle: `oracle.fodemo.masterpricelist.resources.UIStrings` For more information, see Section 10.2, "Using Resource Bundles in an Integrated Excel Workbook."
`Runtime Ribbon Tab`	-	-	This group of properties defines whether and how a Ribbon tab appears in Excel at runtime. The following entries in this table describe the properties in the `Runtime Ribbon Tab` group. For more information about Ribbon tab and its commands, see Section 8.3, "Configuring the Runtime Ribbon Tab."
`Runtime Ribbon Tab.Annotation`	String	N	For information about this property, see Section A.1, "Frequently Used Properties in the ADF Desktop Integration."
`Runtime Ribbon Tab.Visible`	Boolean	N	If `TRUE`, the Ribbon tab appears at runtime. The Ribbon tab does not appear if you set `Enabled` to `FALSE`. `TRUE` is the default value.

Table A–18 (Cont.) Workbook Properties

Name	Type	EL	Description
`Runtime Ribbon Tab.Title`	String	Y	Specify an EL expression that evaluates to the title that appears for the Ribbon tab in the title area. Excel imposes a maximum limit of 1024 characters for Ribbon tab titles. Ensure that the runtime value of the EL expression you specify does not exceed 1024 characters as ADF Desktop Integration truncates the value so that Excel does not generate an error message.
`Runtime Ribbon Tab.Workbook Commands`	Array	N	Each element in this array corresponds to a workbook command at runtime. Each element in the array uses the following properties: ■ `Annotation` For more information about this property, see Table A–1. ■ `Label` For more information about this property, see Table A–1. If you want the `&` character to appear in the command label, you must specify `&&`. Excel interprets a single `&` character as a special character, and assigns the next character after `&` as the keyboard accelerator for the workbook command at runtime. ■ `Method` Specify the workbook action that the workbook ribbon button invokes. For more information about workbook actions, see Table A–17.
`WebAppRoot`	String	N	A fully qualified URL to the Fusion web application's root.
`WebPagesFolder`	String	N	Specify the path to the directory that contains the web pages that you intend to use with your integrated Excel workbooks. The value that you specify for the path most be relative to the value of `ApplicationHomeFolder`.
`WorbookID`	String	N	A unique identifier for the integrated Excel workbook. ADF Desktop Integration generates the unique identifier when you open the workbook for the first time in design mode. The value cannot be modified. However, ADF Desktop Integration can generate a new value if you use the **Reset WorkbookID** link in the Edit Workbook Properties dialog. The value of this property is used during tamper check, as described in Section 11.3, "Checking the Integrity of an Integrated Excel Workbook's Metadata."

A.13 Worksheet Actions and Properties

An Excel worksheet with ADF Desktop Integration can invoke the following actions:

- `UpSync`

  Synchronizes any pending changes from the ADF Input Text and ADF List of Values components in the worksheet to the Fusion web application.

- `DownSync`

  Downloads any changes from the Fusion web application to the ADF Input Text, ADF Output Text, and ADF List of Values components in the worksheet.

- `DisplayWorksheetErrors`

Displays a detailed list of errors in a message dialog for the integrated Excel worksheet if any errors are available. Invoke this action in an action set that is invoked by an ADF component (other than the ADF Table-type components) or a worksheet ribbon button.

When you configure an ADF Button component to invoke an action binding or method action binding, the action set to invoke when a user clicks the ADF Button component at runtime is populated as follows by default:

1. `UpSync`
2. Action or method action binding that you specify for the ADF Button component
3. `DownSync`

If the first action that you invoke on a worksheet with an empty form is the `UpSync` worksheet action, you may encounter errors. For this reason, ensure that the first action invoked is the `DownSync` worksheet action. You can configure the ADF Button component's action set or one of the worksheet events (`Startup` or `Activate`) described in Table A–19 to invoke the `DownSync` worksheet action first.

Table A–19 describes the ADF Desktop Integration properties that an Excel worksheet can use.

Table A–19 Worksheet Properties

Name	Type	EL	Description
`Annotation`	String	N	For information about this property, see Table A–1.
`Events`	Array	N	Each element in this array specifies an action set to invoke if the associated worksheet event occurs. For information about action sets, see Section A.11, "Action Set Properties." For information about worksheet events, see the entry in this table for `Events.n.Event`. The following entries in this table prefaced by *Events.n* describe the properties that an element in this array supports where *n* refers to a specific element in the array.
`Events.n.ActionSet`	ActionSe t	N	For more information about the properties of action sets, see Section A.11, "Action Set Properties."
`Events.n.InvokeOnceOnly`	Boolean	N	The default value of this property is `FALSE`. When set to `TRUE`, the workbook stores information about whether the worksheet invoked the action set for this event and, if so, prevents the worksheet from invoking the action set a second time. Note that if the workbook is not saved, this information is lost. This means that the worksheet can invoke the event again the next time that the workbook opens.
`Events.n.Annotation`	String	N	For information about the annotation property, see Table A–1.

Table A–19 (Cont.) Worksheet Properties

Name	Type	EL	Description
`Events.n.Event`	List	N	The worksheet supports the following events that you can configure to invoke an action set: ■ `Startup` Excel starts. ■ `Shutdown` Excel workbook closes or Excel application exits. ■ `Activate` User navigates to the current worksheet. ■ `Deactivate` User navigates away from the current worksheet or Shutdown event triggered. Note that the worksheet events complete execution even if the action sets that it invokes fails. For more information about worksheet events and action sets, see Section 8.2.4, "How to Invoke an Action Set from a Worksheet Event."
`Protection.Mode`	List	N	The worksheet provides two options: ■ `Off` Worksheet protection is not used at runtime. ■ `Automatic` Worksheet protection is enabled automatically at runtime. The default value for this property is `Off`.
`Protection.Password`	String	N	Specify a password to prevent end-users from turning off sheet protection at runtime. The maximum password length allowed by Excel is 255 characters.
`Ribbon Commands`	Array	N	Specify one or more workbook actions that appear as commands at runtime. Each command is an element in the `WorksheetMenuItem` array. Entries in this array support the following properties: ■ `Annotation` ■ `Label` ■ `SelectActionSet` For more information about the `Annotation` and `Label` properties, see Table A–1. For more information about the `SelectActionSet` property, see Section A.11, "Action Set Properties." If you want the `&` character to appear in the command label, you must specify `&&`. Excel interprets a single `&` character as a special character, and assigns the next character after `&` as the keyboard accelerator for the worksheet command at runtime.
`Page Definition`	String	N	Specify the page definition file to associate with the worksheet. For information about page definition files, see Section 4.3, "Working with Page Definition Files for an Integrated Excel Workbook."

Table A–19 (Cont.) Worksheet Properties

Name	Type	EL	Description
`Parameters`	Array	N	An array of worksheet parameters that you configure to pass the parameters from a workbook `Parameters` property to a worksheet in an integrated Excel workbook. Each element in the array supports the following properties: ■ `Annotation` For more information about this property, see Table A–1. ■ `Parameter` Specify the ID of a `parameter` element that you added to the page definition file associated with the worksheet. ■ `Value` Write an EL expression that references the value of the `Parameter` property you specified for the workbook initialization parameter (workbook `Parameters.Parameter` property). The workbook `Parameters.Parameter` property supplies this value the first time that the page definition file associated with this worksheet is initialized. For information about using this property, see Section 14.5, "Passing Parameter Values from a Fusion Web Application Page to a Workbook."
`RowData`			Set values for the `CachedAttributes` property when you want to cache data in an integrated Excel workbook across a multiple sessions with the Fusion web application. Set a value for the `ChangeIndicatorAttributeID` property to determine if a row has been modified by another user since you downloaded it from the Fusion web application.
`RowData.CachedAttributes`	Array	N	Specify values for the properties in this array to determine the attributes for which data is cached. Each `CachedAttribute` element in this array supports the following properties: ■ `AttributeID` This property references the attribute binding for which data is to be cached. Do not specify an attribute binding for `AttributeID` and as an editable field in a form (for example, in an ADF Input Text component) in the same worksheet. ■ `Annotation` For more information about this property, see Table A–1. For more information about clearing the values of cached attributes, see Section 7.18, "Clearing the Values of Cached Attributes in an ADF Table Component."

Table A–19 (Cont.) Worksheet Properties

Name	Type	EL	Description
`RowData.ChangeIndicatorAttributeID`	Binding	N	Specify the row-specific attribute of the tree binding used to determine if a row has been modified by another user since the row was last downloaded by to your integrated Excel workbook. For more information, see Section 12.7, "Handling Data Conflicts When Uploading Data from a Workbook."
`ServerContext`			This group of properties references the attribute bindings that uniquely identify the row displayed in the current worksheet so that you can reestablish server data context across multiple sessions. For more information, see Section 15.2, "Restore Server Data Context Between Sessions."
`ServerContext.CacheDataContexts`	Array	N	Add elements to the `CacheDataContexts` array for cases where there is more than one iterator defined in the binding container whose server-side context must be reestablished. The `CacheDataContexts` array supports the following properties to store the worksheet's cached data context: ■ `RestoreDataContextActionID` References an action binding to invoke. ■ `CachedServerContexts` An array that identifies the attribute binding values to cache and set before the action binding specified by `RestoreDataContextActionID` is invoked. Each element in the `CachedServerContext` array supports the `CachedAttributeID` and `RestoredAttributeID` properties. `CachedAttributeID` identifies the attribute binding value to cache in the worksheet. `RestoredAttributeID` is an optional property for which you specify a value when the destination attribute binding value is different from the source attribute binding value. If you do not specify a value for `RestoredAttributeID`, the value of `CachedAttributeID` is used as the destination attribute binding value and its value is set before invoking the action set. ■ `Annotation` For more information about this property, see Section A.1, "Frequently Used Properties in the ADF Desktop Integration."

Table A–19 (Cont.) Worksheet Properties

Name	Type	EL	Description
`ServerContext.IDAttributeID`	Binding	N	Specifies an attribute binding that uniquely identifies the row displayed in the current worksheet. This property is used at runtime to determine whether the server context has been reestablished properly for non-table type components in the worksheet.
`ServerContext.SendParameters`	Boolean	N	The default value of this property is `FALSE`. When set to `TRUE`, the workbook sends initialization parameters for this worksheet when reestablishing context across multiple sessions.
`Title`	String	Y	Specifies an EL expression that resolves to a string and sets the name of the worksheet. At design time, the EL expression can be of any length and can include the following special characters: `[ ] \ / * ?` At runtime, the evaluated string can display a maximum of 31 characters and ignores the above special characters. If the length of the evaluated string exceeds 31 characters, the extra characters are truncated and are not displayed. Ensure that the EL expressions you write for the `Title` property generate unique values for each worksheet at runtime and contain fewer than 31 characters. For example, if an EL expression generates a value for the `Title` property of an integrated worksheet that matches the name of an existing worksheet, an error occurs.

B

ADF Desktop Integration EL Expressions

This appendix describes the syntax for EL expressions in ADF Desktop Integration and provides guidelines for writing EL expressions.

This appendix includes the following sections:

- Section B.1, "Guidelines for Creating EL Expressions"
- Section B.2, "EL Syntax for ADF Desktop Integration Components"
- Section B.3, "Attribute Control Hints in ADF Desktop Integration"

B.1 Guidelines for Creating EL Expressions

The following list describes the characteristics that EL expressions for your integrated Excel workbook can have and provides recommendations for writing EL expressions:

- Literal values that evaluate correctly to the type expected for the Oracle ADF component property. The following list describes some examples:
  - Boolean values `true` and `false`
  - Integer values such as `-1`, `0`, and `100`
  - String values such as `hello world`
- Strings that contain one or more valid EL expression parts. The following list shows examples of valid syntax:
  - `#{row.bindings.ProductId.inputValue}`
  - `#{components.TAB416222534.errors}`
  - `#{res['excel.saveButton.label']}`
- A valid Excel formula. An Excel formula string must start with the = character. If the literal string includes an `#{...}` expression, ADF Desktop Integration evaluates this expression first and inserts the resulting value into the Excel formula string. Excel then evaluates the Excel formula.

  Note the following points if you write an EL expression:
  - Excel formula elements must not be used inside an `#{...}` expression.
  - EL expressions should not contain references to Excel cells because EL expressions are managed within ADF metadata. Excel cannot update the ADF metadata if the referenced cell moves. A workaround is to define a named cell reference or range using the **Name** box in the Excel Formula Bar. You can reference the named cell reference or named cell range reference from an EL

expression. For information about defining named cell references or ranges, see Excel's documentation.

- EL expressions in a page definition file

  For information about the syntax that you use to write EL expressions in a page definition file, see Section 4.3, "Working with Page Definition Files for an Integrated Excel Workbook."

B.2 EL Syntax for ADF Desktop Integration Components

Table B–1 lists supported expression properties for the ADF Desktop Integration components that support EL expressions.

The EL expressions use the following syntax to reference these properties:

`#{components.`*`componentID.property`*`}`

where *`componentID`* references the ID of the component and *`property`* references the property (for example, `rowCount`).

Table B–1 Expression Properties for ADF Desktop Integration Components

Property	Component Type	Property Type	Expected Runtime Values	Value at Design Time
`rowCount`	Table ROTable	Int	>=0	0
`currentRowIndex`	Table ROTable	Int	>= 0 AND < RowCount (zero based index)	-1
`currentRowMode`	Table	String	"insert" "update"	"unknown"
`errors`	Table	String	N/A	N/A
`readOnly`	Table.Column	Boolean	TRUE FALSE	FALSE

Write EL expressions with the following syntax to retrieve:

- Worksheet errors at runtime

  `#{worksheet.errors}`

  For more information about worksheet errors, see Section 12.4, "Error Reporting in an Integrated Excel Workbook."

- Workbook initialization parameters

  `#{workbook.params.`*`parameterName`*`}`

  where *`parameterName`* is the name of the workbook initialization parameter. For information about using these parameters, see Section 14.5, "Passing Parameter Values from a Fusion Web Application Page to a Workbook."

- Resource bundle string key values

  `#{`*`resourceBundleAlias`*`['`*`resourceBundleKey`*`']}`

  where *`resourceBundleAlias`* is the alias of the resource bundle and *`resourceBundleKey`* is the string key value. For more information about resource bundles, see Section 10.2, "Using Resource Bundles in an Integrated Excel Workbook."

Table B–2 describes the supported syntax and properties for Oracle ADF control bindings. For information about the attribute control hints (`controlHint`) that ADF Desktop Integration supports, see Table B–3.

You can use the expression builder described in Section 5.8, "Using the Expression Builder" to generate some of the EL expressions described in Table B–2. You have to write some other EL expressions as indicated in Table B–2.

Table B–2 Expression Properties and Syntax for Oracle ADF Control Bindings

Syntax	Component Type	Object Property	Value at Design Time

Use the expression builder to generate EL expressions with the following syntax:
`#{bindings.`*`attributeID`*`}`
`#{bindings.`*`attributeID`*`.label}`
`#{bindings.`*`attributeID`*`.hints.`*`controlHint`*`}`
 	
You can also write the previous EL expressions in addition to the following EL expression:
`#{bindings.`*`attributeID`*`.inputValue}`
	Attribute	Attribute control hint	" "

Use the expression builder to generate EL expressions with the following syntax:
`#{bindings.`*`ListID`*`}`
`#{bindings.`*`ListID`*`.label}`
`#{bindings.`*`ListID`*`.hints.`*`controlHint`*`}`
	List	Attribute control hint	" "

Write EL expressions with the following syntax for a column in a table-type component
`#{row.bindings.`*`attributeID`*`.inputValue}`
 	
Write an EL expression with the following syntax when adding a dynamic column to an ADF Table component as described in Section 7.15, "Adding a Dynamic Column to Your ADF Table Component":
`#{bindings.`*`TreeID`*`.[`*`TreeNodeID`*`].`*`AttributeNamePrefix`*`*.input Value}`
`#{bindings.`*`TreeID`*`.`*`AttributeNamePrefix`*`*.inputValue}`
`#{bindings.`*`TreeID`*`.[`*`TreeNodeID`*`].hints.`*`AttributeNamePrefix`*`* .`*`controlHint`*`}`
`#{bindings.`*`TreeID`*`.[`*`TreeNodeID`*`].hints.`*`AttributeNamePrefix`*`* .label}`
 	
A value for *`AttributeNamePrefix`* and *`[TreeNodeID]`* is optional while `*` is required.
	Table.Column	inputValue	" "

B.3 Attribute Control Hints in ADF Desktop Integration

ADF Desktop Integration can read the values of the attribute control hint names described in Table B–3. You write EL expressions that ADF Desktop Integration uses to retrieve the value of an attribute control hint from your Fusion web application.

Table B–2 describes the EL expression syntax that retrieves the values of attribute control hints at runtime.

You configure attribute control hints in your Fusion web application. Information about how to add an attribute control hint to an entity object can be found in the "Defining Attribute Control Hints for Entity Objects" section of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. Information about how to add an attribute control hint to a view object can be found in the "Defining Attribute Control Hints for View Objects" section of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Table B–3 Attribute Control Hints Used by ADF Desktop Integration

Attribute Control Hint	**Type**	**Value to configure in the Fusion web application**
`label`	String	References the value of the label attribute control hint configured for an entity or view object.
`updateable`	Boolean	Returns `true` if the associated attribute binding is updatable.

`readOnly`	Boolean	This attribute control hint is unique to ADF Desktop Integration. Returns `true` if the associated attribute binding is not updatable.
		To optimize the performance of an integrated Excel workbook when it evaluates Excel formulas in EL expressions, it is recommended that you write an EL expression with the following syntax for a component's `ReadOnly` property:
		`#{bindings.`*`attributeID`*`.hints.readOnly}`
		rather than:
		`=NOT(#{bindings.`*`attributeID`*`.hints.updateable})`
		Note that the attribute control hint `readOnly` property differs to the `ReadOnly` property of ADF Desktop Integration components described in Section A.1, "Frequently Used Properties in the ADF Desktop Integration."

`mandatory`	Boolean	Returns `true` if a value for the associated attribute binding is required.
`dataType`	String	Returns the data type of the attribute control hint. A Fusion web application can support many data types with complex names. The `dataType` attribute control hint was introduced in ADF Desktop Integration to simplify the writing of EL expressions. It maps the data types that a Fusion web application supports to the values supported by ADF Desktop Integration listed here: ▪ `string` ▪ `number` ▪ `date` ▪ `boolean` ▪ `other`

The ADF Desktop Integration attribute control hints are based on information available in the web application's model metadata. ADF Desktop Integration supports view object or entity object hint values, but does not support programmatic overrides of hint values if they are calculated at a row-by-row level at runtime.

C Troubleshooting an Integrated Excel Workbook

This appendix provides guidelines on how you can troubleshoot an integrated Excel workbook when you encounter problems during development. It also describes possible solutions for a number of problems that you may encounter.

This appendix includes the following sections:

Note: The property inspector does not validate that values you enter for a property or combinations of properties are valid. Invalid values may cause runtime errors. To avoid runtime errors, make sure you specify valid values for properties in the property inspector. For more information about the property inspector, see Section 5.6, "Using the Property Inspector."

C.1 Verifying That Your Fusion Web Application Supports ADF Desktop Integration

Using a specific URL, you can verify that the Fusion web application is running the ADF Desktop Integration remote servlet (`adfdiRemote`), and the version of ADF Desktop Integration. This information can be useful if you encounter errors with an integrated Excel workbook. For example, you can determine whether the ADF Desktop Integration remote servlet is running when you are troubleshooting an integrated Excel workbook.

To verify that the ADF Desktop Integration remote servlet is running:

1. Log on to the Fusion web application.
2. Type the concatenated values of the workbook properties `WebAppRoot` and `RemoteServletPath` into the address bar of your web browser. This corresponds to a URL similar to the following:

   ```
   http://hostname:7101/FusionApp/adfdiRemoteServlet
   ```

If the ADF Desktop Integration remote servlet is running, a web page returns displaying a message similar to the following:

```
ADF Desktop Integration Remote Servlet 11g (11.1.1.48.86) [520]
Response from oracle.adf.desktopintegration.servlet.DIRemoteServlet: OK.
```

C.2 Verifying End-User Authentication for Integrated Excel Workbooks

If end users of an integrated Excel workbook do not get prompted for user credentials when they invoke an action that interacts with the Fusion web application configured with ADF security, it may mean that security is not configured correctly for either the integrated Excel workbook or the Fusion web application. You can verify that your secure Fusion web application authenticates end users and that it is security-enabled by carrying out the following procedure.

To verify that a secure Fusion web application authenticates end users:

Enter the value of the workbook properties `WebAppRoot` into the address bar of your web browser. This corresponds to a URL similar to the following:

```
http://hostname:7101/FusionApp/adfdiRemoteServlet
```

If the Fusion web application is security-enabled, it will request that you enter user credentials.

For more information about securing your integrated Excel workbook, see Chapter 11, "Securing Your Integrated Excel Workbook."

C.3 Generating Log Files for an Integrated Excel Workbook

ADF Desktop Integration can generate log files that capture information based on events triggered by the following pieces of software within ADF Desktop Integration:

- HTTP filter and the ADF Desktop Integration remote servlet on the web server (server-side logging)

  For more information about server-side logging, see Section C.3.1, "About Server-Side Logging."

- Excel workbook which you integrate with your Fusion web application (client-side logging)

  For more information about client-side logging, see Section C.3.2, "About Client-Side Logging."

C.3.1 About Server-Side Logging

You configure the generation of server-side log files for ADF Desktop Integration the same way as for other Oracle ADF modules. This involves setting values that specify the verbosity level and output location in a configuration file named `j2ee-logging.xml`. You can also use Oracle Diagnostic Logging Configuration of JDeveloper to configure the logging levels specified in the `logging.xml` file. For more information about using the JDeveloper debugging tools and ADF Logger, see the "Using the ADF Logger" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Table C–1 describes the package names that you supply as attribute parameters to the `<logger>` elements in the `j2ee-logging.xml` file to configure log file generation in ADF Desktop Integration.

Table C–1 Package Names for Log File Configuration

To generate log file entries for this component...	Enter this package name...
All ADF Desktop Integration server logic	`oracle.adf.desktopintegration`
ADF Desktop Integration remote servlet	`oracle.adf.desktopintegration.servlet`
ADF Desktop Integration HTTP filter	`oracle.adf.desktopintegration.filter`

Table C–2 describes the types of information that the log file captures and the corresponding log file entry level.

Table C–2 Server-Side Logging Levels

Log Level	Description
`SEVERE (ERROR)`	Captures all exceptions and errors.
`WARNING`	Captures all irrecoverable problem conditions.
`INFO` or `CONFIG`	Captures lifecycle events such as servlet initialization, and so on.
`CONFIG`	Captures "heartbeat" events that echo status and execution context for each client-server interaction.

`FINE`
`FINER`
`FINEST`
	These values generate increasing levels of diagnostic information.

C.3.2 About Client-Side Logging

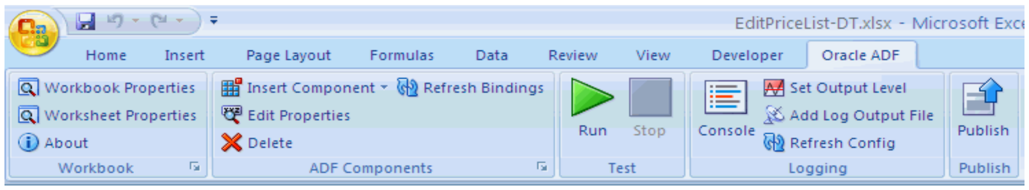
You can configure ADF Desktop Integration to save logs of triggered events on the client. By default, no log files are generated. For more information about how to configure the Oracle ADF Desktop Integration module to save logs, see Section C.3.2.1, "How to Configure ADF Desktop Integration to Save Logs.".

C.3.2.1 How to Configure ADF Desktop Integration to Save Logs

ADF Desktop Integration provides logging tools to generate event logs and make them easily accessible. The logging tools are located in the Logging group of the **Oracle ADF** tab, and are available in both the design mode and the test mode.

Figure C–1 shows the logging tools in the **Oracle ADF** tab.

Figure C–1 Logging Tools in Oracle ADF Tab



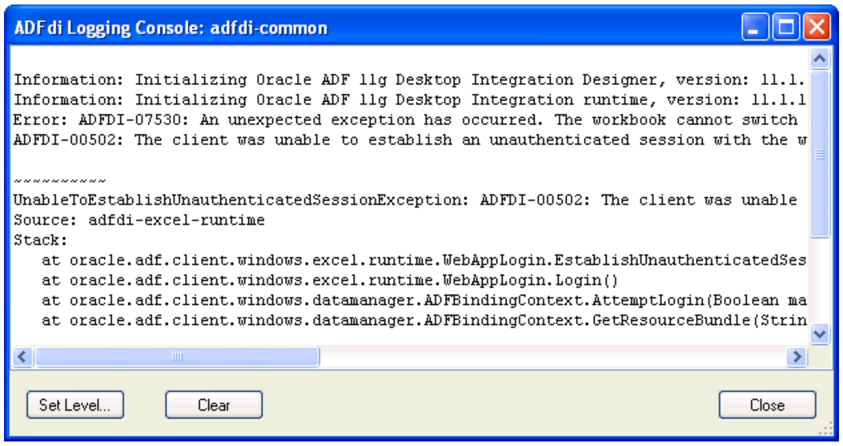
The Logging group provides the following buttons:

- **Console**

Displays the ADFdi Logging Console window, which enables you to review the recent log entries while you are developing and testing the integrated Excel workbook. The console displays entries that are logged while the console is open. Figure C–2 illustrates the ADFdi Logging Console window with information and error log entries.

The console is a resizable, non-modal window with a buffer size of 64,000 characters. When the buffer is full, the old entries are removed. If you want to save log entries, you can copy them to a text file.

Figure C–2 ADFdi Logging Console Window



```
Information: Initializing Oracle ADF 11g Desktop Integration Designer, version: 11.1.
Information: Initializing Oracle ADF 11g Desktop Integration runtime, version: 11.1.1
Error: ADFDI-07530: An unexpected exception has occurred. The workbook cannot switch
ADFDI-00502: The client was unable to establish an unauthenticated session with the w

~~~~~~~~~~
UnableToEstablishUnauthenticatedSessionException: ADFDI-00502: The client was unable
Source: adfdi-excel-runtime
Stack:
   at oracle.adf.client.windows.excel.runtime.WebAppLogin.EstablishUnauthenticatedSes
   at oracle.adf.client.windows.excel.runtime.WebAppLogin.Login()
   at oracle.adf.client.windows.datamanager.ADFBindingContext.AttemptLogin(Boolean ma
   at oracle.adf.client.windows.datamanager.ADFBindingContext.GetResourceBundle(Strin
```

The dialog has the following buttons:

– **Set Level:** Click to set the log output level. The button opens the Logging Output Level dialog, where you can choose the desired log output level.

– **Clear:** Click to clear the log buffer.

– **Close:** Click to close the dialog.

Note: A common ADFdi Logging Console window logs entries for all open integrated Excel workbooks.

- **Set Output Level**

  Prompts you to choose the log output level. Table C–3 describes the log levels that client-side logging supports.

Figure C–3 Logging Output Level Dialog

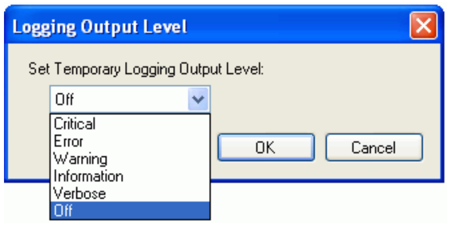


Table C–3 Client-Side Logging Levels

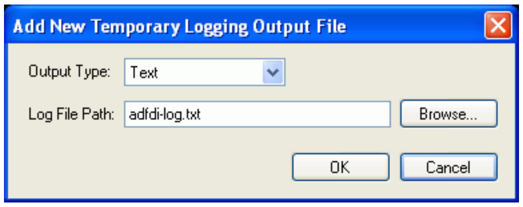
Level	Description
`Critical`	Captures critical information.
`Error`	Captures information about severe errors and exceptions.
`Warning`	Captures irrecoverable conditions.
`Information`	Captures lifecycle and control flow events.
`Verbose`	Captures detailed information about the execution flow of the application.
`Off`	No logs are captured. This is the default value.

Note: The log output level applies to all listeners for a given logger.

- **Add Log Output File**

  Creates a new temporary logging listener to direct logging output to the specified file or format. In the Add New Temporary Logging Output File dialog, choose the desired file output type (text or XML), and specify the path and file name of the log output file.

Figure C–4 Add New Temporary Logging Output File Dialog



Output Type: Text
Log File Path: adfdi-log.txt

The temporary listener directs the logging output for the current Excel session only, and is not registered in the ADF Desktop Integration configuration file. After you close the integrated Excel workbook, the temporary listener is removed.

Note: When you click the **Add Log Output File** button, a new listener is created. The new listener does not replace any existing listener defined in the ADF Desktop Integration configuration file, or any other temporary listener.

- **Refresh Config**

  Reloads the ADF Desktop Integration configuration file. The ADF Desktop Integration configuration file determines the type of information logged by ADF Desktop Integration. It also determines the location and the output format of the log file.

  For more information about the creation and configuration of the ADF Desktop Integration configuration file, see Section C.3.2.2, "About the ADF Desktop Integration Configuration File."

C.3.2.2 About the ADF Desktop Integration Configuration File

The ADF Desktop Integration configuration file is saved as `adfdi-excel-addin.dll.config` in the Designer edition, and as `adfdi-excel-addin-runtime.dll.config` in the Runtime edition. To determine the correct file name and location, click the **About** button in the Workbook group of the **Oracle ADF** tab. In the dialog that opens, click the **Properties** tab, and consult the **Configuration** entry for file name and location of configuration file.

For more information about elements of the configuration file, see the "Configuration File Schema for the .NET Framework" section in Microsoft Developer Network documentation. For more information about trace and debug settings, see the "Trace and Debug Settings Schema" section in Microsoft Developer Network documentation.

Example C–1 shows a sample configuration file, one of many valid ways to configure client-side logging, that generates two different log files with different formats (`.txt` and `.xml`). The file captures different types of information such as `ThreadId`, `ProcessId`, and `DateTime` at a `Verbose` logging level.

Example C–1 Sample Configuration File

```
<?xml version="1.0"?>
<configuration>
  <system.diagnostics>
    <sources>
      <source name="adfdi-common" switchValue="Verbose">
        <listeners>
          <add type="System.Diagnostics.DelimitedListTraceListener"
            name="adfdi-common-excel.txt"
            initializeData="c:\logs\adfdi-common-excel.txt"
            delimiter="|"
            traceOutputOptions="ThreadId, ProcessId, DateTime"/>
          <add type="System.Diagnostics.XmlWriterTraceListener"
            name="adfdi-common-excel.xml"
            initializeData="c:\logs\adfdi-common-excel.xml"
            traceOutputOptions="None"/>
        </listeners>
      </source>
    </sources>
  </system.diagnostics>
</configuration>
```

C.3.2.3 How to Configure Logging Using User Environment Variables

Users who do not have access to the directory that stores the ADF Desktop Integration configuration file can change the location where log files are saved, and the logging level by setting values for user environment variables. You can add two user environment variables to configure the logging level and location for XML log files.

To add or configure user environment variables on Windows:

1. Click the Windows **Start** button and then click **Settings** > **Control Panel**.
2. In the Control Panel, double-click **System**.
3. In the System Properties dialog, click the **Advanced** tab, and then click the **Environment Variables** button.
4. In the Environment Variables dialog, click **New** under the *User variables for **username*** input field, and add variables as described in the following table.

Table C–4 User Enviroment Variables to Configure Logging

Enter a variable named...	With a value...
`adfdi-common-file`	That defines the directory path and file name for the XML file that captures logging information. The directory that you specify here must exist before you add the `adfdi-common-file` variable. The generated log file will be in XML format.
`adfdi-common-level`	That specifies the level of logging. Table C–3 lists valid values.

5. Click **OK**.

C.3.2.4 What You May Need to Know About the adfdi-common Object

The `adfdi-common` object is an instance of the `TraceSource` class from the `System.Diagnostics` namespace in the Microsoft .NET Framework. This object is used to generate log files that capture information about events triggered by the Excel workbook that you integrate with your Fusion web application.

For more information about the `TraceSource` class, see Microsoft Developer Network documentation.

C.4 Exporting Excel Workbook Metadata

You can export the XML metadata in your Excel workbook to an XML file with a name and location that you specify. This file may be useful if you have to debug or analyze an Excel workbook that is integrated with a Fusion web application. It contains child elements for each worksheet in the workbook, resources such as the relative path to the remote servlet, and so on.

The following procedure describes how you export XML metadata from an Excel workbook.

To export XML metadata from an integrated Excel workbook:

1. Click **About** in the **Oracle ADF** tab.

   The About Oracle ADF 11g Desktop Integration dialog box appears.

2. Click the **Properties** view tab and then click the **Export Metadata** button.

   A dialog box appears that asks you to specify a file name and location for the file that stores the exported metadata.

3. Specify a file name, a location, and then click **Save**.

   The integrated Excel workbook exports the metadata to the specified file in the specified format.

C.5 Common ADF Desktop Integration Problems

This section describes the most common problems and their solutions.

Error message: [ADFDI-00127] A version mismatch was detected for `SyncServletResponse`**. Version *x* was found, version *y* was expected**

Cause: The client version of ADF Desktop Integration does not match the ADF Desktop Integration version in the web application.

Action: Uninstall client ADF Desktop Integration, and install the web application specific ADF Desktop Integration version. For more information about installing ADF Desktop Integration client, see Section 3.5, "Installing ADF Desktop Integration."

Error message: 404 Error - servlet not found

Cause: The `web.xml` deployment descriptor settings are not in sync with `Workbook.RemoteServletPath` property value.

Action: Open Workbook Properties editor and verify the `Workbook.RemoteServletPath` property value.

Error message: Uncaught exception thrown by method called through Reflection. (Exception from HRESULT: 0x80131604)

Cause: Microsoft .NET Programmability Support is not enabled.

Action: Enable Microsoft .NET Programmability Support. For more information, see Section 3.3, "Enabling Microsoft .NET Programmability Support."

Problem: Oracle ADF tab is not visible in your integrated Excel Workbook after installing ADF Desktop Integration

Cause: The ADF Desktop Integration add-in is not enabled in Excel.

Action: Enable the ADF Desktop Integration add-in in the Excel Options dialog. In Excel, click the Microsoft Office button, and then click **Excel Options** to open the Excel Options dialog. In the Add-Ins tab, open the **Manage** dropdown list, choose **COM Add-ins**, and click **Go**. In the COM Add-ins dialog, select the **Oracle ADF 11g Desktop Integration Add-in for Excel** checkbox and click **OK**.

D Using Workbook Management Tools

This appendix describes how to use the workbook administration tool. You can use the tool to manage workbooks that you integrate with a Fusion web application.

This appendix includes the following sections:

- Section D.1, "Using the Workbook Administration Tool"

D.1 Using the Workbook Administration Tool

Use the workbook administration tool to set values for a number of workbook properties, such as `WebAppRoot`, after you publish the finalized integrated Excel workbook. Use this tool if, for example, the URL of your Fusion web application changes after you publish the integrated Excel workbook.

You can also use it when you want to change workbook settings but cannot, or do not want to set values for the HTTP filter as described in Section E.2, "Configuring the ADF Desktop Integration Excel Download Filter."

The workbook administration tool is a Java-based program that can be executed on operating systems that support the version of Java used by Oracle ADF. It also requires access to the `adf-desktop-integration-admin-tool.jar` file, which is located in the following directory:

MW_HOME`\jdeveloper\adfdi\lib`

The other requirements for components or utilities in ADF Desktop Integration, as outlined in Chapter 3, "Setting Up Your Development Environment," do not apply to the workbook administration tool.

To change workbook settings using the workbook administration tool:

- Open a command line console and execute the following command:

```
java -cp adf-desktop-integration-admin-tool.jar
oracle.adf.desktopintegration.admintool.WorkbookAdminTool <arg(s)>
```

where `<arg(s)>` is one or more of the required or optional arguments that are described in Table D–1.

Table D–1 Command-line Options for the Workbook Administration Tool

Provide a value for this argument...	To...	Is a value for this argument required?
`-workbook`	Specify the directory path to the workbook to update.	Yes
`-root`	Set the value for this property to the fully qualified URL for the web application to integrate your desktop application with.	No
`-mode`	Change the workbook mode to one of the following: ■ `RT` where `RT` specifies runtime mode. ■ `DT` where `DT` specifies design mode. ■ `TST` where `TST` specifies test mode. For more information about workbook modes, see Section 5.1, "Introduction to Development Tools."	No
`-out`	Specify the directory path and file name for the output file.	Yes
`-quiet`	Specify this argument if you do not want to generate verbose output.	No
`-help`	Print help information.	No

The following command creates a copy of the workbook (`text.xlsx`) in runtime mode (`RT`) for a Fusion web application (`http://hostname:7101/FODADFBC`) and writes it to a directory with a new file name (`myresult.xlsx`):

```
java -cp adf-desktop-integration-admin-tool.jar
oracle.adf.desktopintegration.admintool.WorkbookAdminTool -workbook test.xlsx
-mode RT -root http://hostname:7101/FODADFBC -out myresult.xlsx
```

E

ADF Desktop Integration Settings in the Web Application Deployment Descriptor

This appendix describes the values that you set for the ADF Desktop Integration servlet (`adfdiRemote`) so that the Fusion web application can use it. The appendix also describes the values in the deployment descriptor file that determine the behavior of the HTTP filter that ADF Desktop Integration provides. Finally, it provides an extract from a deployment descriptor file that shows these values in use.

This appendix includes the following sections:

- Section E.1, "Configuring the ADF Desktop Integration Servlet"
- Section E.2, "Configuring the ADF Desktop Integration Excel Download Filter"
- Section E.3, "Examples in a Deployment Descriptor File"

Note: Adding ADF Desktop Integration and ADF Library Web Application Support to the technology scope of your desktop integration project automatically generates the entries in the `web.xml` file discussed in this appendix. For more information, see Section 4.2, "Adding ADF Desktop Integration to a Fusion Web Application."

E.1 Configuring the ADF Desktop Integration Servlet

A Fusion web application with integrated Excel workbooks must contain entries in its deployment descriptor file (`web.xml`) to use the `adfdiRemote` servlet. The Excel workbooks that you integrate with a Fusion web application call this servlet to synchronize data with the Fusion web application. The `adf-desktop-integration.jar` file stores the servlet in the following directory:

```
MW_HOME\oracle_common\modules\oracle.adf.desktopintegration_
11.1.1
```

where *MW_HOME* is the Middleware Home directory.

When you add ADF Desktop Integration to the technology scope of your project as described in Section 4.2, "Adding ADF Desktop Integration to a Fusion Web Application," ADF Desktop Integration automatically configures your deployment descriptor with the necessary entries to enable the servlet (`DIRemoteServlet`) on your Fusion web application. If required, then you can configure the servlet manually.

To configure the ADF Desktop Integration servlet:

1. In JDeveloper, locate and open the deployment descriptor file (`web.xml`) for your ADF Desktop Integration project.

Typically, this file is located in the `WEB-INF` directory of your project.

2. Click the Servlets page, and then click the **Add** icon to create a row entry in the Servlets table. The icon is in the top-right corner of the servlets table.

   Enter the values as described in Table E–1 to enable the `adfdiRemote` servlet on the Fusion web application.

Table E–1 Values to Enable adfdiRemote Servlet

For this property...	Enter this value...
Name	`adfdiRemote`
Type	Servlet Class
Servlet Class/JSP file	`oracle.adf.desktopintegration.servlet.DIRemoteServlet`

3. In Servlets page, click the **Servlet Mappings** tab, and then click the **Add** icon to create a row in the Servlet Mapping table.

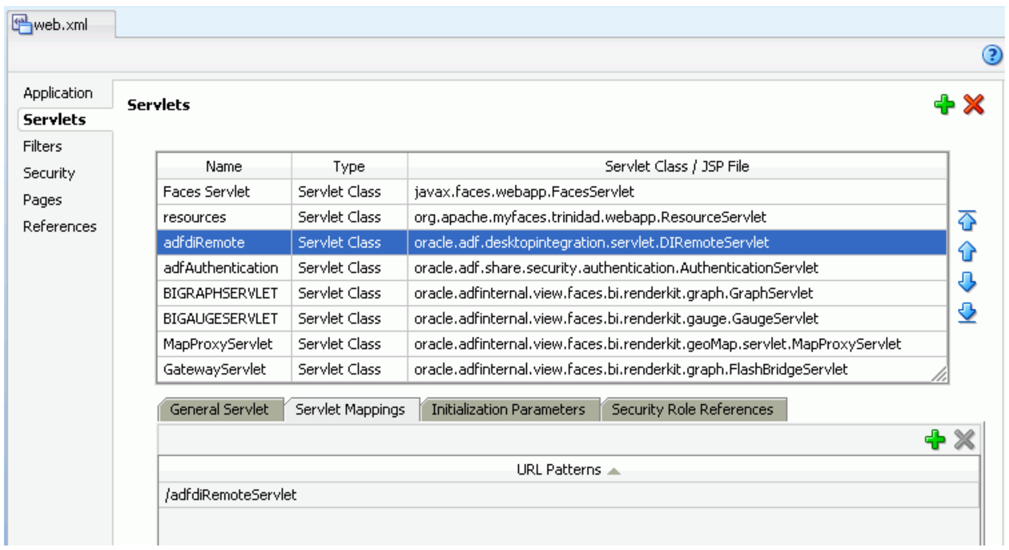
   Enter the value as described in Table E–2 to add a URL pattern for the `adfdiRemote` servlet in the Fusion web application. The value that you enter must match the value that you specify in the integrated Excel workbook for the `RemoteServletPath` workbook property. Note that values are case sensitive.

Table E–2 Values to Add A URL Pattern to adfdiRemote Servlet

For this property...	Enter this value...
URL Patterns	`/adfdiRemoteServlet`

Figure E–1 displays the Servlets page of `web.xml` of Master Price List module.

Figure E–1 Servlets Page of Deployment Descriptor



Name	Type	Servlet Class / JSP File
Faces Servlet	Servlet Class	javax.faces.webapp.FacesServlet
resources	Servlet Class	org.apache.myfaces.trinidad.webapp.ResourceServlet
adfdiRemote	Servlet Class	oracle.adf.desktopintegration.servlet.DIRemoteServlet
adfAuthentication	Servlet Class	oracle.adf.share.security.authentication.AuthenticationServlet
BIGRAPHSERVLET	Servlet Class	oracle.adfinternal.view.faces.bi.renderkit.graph.GraphServlet
BIGAUGESERVLET	Servlet Class	oracle.adfinternal.view.faces.bi.renderkit.gauge.GaugeServlet
MapProxyServlet	Servlet Class	oracle.adfinternal.view.faces.bi.renderkit.geoMap.servlet.MapProxyServlet
GatewayServlet	Servlet Class	oracle.adfinternal.view.faces.bi.renderkit.graph.FlashBridgeServlet

URL Patterns
/adfdiRemoteServlet

4. Click the Filters page, and verify that whether a `adfBindings` filter exists in the Filters table. If an entry exists, select it and proceed to the next step. If there is no such entry, then click the **Add** icon to create a row entry in the Filters table. The icon is available in the top-right corner of the filters table.

Enter the values as described in Table E–3 to add the ADF binding filter to the `adfdiRemote` servlet.

Table E–3 Values to Add Binding Filter to adfdiRemote Servlet

For this property...	Enter this value...
Name	`adfBindings`
Class	`oracle.adf.model.servlet.ADFBindingFilter`

5. In Filters page, click the **Filter Mappings** tab, and then click the **Add** icon to create a row in the Filter Mapping table.

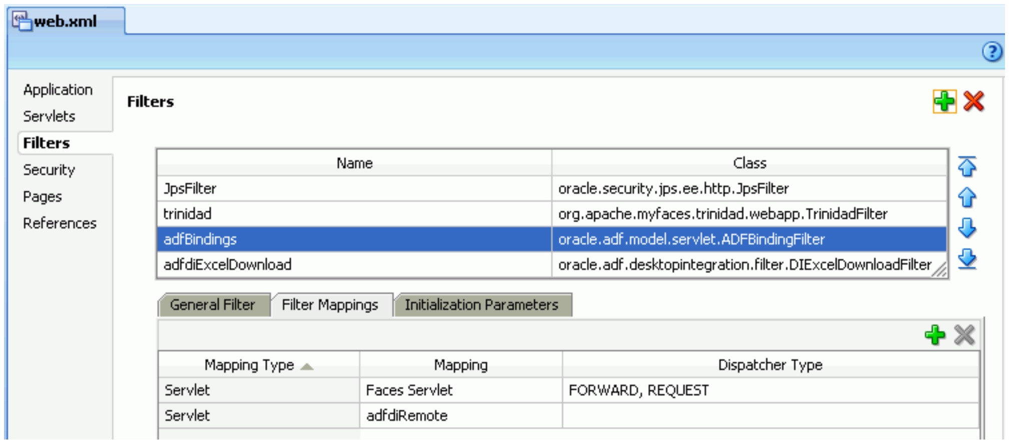
   Enter the values as described in Table E–4 to add the mapping filter to the `adfdiRemote` servlet. The filter mapping must match with the Servlet name in Step 2.

Table E–4 Values to Add Mapping Filter to adfdiRemote Servlet

For this property...	Enter this value...
Mapping Type	Servlet
Mapping	`adfdiRemote`

Figure E–2 displays the Filters page of `web.xml` of Master Price List module.

Figure E–2 Filters Page of Deployment Descriptor



Name	Class
JpsFilter	oracle.security.jps.ee.http.JpsFilter
trinidad	org.apache.myfaces.trinidad.webapp.TrinidadFilter
adfBindings	oracle.adf.model.servlet.ADFBindingFilter
adfdiExcelDownload	oracle.adf.desktopintegration.filter.DIExcelDownloadFilter

Mapping Type	Mapping	Dispatcher Type
Servlet	Faces Servlet	FORWARD, REQUEST
Servlet	adfdiRemote	

6. Save the deployment descriptor file, and then rebuild your ADF Desktop Integration project to apply the changes you made.

E.2 Configuring the ADF Desktop Integration Excel Download Filter

ADF Desktop Integration includes an HTTP filter in the `adf-desktop-integration.jar` stored in the following directory:

`MW_HOME``\oracle_common\modules\oracle.adf.desktopintegration_11.1.1`

where *`MW_HOME`* is the Middleware Home directory.

You configure an entry in the deployment descriptor file (`web.xml`) of your Fusion web application so that the application invokes the HTTP filter to make changes in an integrated Excel workbook before the integrated Excel workbook is downloaded by the end user from the Fusion web application. These changes ensure that the integrated Excel workbook functions correctly when the end user opens it. The HTTP filter makes the following changes:

- `WebAppRoot`

  Sets the value for this property to the fully qualified URL for the Fusion web application from which the end user downloads the integrated Excel workbook.

- Workbook mode

  Changes the integrated Excel workbook mode to runtime mode in case the workbook was inadvertently left in design mode or test mode.

> **Note:** If you choose not to use the `adfdiExcelDownload` filter, you can instead use the workbook administration tool to set the `WebAppRoot` property on your workbooks. For more information, see Section D.1, "Using the Workbook Administration Tool."

By default, JDeveloper adds the HTTP filter to your ADF Desktop Integration project when you add ADF Desktop Integration to the technology scope of your project as described in Section 4.2, "Adding ADF Desktop Integration to a Fusion Web Application."

To configure the HTTP filter:

1. In JDeveloper, locate and open the deployment descriptor file (`web.xml`) for your ADF Desktop Integration project.

   Typically, this file is located in the `WEB-INF` directory of your project.

2. Click the Filters page, and verify that a `adfBindings` filter exists in the Filters table. If an entry exists, select it and proceed to the next step. If there is no such entry, then click the **Add** icon to create a row entry in the Filters table.

   Enter the values as described in Table E–5 to create a filter, or configure the values to modify the existing HTTP filter.

Table E–5 Properties to Configure HTTP Filter

For this property...	Enter this value...
Name	`adfdiExcelDownload`
Class	`oracle.adf.desktopintegration.filter.DIExcelDownloadFilter`
Display Name	(Optional) In General Filter tab, enter a display name for the filter that appears in JDeveloper.
Description	(Optional) In General Filter tab, enter a description of the filter.

3. In the Filters page, click the **Filter Mappings** tab, and then click the **Add** icon to create a row in Filter Mapping table.

   Add a filter mapping for integrated Excel workbooks that use the default file format (.xlsx) by entering values as described in Table E–6.

Table E–6 Properties to Configure Filter Mappings

For this property...	Enter this value...
Mapping Type	URL Pattern
Mapping	`*.xlsx`
Dispatcher Type	No value is required for this property.

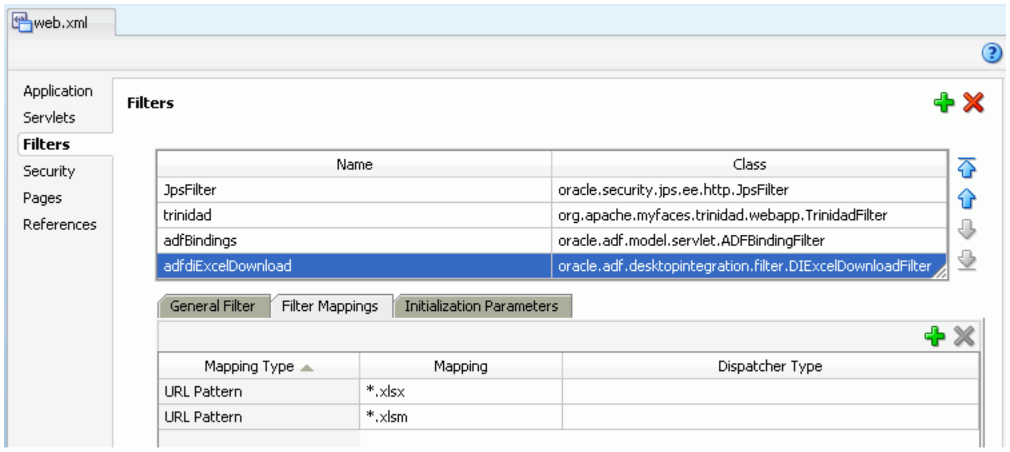
4. Add another filter mapping for integrated Excel workbooks that use the macro-enabled workbook format (.xlsm) by entering values as described in Table E–7.

Table E–7 Properties to Configure Filter Mappings

For this property...	Enter this value...
Mapping Type	URL Pattern
Mapping	`*.xlsx`
Dispatcher Type	No value is required for this property.

Figure E–3 displays the Filters page of `web.xml` of Master Price List module.

Figure E–3 Filters Page of Deployment Descriptor



Name	Class
JpsFilter	oracle.security.jps.ee.http.JpsFilter
trinidad	org.apache.myfaces.trinidad.webapp.TrinidadFilter
adfBindings	oracle.adf.model.servlet.ADFBindingFilter
adfdiExcelDownload	oracle.adf.desktopintegration.filter.DIExcelDownloadFilter

Mapping Type	Mapping	Dispatcher Type
URL Pattern	*.xlsx	
URL Pattern	*.xlsm	

5. Click the Application page, expand MIME Mappings section, and click the **Add** icon.

   Add a MIME type for integrated Excel workbooks that use the default file format (.xlsx) by entering values as described inTable E–8.

Table E–8 Properties to Add MIME Mappings

For this property...	Enter this value...
Extension	`*.xlsx`
MIME Type	`application/vnd.openxmlformats-officedocument.spreadsheetml.sheet`

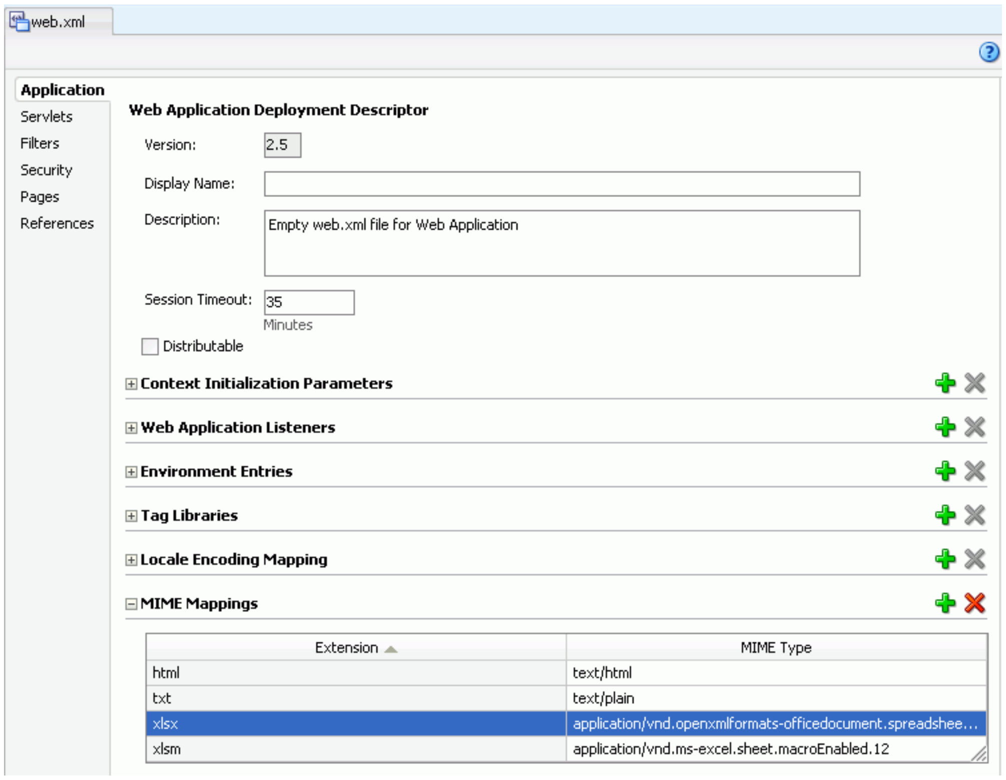
6. Add another MIME type for integrated Excel workbooks that use the macro-enabled workbook format (.xlsm) by entering values as described in Table E–9.

Table E–9 Properties to Add MIME Mappings

For this property...	Enter this value...
Extension	`*.xlsm`
MIME Type	`application/vnd.ms-excel.sheet.macroEnabled.12`

Figure E–4 displays the Application page of `web.xml` of Master Price List module.

Figure E–4 Application Page of Deployment Descriptor



Version: 2.5
Display Name:
Description: Empty web.xml file for Web Application
Session Timeout: 35 Minutes
Distributable

Extension	MIME Type
html	text/html
txt	text/plain
xlsx	application/vnd.openxmlformats-officedocument.spreadshee...
xlsm	application/vnd.ms-excel.sheet.macroEnabled.12

7. Save the deployment descriptor file, and then rebuild your ADF Desktop Integration project to apply the changes you made.

While updating filter and filter mapping information in the `web.xml` file, ensure that the filter for ADF Library Web Application Support (`<filter-name>ADFLibraryFilter</filter-name>`) appears below the `adfdiExcelDownload` filter entries, so that integrated Excel workbooks can be downloaded from the Fusion web application.

E.3 Examples in a Deployment Descriptor File

The following extracts from the `web.xml` file of a Fusion web application with ADF Desktop Integration in its technology scope show the entries that you configure for a desktop integration project. For more information ordering of filters, see Section 4.2.2, "What Happens When You Add ADF Desktop Integration to Your JDeveloper Project."

```
<filter>
    <filter-name>adfBindings</filter-name>
    <filter-class>
        oracle.adf.model.servlet.ADFBindingFilter</filter-class>
    </filter>
    <filter>
        <filter-name>adfdiExcelDownload</filter-name>
        <filter-class>
            oracle.adf.desktopintegration.filter.DIExcelDownloadFilter
        </filter-class>
    </filter>
    <filter-mapping>
        <filter-name>adfBindings</filter-name>
        <servlet-name>adfdiRemote</servlet-name>
    </filter-mapping>
    <filter-mapping>
        <filter-name>adfdiExcelDownload</filter-name>
        <url-pattern>*.xlsx</url-pattern>
    </filter-mapping>
    <filter-mapping>
        <filter-name>adfdiExcelDownload</filter-name>
        <url-pattern>*.xlsm</url-pattern>
    </filter-mapping>
    <servlet>
        <servlet-name>adfdiRemote</servlet-name>
        <servlet-class>
            oracle.adf.desktopintegration.servlet.DIRemoteServlet
        </servlet-class>
    </servlet>
<servlet-mapping>
    <servlet-name>adfdiRemote</servlet-name>
    <url-pattern>/adfdiRemoteServlet</url-pattern>
</servlet-mapping>
<mime-mapping>
    <extension>xlsx</extension>
    <mime-type>
        application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
    </mime-type>
</mime-mapping>
<mime-mapping>
    <extension>xlsm</extension>
    <mime-type>
        application/vnd.ms-excel.sheet.macroEnabled.12
    </mime-type>
</mime-mapping>
```

F

String Keys in the Overridable Resources

This appendix describes the string keys in the reserved resource bundle that you can override.

Table F–1 lists the string keys and their current English values. Create a resource bundle where you define the string keys in Table F–1 and the values that you want to appear at runtime. For information about how to override the reserved resource bundle, see Section 10.2.3, "How to Override Resources That Are Not Configurable."

Table F–1 String Keys and Values in the Reserved Resource Bundle

Area where string key value appears at runtime	String key	English value in the ADF Desktop Integration reserved resource bundle	Comment
`Upload Options`	`UPLOAD_OPTIONS_TITLE`	Upload Options	
`Upload Options`	`UPLOAD_OPTIONS_PROMPT`	Specify options to use during the Upload operation	
`Upload Options`	`UPLOAD_OPTIONS_CONTINUE_ON_FAIL_LABEL`	On failure, continue to upload subsequent rows	
`Upload Options`	`UPLOAD_OPTIONS_DOWNLOAD_AFTER_LABEL`	Download all rows after successful upload	
`Table.Download`	`DOWNLOAD_OVERWRITE_TITLE`	Download	
`Table.Download`	`DOWNLOAD_OVERWRITE_PROMPT`	Do you wish to discard the pending changes?	
`Table.Download`	`ROWLIMIT_WARNINGS_TITLE`	Row Limit Exceeded	
`Table.Initialize`	`INITIALIZE_OVERWRITE_TITLE`	Initialize	
`Table.Initialize`	`INITIALIZE_OVERWRITE_PROMPT`	Do you wish to discard the pending changes?	
`Workbook.ClearAllData`	`CLEARDATA_CONFIRM_TITLE`	Clear All Data	
`Workbook.ClearAllData`	`CLEARDATA_CONFIRM_PROMPT`	This command will log you out of your current session and clear all the data from all worksheets in the workbook. Are you sure?	
`Workbook.Logout`	`LOGOUT_STATUS_TITLE`	Logout	
`Workbook.Logout`	`LOGOUT_STATUS_PROMPT`	You have been logged out from your current session.	
`Table.Upload`	`COMPONENTS_TABLE_DYN_COLS_NOT_AVAIL_TITLE`	Upload	

Table F–1 (Cont.) String Keys and Values in the Reserved Resource Bundle

Area where string key value appears at runtime	String key	English value in the ADF Desktop Integration reserved resource bundle	Comment
`Table.Upload`	`COMPONENTS_TABLE_DYN_COLS_NOT_AVAIL_PROMPT`	One or more dynamic columns is no longer available, do you wish to continue?	
Table status	`UPLOAD_STATUS_NO_UPDATES`	No updates detected	
Table status	`TABLE_UPLOAD_RECORD_NOT_FOUND`	Record not found	
Table status	`TABLE_UPLOAD_CANNOT_INSERT_MORE_THAN_ONCE`	Cannot insert record more than once	
Table status	`TABLE_COMMIT_FAILED_1`	See Error Detail {0}	{0} is a batch number
Table status	`TABLE_COMMIT_FAILURE_DETAILS_2`	Error Detail {0}:{1}	{0} is a batch number {1} is an error message
Table status	`TABLE_UPLOAD_ROW_UPDATE_SUCCESS`	Row updated successfully	
Table status	`TABLE_UPLOAD_ROW_INSERT_SUCCESS`	Row inserted successfully	
Table status	`TABLE_UPLOAD_ROW_UPDATE_FAILURE`	Update failed	
Table status	`TABLE_UPLOAD_ROW_INSERT_FAILURE`	Insert failed	
Table status	`TABLE_DELETE_ROW_FAILURE`	Delete failed	
Table status	`MESSAGE_DETAILS_NONE`	No error details available.	
Table status	`MESSAGE_DETAILS_ROW_TITLE`	Row Errors	
Table status	`MESSAGE_DETAILS_ROW_PROMPT`	Errors for this row:	
Table status	`MESSAGE_DETAILS_TABLE_TITLE`	Table Errors	
Table status	`MESSAGE_DETAILS_TABLE_PROMPT`	Error details for this table:	
Table status Table errors Worksheet errors	`MESSAGE_DETAILS_HELP_LABEL`	Click on each error to reveal additional information.	Appears in the error list.
Table status Table errors Worksheet errors	`MESSAGE_LABEL_DEFAULT_CONTEXT`	Action	
Worksheet errors	`MESSAGE_DETAILS_WORKSHEET_TITLE`	Worksheet Errors	
Worksheet errors	`MESSAGE_DETAILS_WORKSHEET_PROMPT`	Error details for this worksheet:	
Worksheet errors	`MESSAGE_DETAILS_PARSE_FAILURE`	A problem has occurred while retrieving the error details. The information is no longer available.	
Worksheet errors	`MESSAGE_LABEL_FAILED_1`	{0} failed	{0} is a context label
`Workbook.Login`	`LOGIN_WINDOW_TITLE`	Login	

Table F–1 (Cont.) String Keys and Values in the Reserved Resource Bundle

Area where string key value appears at runtime	String key	English value in the ADF Desktop Integration reserved resource bundle	Comment
`Workbook.Login`	`LOGIN_CONFIRM_CONNECT_2`	You are about to connect to the following application:\n{0}\nand\n{1}\n\nDo you want to connect?	{0} and {1} are the URLs the application uses.
`Workbook.EditOptions`	`SETTINGS_EDIT_TITLE`	Edit Options	
`Workbook.EditOptions`	`SETTINGS_EDIT_PROMPT`	Enter a value for Web App Root. For example: 'http://localhost:1234/MyApp'.	
`Workbook.EditOptions`	`SETTINGS_CONFIRM_TITLE`	Web App Root	
`Workbook.EditOptions`	`SETTINGS_CONFIRM_PROMPT`	Changing the Web App Root will log you out of your current session and clear all the data from all worksheets in the workbook. Are you sure?	

G
Java Data Types Supported By ADF Desktop Integration

This appendix lists the Java data types that an ADF Desktop Integration project supports.

Primitive Java Types

- `long`
- `int`
- `short`
- `boolean`
- `double`
- `float`

Object Java Types

- `java.lang.Long`
- `java.lang.Integer`
- `java.lang.Short`
- `java.lang.Boolean`
- `java.lang.String`
- `oracle.jbo.domain.Date`
- `oracle.jbo.domain.Timestamp`
- `oracle.jbo.domain.TimestampLTZ`
- `oracle.jbo.domain.TimestampTZ`
- `java.util.Date`
- `java.sql.Date`
- `java.sql.Time`
- `java.sql.Timestamp`
- `java.lang.Double`
- `java.lang.Float`
- `java.math.BigDecimal`
- `oracle.jbo.domain.RowID`

- `oracle.jbo.domain.Number`

H Using ADF Desktop Integration Model API

This appendix describes how to use the ADF Desktop Integration Model API library to enable custom `ApplicationModule` methods to access attribute values passed during upload process when there are no actual rows available in a tree node binding.

This appendix includes the following sections:

- Section H.1, "About the Temporary Row Object"
- Section H.2, "About ADF Desktop Integration Model API"
- Section H.3, "ADF Desktop Integration Model API Classes and Methods"

H.1 About the Temporary Row Object

Each ADF Table component is bound to a tree binding defined within a page definition. Each tree control binding has one (or more) tree nodes defined. For parent-child relationships, the tree binding has two nodes, one for parent table and another for child table. At runtime, the ADF Table component displays both parent and child attributes within each worksheet row. On upload, ADF Desktop Integration sets attribute values to both the parent and child nodes.

In certain situations, a particular tree node may not have actual data rows available during `Table.Upload` request processing. Two common scenarios where a tree node may not have data are:

- The tree node's iterator result set does not have any data rows available. This could be because of a query returning zero rows.
- In a parent-child relationship, if the foreign key has not been populated in the parent table, the link between parent and child tree node may not contain actual rows.

There may be certain cases when, even though there is no actual row available on the server, you still want to allow the end user to enter values in the worksheet and upload them to the server. During upload, ADF Desktop Integration creates a temporary row object and stores the values uploaded from the worksheet row. Using the ADF Desktop Integration Model API, you can write custom Java code to access the temporary row object and collect its values.

To call your custom Java code during upload, you must expose your custom Java code through a `pageDef` action binding and then configure the ADF Table component's `UpdateRowActionID` or `InsertAfterRowActionID` to point to the `pageDef` action binding.

H.2 About ADF Desktop Integration Model API

While data is being uploaded, if a tree node of the ADF Table component contains no actual rows, the ADF Desktop Integration remote servlet creates a temporary row object to store the attribute values. If you want to access the temporary row object and its attribute values, you must write custom Java code that uses the ADF Desktop Integration Model API library.

For more information about the classes and methods available in the API, see Section H.3, "ADF Desktop Integration Model API Classes and Methods."

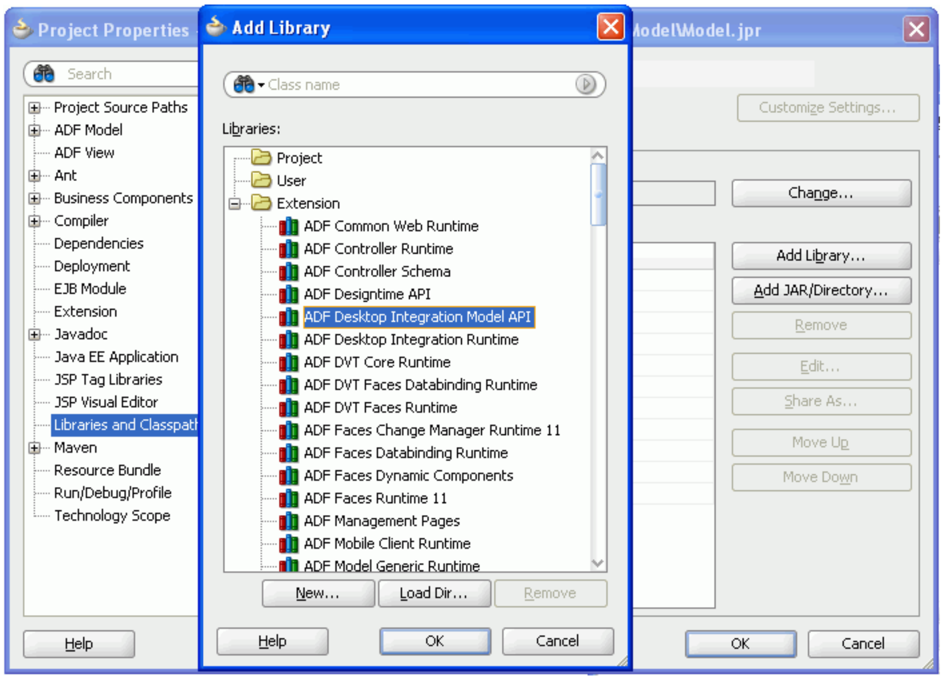
H.2.1 How to Add ADF Desktop Integration Model API Library to Your JDeveloper Project

You typically add the ADF Desktop Integration Model API Library to your application's data model project. The library is an independent library, not included with any technology scope. You can add it through Project Properties dialog box.

To add ADF Desktop Integration Model API library to your project:

1. In the Application Navigator, right-click the data model project and choose **Project Properties**.
2. In the Project Properties dialog, select **Libraries and Classpath** to view the list of libraries available.
3. Click **Add Library** and in the Add Library dialog box, select the **ADF Desktop Integration Model API** library.

Figure H–1 Add Library Dialog



4. Click **OK**. The library name adds to the **Classpath Entries** list.
5. Click **OK** to close the Project Properties dialog box.

H.3 ADF Desktop Integration Model API Classes and Methods

The ADF Desktop Integration Model API library contains one public class that contains APIs for retrieving temporary row objects.

H.3.1 The oracle.adf.desktopintegration.model.ModelHelper Class

The `ModelHelper` class is a public class that exposes Model APIs. The following sections describe the methods available in the class.

H.3.1.1 The getAdfdiTempChildRow Method

The method is used to lookup temporary child row object (`ViewRowImpl` object) associated with a particular master row. When required, the servlet code creates the temporary `ViewRowImpl` object and stores attribute values when there are no actual `ViewRowImpl` objects available.

The method returns the temporary child `ViewRowImpl` object containing any attribute values sent from worksheet.

Method Syntax

```
public static final ViewRowImpl getAdfdiTempChildRow(ViewRowImpl masterRow,
java.lang.String childAccessor)
```

Parameters

- `masterRow` – master row object
- `childAccessor` – child attribute name

H.3.1.2 The getAdfdiTempRowForView Method

The method is used to lookup temporary child row object (`ViewRowImpl` object) associated with a particular view. When required, the servlet code creates the temporary `ViewRowImpl` object and stores attribute values when there are no actual `ViewRowImpl` objects available.

The method returns the temporary child `ViewRowImpl` object containing any attribute values sent from worksheet.

Method Syntax

```
public static final ViewRowImpl getAdfdiTempRowForView(ApplicationModuleImpl am,
java.lang.String viewDefName)
```

Parameters

- `am` – application module instance
- `viewDefName` – view definition name

H.3.1.3 The getChildViewDef Method

The method is used to lookup polymorphic child view definition if the view link destination attributes specify one or more child discriminator attributes. The master row source attributes lookup the correct polymorphic child view definition through `ViewObjectImpl.findViewDefFromDiscrValues` API. If no child discriminator attributes are defined, or the child view is non-polymorphic, the default child `ViewDefImpl` object is returned.

The method returns the temporary child `ViewRowImpl` object containing any attribute values sent from worksheet, or returns null if the object is not found.

Method Syntax

```
public static final ViewDefImpl getChildViewDef(ViewRowImpl masterRow,
java.lang.String childAccessor)
```

Parameters

- `masterRow` – master row object
- `childAccessor` – child attribute name

I End User Actions

This appendix describes the actions your end user would be performing while using your application and integrated Excel workbook.

The actions described in this appendix assume that you have developed a functioning Fusion web application similar to Master Price List module. However, your application might not support all actions provided by Master Price List module.

This appendix includes the following sections:

- Section I.1, "Installing the Runtime Edition of ADF Desktop Integration"
- Section I.2, "Importing Data from a Non-Integrated Excel Worksheet"
- Section I.3, "Removing Personal Information"
- Section I.4, "Changing an Integrated Excel Workbook at Runtime"
- Section I.5, "Limitations of Integrated Excel Workbook at Runtime"
- Section I.6, "Using An Integrated Excel Workbook"

I.1 Installing the Runtime Edition of ADF Desktop Integration

To enable end users to use ADF Desktop Integration and integrated Excel workbooks, you must install the Runtime edition of ADF Desktop Integration.

When you run the ADF Desktop Integration setup tool, it verifies whether required software is installed on the system. For more information about the required software, see the following:

- Section 3.2, "Required Oracle ADF Modules and Third-Party Software"
- Section 3.3, "Enabling Microsoft .NET Programmability Support"
- Section 3.4, "Allowing ADF Desktop Integration to Access Microsoft Excel"

Note: JDeveloper is not required to install the runtime edition of ADF Desktop Integration.

To install the Runtime edition of ADF Desktop Integration:

1. Navigate to the `*MW_HOME*\oracle_common\modules\oracle.adf.desktopintegration_11.1.1` directory, where `*MW_HOME*` is the Middleware Home directory.
2. Extract the contents of `adfdi-excel-runtime-client-installer.zip` to a temporary directory.

3. Run the `setup.exe` file located in the extracted directory of the `adfdi-excel-runtime-client-installer.zip` file.
4. Follow the instructions that appear in the dialog boxes launched by `setup.exe` to successfully install the required components.
5. If prompted, click **Yes** to restart the system and complete the setup of ADF Desktop Integration.

> **Note:** You cannot install the Runtime edition of ADF Desktop Integration from JDeveloper.

Note that you cannot install both the Designer and the Runtime editions of ADF Desktop Integration on a system. You must uninstall one before installing the other edition.

I.2 Importing Data from a Non-Integrated Excel Worksheet

End users who use the ADF Table component in an integrated Excel workbook to upload large batches of data rows to the Fusion web application can prepare these rows of data in a non-integrated Excel worksheet. They can then insert the data into the ADF Table component prior to invoking the ADF Table component's `Upload` action.

To prepare data in a non-integrated Excel workbook:

1. End users arrange the layout of data in a non-integrated Excel worksheet to match the layout of the ADF Table component in the integrated Excel workbook.

   For example, if an ADF Table component contains columns such as `Product`, `Price`, and `Description`, reproduce this layout in the non-integrated Excel worksheet.

   > **Tip:** Copy the column headers from the ADF Table component to the non-integrated Excel worksheet.

2. End users use functionality of Excel to import the rows of data into the non-integrated Excel worksheet in rows under the columns arranged in Step 1.
3. Row values that will be inserted into ADF Table component columns that use the TreeNodeList subcomponent must match a choice from the list of values.

   > **Tip:** Copy an ADF Table component row from the integrated Excel workbook to another worksheet of the same workbook, as the proper constraints will be defined for such a row and can be reproduced.

To insert data into the ADF Table component from a non-integrated Excel workbook:

1. In the ADF Table component, end users highlight *n* existing downloaded rows or new rows at the end of the ADF Table component where *n* is the number of rows to insert.
2. End users right-click and choose **Insert** from the Excel context menu.
3. In the non-integrated Excel worksheet, end users select the cells that they want to insert into the rows of the ADF Table component created in Step 2.

WARNING: Select the cells in the non-integrated Excel worksheet and not the rows or columns.

4. In the Excel menu, choose **Home** > **Copy**.
5. In the ADF Table component, select the upper left corner cell of the rows inserted in Step 2.
6. In the Excel menu, choose **Home** > **Paste**. Ensure that you do not paste any value in the **Key** column.

Note: Integrated Excel worksheets that contain an ADF Table component hide column A.

7. End users can now invoke the ADF Table component's `Upload` action using whatever functionality you configured for them as described in Section 7.8, "Configuring an Oracle ADF Component to Upload Changes from an ADF Table Component."

I.3 Removing Personal Information

If the Fusion web application that you integrate an Excel workbook with uses a security mechanism, such as single sign-on, personally identifying information may be stored in cookies on the system where the end user accesses the integrated Excel workbook. End users can remove this information using Microsoft Internet Explorer. End users must log out and close all integrated Excel workbooks to invalidate all active cookie-based web sessions.

For information about removing personal information, see Microsoft Internet Explorer documentation.

I.4 Changing an Integrated Excel Workbook at Runtime

Once you publish and deploy a finalized integrated Excel workbook, as described in Chapter 14, "Deploying Your Integrated Excel Workbook." end users can make the following changes to a workbook at runtime:

- Delete a column from an ADF Table or ADF Read-only Table component
- Drag and drop cells to move ADF components other than an ADF Button component
- Insert new rows into an ADF Table component
- Change the order of columns in an ADF Table or ADF Read-only Table component
- Insert non-integrated columns between the columns of an ADF Table or ADF Read-only Table component

However, some changes to a workbook at runtime can corrupt the integration and are not supported. For example, you must not delete or move the first column of the ADF Table or ADF Read-only Table component at runtime. For more information about what changes are not allowed at runtime, see Section I.5, "Limitations of Integrated Excel Workbook at Runtime."

I.5 Limitations of Integrated Excel Workbook at Runtime

There are some known limitations on changing ADF Desktop Integration components at runtime.

- **Moving a column in an ADF Read-only Table component** – If the end user moves a column of an ADF Read-only Table component to be the leftmost column of the table, ADF Desktop Integration generates an exception when the end user tries to download data.

  To resolve the problem, the end user must close and reopen the workbook without saving changes.

- **Deleting an Integrated Excel Worksheet** – If the end user deletes an integrated Excel worksheet, ADF Desktop Integration generates an exception when the end user tries to save the integrated Excel workbook.

  To resolve the problem, the end user must close and reopen the workbook without saving changes.

Additional known limitations:

- Excel's conditional formatting of cells at runtime has no impact on the selected cells or on the integration of workbook.
- The ADF Button components are disabled when the end user zooms in or out on an integrated Excel worksheet. The ADF Button components are active at 100% zoom only.
- Excel's Protect Sheet feature is incompatible with integrated Excel workbooks.

I.6 Using An Integrated Excel Workbook

End users who are new to the ADF Desktop Integration technology and integrated Excel workbook must be made aware of the following common actions:

- To insert a row in an ADF Table component, insert a full row in the worksheet, and add data in all mandatory columns. Do not insert partial rows, as partial rows corrupts the ADF Table component.

  For more information, see Section 7.5, "Configuring an ADF Table Component to Insert Data."

- To delete a row from the web application, flag the row by double-clicking the respective cell of the Flagged column, and click the respective delete button. Clearing the cell values of a row, or removing the row from the worksheet, does not remove the row from the web application. Also, deleting the row from Excel does not delete the row from the web application.

  For more information about row flagging, see Section 7.10.2, "Row Flagging in an ADF Table Component."

- To download all rows after uploading the changed data, ensure that **Download all rows after successful upload** checkbox is selected in Upload Options dialog box.
- To sort table data based on a particular column, select any cell of the column or the column header. Click **Sort and Filter** on the Home tab, and choose your desired option.

  To sort table data based on multiple columns, do not select data rows or columns partially. Select any cell of the table, click **Sort and Filter** on the Home tab, and choose **Custom Sort** to specify desired columns. You will notice that all columns of the table are automatically selected. Before you proceed, ensure that the **Key** column is also selected. In the Sort dialog box, add the columns, their order preference, and then click **OK**. Ensure that the **My data has headers** checkbox is enabled before you click **OK**.

- Before uploading the changes, ensure that the Changed column of all modified rows is marked with an upward pointing triangle. A double-click on the upward pointing triangle character removes it, and the data of the relevant row is not uploaded.
- While uploading, if you want to have Excel retain the format of a numeric or date value in a cell formatted with a text style, add an apostrophe symbol (’) before entering the value. The apostrophe symbol acts as an escape character and is not displayed with the value.
- Do not delete, edit, or clear any cells in the **Key** column of the table. Any change to these values can lead to upload failures and even data corruption.
- Do not change Excel’s settings for Protect Sheet or Protect Workbook. These settings are available in the Changes group of Review tab.

Index