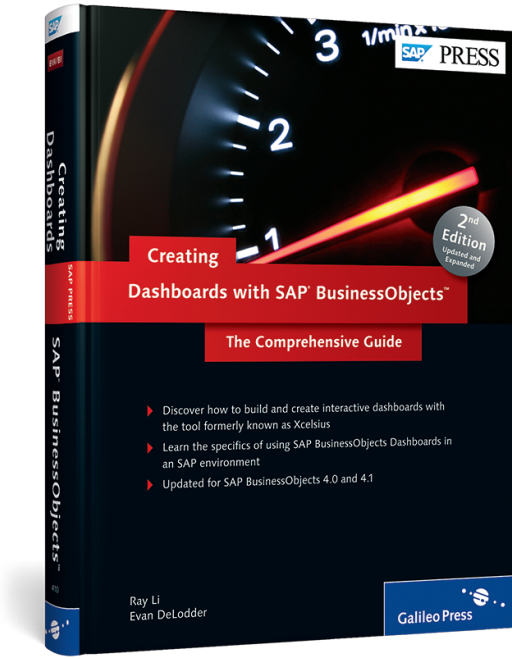
Ray Li and Evan DeLodder

Creating Dashboards with SAP® BusinessObjects™

The Comprehensive Guide



Galileo Press

Bonn • Boston

Contents at a Glance

1 **Introduction to SAP BusinessObjects Dashboards** 19

2 **Becoming Familiar with SAP BusinessObjects Dashboards** 35

3 **Dashboard Tutorial** 95

4 **UI Component Basics** 109

5 **Advanced UI Components** 221

6 **Data Connectivity Basics** 335

7 **Advanced Data Connectivity** 375

8 **Special Features** 441

9 **A Comprehensive Hands-On Example** 489

10 **Introduction to the SAP BusinessObjects Dashboards SDK** 511

11 **Get Started with Custom Component Basics** 527

12 **Implement Advanced Custom Add-On Component Features** 557

13 **Hands-On: Develop Your Custom Add-On Component** 581

A **Location Intelligence** 611

B **Tips for Using SAP BusinessObjects Dashboards** 635

C **The Authors** 669

Contents

Foreword 17

1 Introduction to SAP BusinessObjects Dashboards 19

1.1 What Is SAP BusinessObjects Dashboards? 19
1.1.1 Who Works with SAP BusinessObjects Dashboards? 19
1.1.2 Installation 20
1.1.3 Relationship with Excel 21
1.2 What Can SAP BusinessObjects Dashboards Do? 22
1.2.1 Data Visualization Capabilities 23
1.2.2 Data Connectivity Capabilities 27
1.2.3 Distribution 28
1.2.4 Changes in SAP BusinessObjects Dashboards 4.0 29
1.2.5 Extensibility 33
1.3 SAP BusinessObjects Dashboards in the SAP BusinessObjects Portfolio 33
1.4 Summary 34

2 Becoming Familiar with SAP BusinessObjects Dashboards 35

2.1 Menu 36
2.1.1 File 36
2.1.2 SAP 58
2.1.3 Edit 62
2.1.4 View 63
2.1.5 Format 68
2.1.6 Data 71
2.1.7 Help 74
2.2 Toolbar 77
2.2.1 Standard 78
2.2.2 Export 78
2.2.3 Themes 79
2.2.4 Format 79
2.2.5 Start Page 80
2.2.6 Summary 80

2.3 Components Browser 82
2.3.1 Category 82
2.3.2 Tree 83
2.3.3 List 84
2.4 Canvas 84
2.5 Embedded Excel Spreadsheet 85
2.6 Property Panel 85
2.7 Object Browser 88
2.8 Query Browser 89
2.8.1 Select a Universe 90
2.8.2 Build Query 90
2.8.3 Preview Query Result 92
2.8.4 Usage Options 92
2.9 Summary 93

3 Dashboard Tutorial 95

3.1 Introduction 95
3.2 Choose the Right UI Components 96
3.3 Bind Data 98
3.3.1 Bind Data for Pie Chart 99
3.3.2 Enable Drill-Down for the Pie Chart 100
3.3.3 Bind Data for Label 103
3.4 Connect to External Data 103
3.5 Formatting 104
3.6 Distribute the Output 106
3.7 Summary 108

4 UI Component Basics 109

4.1 Working with Charts 110
4.1.1 Pie Chart 110
4.1.2 Column Chart 133
4.1.3 Line Chart 155
4.1.4 Bar Chart 163
4.1.5 XY Chart 164
4.1.6 Bubble Chart 168
4.1.7 Area Chart 171

4.2 Selectors 174
4.2.1 Introduction to SAP BusinessObjects Dashboards Selectors 174
4.2.2 Select a Single Item 175
4.2.3 Filter 181
4.2.4 Checkbox 185
4.2.5 Ticker 186
4.2.6 Picture Menus 188
4.2.7 List Builder 192
4.3 Represent a Single Value 194
4.3.1 Introduction to Single-Value Components 195
4.3.2 Slider 195
4.3.3 Progress Bar 200
4.3.4 Dial and Gauge 200
4.4 Use Containers to Wrap Several Components 206
4.4.1 When to Use a Container 207
4.4.2 How to Use a Container 207
4.5 Build Backgrounds to Assist Layout 210
4.5.1 When to Use Backgrounds 211
4.5.2 How to Use Backgrounds 211
4.6 Universe Connectivity 214
4.6.1 Query Refresh Button 215
4.6.2 Query Prompt Selector 216
4.7 Summary 219

5 Advanced UI Components 221

5.1 Advanced Charts 221
5.1.1 Stacked Column Chart 222
5.1.2 Stacked Bar and Area Chart 227
5.1.3 Combination Chart 227
5.1.4 OHLC Chart 229
5.1.5 Candlestick Chart 239
5.1.6 Radar Chart 240
5.1.7 Filled Radar Chart 243
5.1.8 Tree Map 245
5.1.9 Sparkline Chart 247
5.1.10 Bullet Chart 251

5.2 Advanced Selectors 255
5.2.1 Accordion Menu 255
5.2.2 Icon 261
5.2.3 Play Selector 263
5.2.4 Calendar 270
5.3 Advanced Single-Value Components 273
5.3.1 Dual Slider 273
5.3.2 Spinner 275
5.3.3 Play Control 276
5.3.4 Value 278
5.4 Displaying Data in a Table 280
5.4.1 List View 280
5.4.2 Spreadsheet Table 285
5.4.3 Grid 289
5.5 Using Art 292
5.5.1 Image Component 292
5.5.2 Shapes 294
5.5.3 Lines 298
5.6 Use Maps for Geographical Representation 299
5.7 Web Connectivity 304
5.7.1 Connection Refresh Button 304
5.7.2 URL Button 307
5.7.3 Slide Show 312
5.7.4 SWF Loader 314
5.8 Others 315
5.8.1 Local Scenario Button 315
5.8.2 Trend Icon 317
5.8.3 Trend Analyzer 318
5.8.4 History 321
5.8.5 Print Button 324
5.8.6 Reset Button 324
5.8.7 Source Data 325
5.8.8 Panel Set 328
5.9 Summary 333

6 Data Connectivity Basics 335

6.1 Embedded Excel Spreadsheet 336
6.1.1 Role of Excel 337

6.1.2 How to Use Excel 337
6.2 Import Data from an Excel File 340
6.2.1 When to Import Data from an Excel File 340
6.2.2 How to Import Data from an Excel File 341
6.3 Security Issues Related to Accessing External Data 341
6.3.1 Run Locally 342
6.3.2 Run on a Web Server 343
6.4 XML Data 343
6.4.1 When to Use XML Data 346
6.4.2 How to Use XML Data 347
6.4.3 Practice 357
6.5 Web Service Connection 364
6.5.1 When to Use a Web Service Connection 364
6.5.2 How to Use a Web Service Connection 365
6.6 Excel XML Map 369
6.6.1 When to Use an Excel XML Map 370
6.6.2 How to Use an Excel XML Map 370
6.7 Summary 373

7 Advanced Data Connectivity 375

7.1 Query as a Web Service 376
7.1.1 When to Use Query as a Web Service 376
7.1.2 How to Use Query as a Web Service 377
7.2 SAP NetWeaver BW Connection 385
7.2.1 When to Use SAP NetWeaver BW Connection 386
7.2.2 How to Use SAP NetWeaver BW Connection 387
7.3 Live Office Connection 388
7.3.1 When to Use Live Office Connection 389
7.3.2 How to Insert SAP BusinessObjects Reports in Excel 391
7.3.3 How to Use Live Office Connection 391
7.3.4 Practice 397
7.4 Crystal Reports Data Consumer 399
7.4.1 When to Use the Crystal Reports Data Consumer Connection 401
7.4.2 How to Use the Crystal Reports Data Consumer Connection 401
7.4.3 Practice 408

7.5 Flash Variables 411
7.5.1 When to Use Flash Variables 412
7.5.2 How to Use Flash Variables 412
7.6 FS Command 417
7.6.1 When to Use FS Command 417
7.6.2 How to Use FS Command 418
7.6.3 Practice 421
7.7 External Interface Connection 424
7.7.1 When to Use an External Interface Connection 424
7.7.2 How to Use an External Interface Connection 425
7.7.3 Practice 427
7.8 LCDS Connection 431
7.8.1 When to Use an LCDS Connection 432
7.8.2 How to Use an LCDS Connection 433
7.9 Portal Data 435
7.9.1 When to Use Portal Data 436
7.9.2 How to Use Portal Data 436
7.10 Summary 440

8 Special Features 441

8.1 Drill-Down 441
8.1.1 When to Use Drill-Down 442
8.1.2 How to Use Drill-Down 443
8.1.3 Drill Down from One Chart to Another 444
8.1.4 Drill-Down on the Same Chart 447
8.2 Make Smart Use of Dynamic Visibility 452
8.2.1 When to Use Dynamic Visibility 453
8.2.2 How to Use Dynamic Visibility 456
8.2.3 Practice 458
8.3 Alerts 464
8.3.1 How to Use Alerts 464
8.3.2 Practice 465
8.4 Direct Data Binding 472
8.4.1 One-Dimensional Binding 473
8.4.2 Two-Dimensional Binding 474
8.5 Export 474
8.5.1 Flash 475
8.5.2 AIR 475

8.5.3 HTML 476
8.5.4 SAP BusinessObjects Platform 476
8.5.5 PDF 477
8.5.6 PowerPoint Slide 478
8.5.7 Outlook 479
8.5.8 Word 479
8.6 Themes and Colors 479
8.6.1 How to Apply a Theme 480
8.6.2 How to Apply a Color Scheme 482
8.6.3 How to Create a Customized Color Scheme 483
8.7 Summary 488

9 A Comprehensive Hands-On Example 489

9.1 Planning the Dashboard 491
9.1.1 Plan the Workflow 491
9.1.2 Plan the UI 492
9.2 Preparing Data 493
9.2.1 The US Map 494
9.2.2 The Gauge 495
9.2.3 The Column Chart 495
9.2.4 The Line Chart 497
9.2.5 The Radio Button 500
9.2.6 The Pie Chart 500
9.3 Organizing Data in Excel 501
9.4 Designing the Dashboard 505
9.4.1 Position the UI Components 505
9.4.2 Import the Excel File 506
9.4.3 Connect to External Data 508
9.4.4 Adjust the Appearance 509
9.5 Summary 510

10 Introduction to the SAP BusinessObjects Dashboards SDK 511

10.1 About the SAP BusinessObjects Dashboards SDK 511
10.2 About Flex 512
10.3 When to Use the SDK 513
10.4 How to Use the SDK 515
10.5 What Can I Do with the SDK? 518

10.5.1 Flex Applications 520
10.5.2 Data Processors, Connections, and Functions 521
10.6 SDK Best Practices 521
10.6.1 Use Only What You Need 521
10.6.2 Bindings 522
10.6.3 Use Custom Property Sheets 522
10.6.4 Don't Repeat Yourself 522
10.6.5 Develop Test Containers 522
10.6.6 Trace and Alert 523
10.6.7 Development Approaches (MXML versus ActionScript) 523
10.6.8 Styling 523
10.7 SDK Pitfalls 524
10.7.1 Flash Shared Local Objects are Unreliable 524
10.7.2 XLPs and XLXs Should Be Archived 524
10.7.3 Common Component Classes—First in Wins 525
10.8 Summary 525

11 Get Started with Custom Component Basics 527

11.1 Developing Basic Add-On Property Sheets 527
11.1.1 Property Sheet Data Binding 528
11.1.2 Explicitly Setting Property Values 531
11.1.3 Explicitly Getting Property Values 532
11.1.4 Property Sheet Styling 532
11.1.5 Basic Property Sheet Overview 532
11.1.6 Proxy.Bind Explained 542
11.2 Developing Basic Add-On Components 543
11.2.1 Main Component Initialization Event Handler and Import Statements 544
11.2.2 Private Variables 544
11.2.3 Public Chart Color Variable—xcChartColor 545
11.2.4 Public Chart Series Variable 545
11.2.5 Public Chart Data Variable 546
11.2.6 Chart Building Function 548
11.2.7 Tooltip Function 551
11.2.8 MXML Markup: Grid Lines and Cartesian Chart 552
11.3 Creating Basic Component Packages 552
11.3.1 Basic Component Packaging Steps 553

11.3.2 Packaging for Special Components 554
11.3.3 Packaging Best Practices 555
11.4 Summary 555

12 Implement Advanced Custom Add-On Component Features ... 557

12.1 Implementing Advanced Property Sheet Features 557
12.1.1 Subelement Binding 558
12.1.2 Persisting Property Sheet Values 560
12.1.3 Retrieving Persisted Property Sheet Values 561
12.1.4 Setting Custom Component Property Values 562
12.1.5 Retrieving Custom Component Property Values 562
12.1.6 Generating Reusable Property Sheet Patterns 563
12.1.7 Communicating with External Data Services 565
12.1.8 Implementing Advanced Component Features 566
12.1.9 Communicating at the Application Level 577
12.1.10 Additional Packaging Features 578
12.2 Where to Go from Here: Tips, Tricks, and Resources 579
12.3 Summary 579

13 Hands-On: Develop Your Custom Add-On Component 581

13.1 Creating the Chart 581
13.2 Creating the Flex Component and Property Sheet Project 582
13.2.1 Creating the Flex Property Sheet 588
13.2.2 Creating the Flex Component 594
13.3 Creating the Flex Test Container 597
13.4 Creating the Packager and SAP BusinessObjects Dashboards XLX Add-On 597
13.5 Creating the Data Sharing Component 598
13.5.1 Model Locator 599
13.5.2 Component Files 600
13.5.3 SAP BusinessObjects Dashboards Component Files 602
13.5.4 Property Sheet 603
13.6 Summary 608

Appendices 609

A Location Intelligence 611
A.1 What Makes Up Location Intelligence? 612
A.2 Why Location Intelligence Is Important 614
A.3 How Does Location Intelligence Fit into SAP BusinessObjects Dashboards? 615
A.4 Location Intelligence Options in SAP BusinessObjects Dashboards 617
A.5 Common Location Intelligence Use Cases 621
A.6 Location Intelligence Best Practices 626
A.7 Summary 633
B Tips for Using SAP BusinessObjects Dashboards 635
B.1 Using SAP BusinessObjects Dashboards in an SAP BusinessObjects Environment 635
B.2 Deployment and Migration 643
B.3 How to Use SAP BusinessObjects Dashboards with SAP NetWeaver BW and SAP NetWeaver Portal 644
B.4 Supported Excel Functions 647
B.5 SAP BusinessObjects Dashboards Editions 659
B.6 Tips for Creating a Good Dashboard 662
C The Authors 669

Index 671

This chapter provides a detailed walkthrough of a full property sheet and corresponding custom Flex component for SAP BusinessObjects Dashboards.

11 Get Started with Custom Component Basics

In this chapter we'll take a closer look into the core concepts of SAP BusinessObjects Dashboards SDK development. We'll cover the foundational pieces that compose a custom component and work with concrete examples to get you up to speed and prepared for more advanced topics.

Let's start by gaining an understanding of the backbone of a custom component integration that facilitates all component data flow and binding communication: the custom component property sheet.

11.1 Developing Basic Add-On Property Sheets

Conceptually, property sheets will seem very similar to a developer who is familiar with SAP BusinessObjects Dashboards and interacting with the SAP BusinessObjects Dashboards components offered in the default component library. A property sheet is essentially a self-contained Flex application that serves as both a user interface control and a communication proxy between the SAP BusinessObjects Dashboards design environment and your custom component's publicly exposed properties that you want to give users control over.

Property sheets can serve many functions, and since they're Flex-based applications, property sheets enable the custom component developer to exercise creativity in designing the most effective user interfaces that the business users who leverage the custom component will ultimately interact with.

The basic primary property sheet features we'll cover before moving to more advanced property sheet features are:

- Property data binding
- Property value setting/getting

11.1.1 Property Sheet Data Binding

Perhaps the most powerful feature exposed by the SAP BusinessObjects Dashboards SDK is the ability to bind public properties that you specify in your custom Flex component to a variety of data ranges in the underlying SAP BusinessObjects Dashboards Excel data model. Binding types come in a handful of shapes and sizes and enable custom components to read, write, or read and write to the SAP BusinessObjects Dashboards Excel model in every way that is needed to fundamentally communicate with the two-dimensional data model (rows and columns) that's supported by the SAP BusinessObjects Dashboards implementation of the Excel model.

Bindings can be adapted to suit a variety of basic to advanced use cases. In this section we'll review the basic concepts of data binding that a custom component developer is most likely to encounter. Below are the fundamental concepts offered by the SAP BusinessObjects Dashboards SDK binding mechanism.

Here's a binding call example that specifies a read-only two-dimensional array property binding of a custom component's xcChartData property:

```
proxy.bind("xcChartData", null, bindingID, BindingDirection.OUTPUT, "",
    OutputBindings.ARRAY2D);
```

Binding Directions and Data Flow

When establishing a binding between SAP BusinessObjects Dashboards and one of your custom component's properties, you can specify whether your property should be treated as read-only, write-only, or as both a read and write property.

Reading Values from the Spreadsheet

To specify your component's property as a read-only property, meaning that your property will be a consumer of data from the SAP BusinessObjects Dashboards model, you'll need to specify a binding direction type of `OUTPUT`, as described

below. By subscribing to the SAP BusinessObjects Dashboards model through binding as a read-only or `OUTPUT` consumer, your component's property value will change any time the range or cell data your custom component is bound to in the SAP BusinessObjects Dashboards data model changes, both at runtime as well as design time in the SAP BusinessObjects Dashboards environment.

To access this binding direction type, import the *xcelsius.binding.BindingDirection* namespace into your property sheet application. The fully qualified path to the `OUTPUT` type is:

```
xcelsius.binding.BindingDirection.OUTPUT
```

Note

Anything referenced in code samples needs to retain the Xcelsius name. Unfortunately this is confusing, but is the proper name until SAP renames the SDK's guts.

Writing Values to the Spreadsheet

To specify your component's property as a write-only property, meaning that your property will be a provider of data to the SAP BusinessObjects Dashboards model, you'll need to specify a binding direction type of `INPUT`, as described below. By subscribing to the SAP BusinessObjects Dashboards model through binding as a write-only or `INPUT` consumer, your component's property value will update the cell or range that it is bound to in the data model any time the data in your bound custom component property changes.

To access this binding direction type, import the *xcelsius.binding.BindingDirection* namespace. The fully qualified path to the `INPUT` type is:

```
xcelsius.binding.BindingDirection.INPUT
```

Writing Values to and Reading Values from the Spreadsheet

To specify your component's property as both a read and write property, meaning that your property will be a provider and consumer of data via the SAP BusinessObjects Dashboards Excel model, you'll need to specify a binding direction type of `BOTH`, as described below. By subscribing to the SAP BusinessObjects Dashboards model in this manner, your component's property value will update the cell or range that it is bound to in the data model any time the data in your

custom component property changes, and conversely, your component's property value will change anytime the SAP BusinessObjects Dashboards Excel model is updated.

To access this binding direction type, import the *xcelsius.binding.BindingDirection* namespace. The fully qualified path to the `BOTH` type is:

```
xcelsius.binding.BindingDirection.BOTH
```

Binding Types

Whether your custom component's property is read, write, or both, it will need to specify what type of value it is in the `proxy.bind()` operation so the SDK can properly handle its data flow.

The options for `InputBindings` and `OutputBindings` are shown in Figures 11.1 and 11.2.

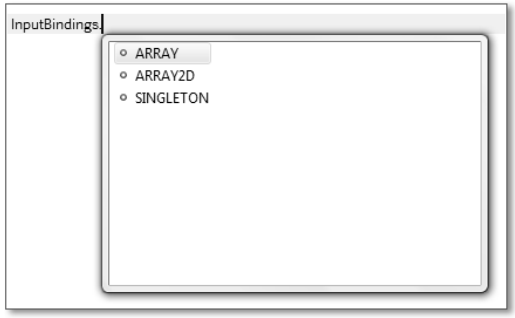


Figure 11.1 InputBinding Options

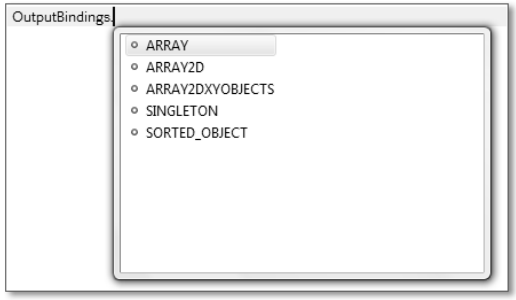


Figure 11.2 OutputBinding Options

InputBindings

- `ARRAY`
  A type of `ARRAY` indicates that your custom component's property will be bound to and able to write to a set of cells in either a single column or row.
- `ARRAY2D`
  A type of `ARRAY2D` indicates that your custom component's property will be bound to and able to write to a set of cells in a row and column format.
- `SINGLETON`
  A type of `SINGLETON` indicates that your custom component's property will be bound to and able to write to a single cell.

OutputBindings

- `ARRAY`
  A type of `ARRAY` indicates that your custom component's property will be bound to and able to read from a set of cells in either a single column or row.
- `ARRAY2D`
  A type of `ARRAY2D` indicates that your custom component's property will be bound to and able to read from a set of cells in a row and column format.
- `SINGLETON`
  A type of `SINGLETON` indicates that your custom component's property will be bound to and able to read from a single cell.
- `ARRAY2DXYOBJECTS`
  A type of `ARRAY2DXY` indicates that your custom component's property will be bound to and able to read multiple columns or rows of cells output as a two-dimensional array of objects with X and Y properties.
- `SORTED_OBJECT`
  A type of `SORTED_OBJECT` indicates that your custom component's property will be bound to and able to read a sorted object representation of a two-dimensional table.

11.1.2 Explicitly Setting Property Values

In addition to binding custom component properties to data in the SAP BusinessObjects Dashboards Excel model, the ability also exists to read from and write to a custom component's property directly through the property sheet, thus bypassing the Excel model altogether.

An example of this functionality would be if you have a Flex ColorPicker control on your property sheet that is responsible for setting a color property on your component. Assuming you had a custom Flex canvas component with a public property named `xcBackgroundColor`, you could write a value to that property by using the following command specified inline in the ColorPicker's `change` event handler, where `cp` represents the ID of your MXML ColorPicker in the Flex property sheet.

```
private var proxy:PropertySheetExternalProxy =
  new PropertySheetExternalProxy();

<mx:ColorPicker left="19" top="175" id="cp"
  change="proxy.setProperty('xcChartColor',cp.selectedColor)"/>
```

11.1.3 Explicitly Getting Property Values

The ability to extract values from your custom component at property sheet design time exists as well and will be covered in Chapter 13.

11.1.4 Property Sheet Styling

While certain factors need to be considered for styling custom components, property sheet styling is direct and enables developers to leverage common styling practices in Flex such as style sheet references, inline style directives, and embedded style sheets.

11.1.5 Basic Property Sheet Overview

With some basic property sheet concepts in mind, let's walk through a functional example that highlights some of the fundamental aforementioned features. The full property sheet code is listed in breakout form of the code and accompanying code explanations in order from top to bottom.

This property sheet is designed as a basic example that is used to control a basic Flex `AreaChart` custom component with a handful of public properties. The full sample source code for the corresponding custom component is listed in Section 11.2.

Figure 11.3 highlights the overall end result of the following property sheet and component walk through by displaying the component, connected to data, and the

data layout, inside the SAP BusinessObjects Dashboards designer. Figure 11.4 shows the isolated property sheet in SAP BusinessObjects Dashboards design mode.

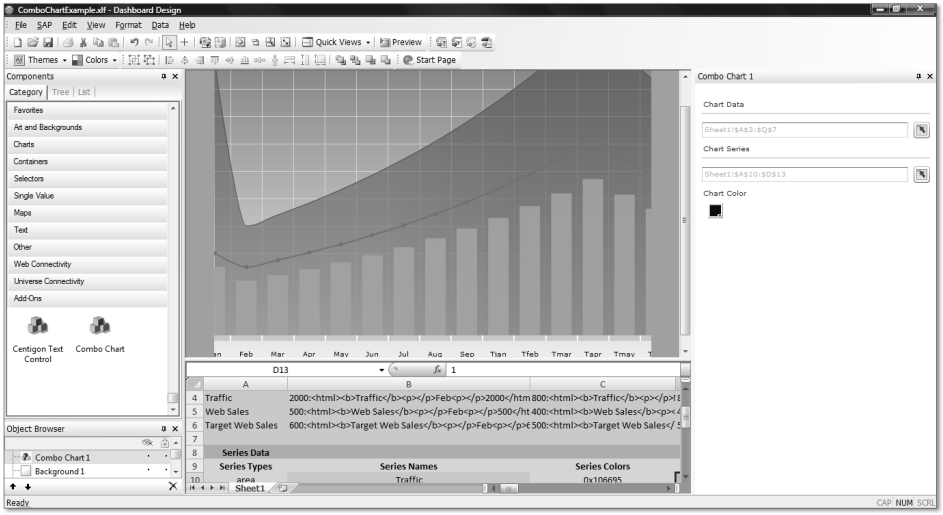


Figure 11.3 Custom Component and Property Sheet End Result

Figure 11.4 Property Sheet in Design Mode

Property Sheet Code Walkthrough

Let's take a walk through the property sheet code now, which has been broken up into the following sections.

- Main property sheet initialization event handler
- Fundamental SDK import statements
- Private SDK variables
- Supporting SDK functions
- `Proxy.Bind` dissected

Property Sheet Application Complete Event Listener

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="absolute" applicationComplete="init();">
```

On line 2 of the property sheet, which is a basic Flex Application MXML file, we listen for the native Flex Application event `applicationComplete`. When this event is triggered, it calls the `init()` function. In the `init()` function, we perform the routine task of initializing and executing the basic SAP BusinessObjects Dashboards SDK event listeners and callback functions needed to perform the custom component data binding and value setting tasks.

Fundamental SDK Import Statements

The import statements below provide us with a reference to the foundational classes and utilities in the SDK needed for the basic property sheet application.

```
<mx:Script>
<![CDATA[
import mx.containers.*;
import mx.controls.*;
import mx.core.Container;
import mx.events.FlexEvent;
import xcelsius.binding.BindingDirection;
import xcelsius.binding.tableMaps.input.InputBindings;
import xcelsius.binding.tableMaps.output.OutputBindings;
import xcelsius.propertySheets.impl.PropertySheetExternalProxy;
import xcelsius.propertySheets.interfaces.PropertySheetFunctionNamesSDK;
```

Here we call out and explain the import statements that are specifically related to the SDK.

```
import xcelsius.binding.BindingDirection;
```

The `BindingDirection` import statement enables you to call on binding direction constants from the SAP BusinessObjects Dashboards SDK. The binding direction options (`INPUT`, `OUTPUT`, `BOTH`) specify the direction of data flow between your SAP BusinessObjects Dashboards component properties and the SAP BusinessObjects Dashboards Excel data model when performing a bind operation through the SDK.

```
import xcelsius.binding.tableMaps.input.InputBindings;
```

The `InputBindings` import statement enables you to call on input binding constants from the SAP BusinessObjects Dashboards SDK. The input binding options (`ARRAY`, `ARRAY2D`, `SINGLETON`) specify the type of data range you want to bind to between your SAP BusinessObjects Dashboards component properties and the SAP BusinessObjects Dashboards Excel data model when performing a bind operation through the SDK. `InputBindings` should be specified when a custom component property needs to input values into the SAP BusinessObjects Dashboards Excel model.

```
import xcelsius.binding.tableMaps.output.OutputBindings;
```

The `OutputBindings` import statement enables you to call on output binding constants from the SAP BusinessObjects Dashboards SDK. The output binding options (`ARRAY`, `ARRAY2D`, `SINGLETON`) specify the type of data range you want to bind to between your SAP BusinessObjects Dashboards component properties and the SAP BusinessObjects Dashboards Excel data model when performing a bind operation through the SDK. `OutputBindings` should be specified when a custom component property needs to read values from the SAP BusinessObjects Dashboards Excel model.

```
import xcelsius.propertySheets.impl.PropertySheetExternalProxy;
```

The `PropertySheetExternalProxy` import statement provides access to the `PropertySheetExternalProxy` class, which is the backbone of the SDK integration performing all binding and value operations between the property sheet and the custom component.

```
import xcelsius.propertySheets.interfaces.PropertySheetFunctionNamesSDK;
```

The `PropertySheetFunctionNamesSDK` import statement provides access to a set of constants that tie to SAP BusinessObjects Dashboards SDK function names.

Basic Private SDK Variables

The three private variables defined in this section are key to managing custom component property data binding and property value setting through the SDK.

```
private var proxy:PropertySheetExternalProxy =
  new PropertySheetExternalProxy();
private var propertyToBind:String;
private var currentBindingID:String;
```

Let's discuss the primary private variables related to the SDK.

```
private var proxy:PropertySheetExternalProxy
```

The `proxy` variable is responsible for performing the job that its class name indicates: to serve as a proxy between the custom component's properties and the property sheet. This variable is responsible for all key communication, property data binding, and value setting in a custom component property sheet.

```
private var propertyToBind:String;
```

The `propertyToBind` variable serves as a string value that is responsible for holding the name of the custom component property that is currently being bound to during user-initiated binding operations. This value allows the developer to manage UI controls, data binding for the given property, and any other housekeeping or special functionality associated with the property. This variable is used primarily in the `initiateBind` and `continueBind` functions listed in the following section, "Supporting SDK Functions."

```
private var currentBindingID:String;
```

The `currentBindingID` variable serves as a string value that is responsible for holding the binding ID of the custom component property that is currently being bound to during binding operations. This value allows the proxy to manage data bindings for the given property. This variable is used primarily in the `initiateBind` and `continueBind` functions listed in the following section.

Supporting SDK Functions

A few common functions can be found across the majority of property sheet implementations. Those common functions are also included in this source code walkthrough as follows.

Initialize Values

The `initValues` function is responsible for retrieving the custom component's property values at property sheet initialization time and populating the property sheet's user interface controls with their property values. In this implementation, we retrieve the three values exposed by the custom component and loop through them to initialize their corresponding property sheet controls. The function code is listed here:

```
// Initializes Property Sheet on load to show the
current Xcelsius custom component property/style value.
private function initValues():void
{
    //Process the array of values for the Xcelsius custom
    //component properties.
    var propertyValues:Array = proxy.getProperties(
    ["xcChartData", "xcChartSeries", "xcChartColor"]);

    var propertyValuesLength:int = (propertyValues != null ?
                                    propertyValues.length : 0);
    for (var i:int=0; i < propertyValuesLength; i++)
    {
        // Get the property name and value.
        var propertyObject:Object = propertyValues[i];
        var propertyName:String = propertyObject.name;
        var propertyValue:* = propertyObject.value;

        // Process the property by name, either show the
        //value or show the cell address if bound to the
        //Excel spreadsheet.
        var bindingText:String = "";
        switch (propertyName)
        {
            case "xcChartData":
                bindingText=
                getPropertyBindDisplayName(propertyName);
                if (bindingText != null)
                {
                    // When bound the user cannot edit the value.
                    tiChartData.enabled = false;
                    // Show the address we are bound to.
                    tiChartData.text = bindingText;
```

```
                }
                else
                {
                    tiChartData.text = "";
                }
                break;
            case "xcChartSeries":
                bindingText =
                getPropertyBindDisplayName(propertyName);
                if (bindingText != null)
                {
                    tiChartSeries.enabled = false;
                    tiChartSeries.text = bindingText;
                }
                else
                {
                    tiChartSeries.text = "";
                }
                break;
            case "xcChartColor":
                cpChartColor.selectedColor = propertyValue;
            break;
        default:
            break;
    }
  }
}
```

Get Bound Property Display Names

The `getPropertyBindDisplayName` function is a utility function that's responsible for returning the Excel range address that a given custom component property is bound to once a user-initiated binding operation has been completed. This cell address value is traditionally used in all custom SAP BusinessObjects Dashboards components to visually indicate to the user through a text input control where in the Excel model the property is bound. The function code is listed here:

```
// Returns the bind display name or null if not bound.
private function
    getPropertyBindDisplayName(propertyName:String):String
{
```

```
    // Get the array of bindings for this property.
    var propertyBindings:Array = proxy.getBindings([propertyName]);
    if ((propertyBindings != null)      &&
        (propertyBindings.length   > 0) &&
        (propertyBindings[0].length > 0))
    {
        // We have at least one binding for this
        //property so pick the 1st one.
        // Note: [0][0] is 1st property in the array,
        //then 1st binding for that property.
        var bindingID:String = propertyBindings[0][0];
        return proxy.getBindingDisplayName(bindingID);
    }
   return null;
}
```

Initiating End-User Interaction Binding Operations

The `initiateBind` function is a utility function that is responsible for launching the SAP BusinessObjects Dashboards binding window as shown in Figure 11.5 and Excel range address that a given custom component property is bound to when the binding button for a particular property is clicked. This function allows the user to select or modify the range that a custom component's property is bound to.

Figure 11.5 Initiate Bind Window

Listed here is the function code needed to launch this utility to initiate data binding operations for a given custom component property:

```
// Allows the user to select the Excel spreadsheet cell
//to bind to an Xcelsius custom component property.
private function initiateBind(propertyName:String):void
{
    //If there is an existing binding for this property
    //show that in the Excel binding selection window.
    //Store the currentBindingID (null if there is no
    //current binding), we need this for "continueBinding".
```

```
    currentBindingID = null;
    var propertyBindings:Array = proxy.getBindings([propertyName]);
    if ((propertyBindings != null) && (propertyBindings.length > 0))
    {
        //Use the 1st binding address for the property.
        currentBindingID = propertyBindings[0];
    }

    //Store the name of the property that we are binding,
    //we need this when we "continueBinding".
    propertyToBind = propertyName;

    //Let the user choose where to bind to in the
    //Excel spreadsheet.
    proxy.requestUserSelection(currentBindingID);
}
```

Finalizing User-Initiated Binding Operations

The `continueBind` function is called as a result of the user clicking the OK button in the SAP BusinessObjects Dashboards binding control shown in Figure 11.5 once he has selected the range he wants to bind the custom component property to. This function performs the final steps via the `proxy` variable needed to establish a binding of any type between the SAP BusinessObjects Dashboards Excel model and a custom component property. Listed here is the function code needed to finalize or commit a data binding operation for a given custom component property. The remainder of the MXML code for this source code example is listed below the `continueBind` function code as well.

```
// Completes the binding when the user has finished selecting
//the cell to bind to or clear the binding.
private function continueBind(bindingID:String):void
{
    // Define common variables here.
    var propertyName:String = propertyToBind;
    var propertyValues:Array;
    var propertyObject:Object;
    var bindingAddresses:Array;

    // Clear any existing bindings - so we can re-bind.
    if (currentBindingID != null)
    {
```

```
    proxy.unbind(currentBindingID);
    currentBindingID = null;
}

// Process the property binding.
switch (propertyName)
{
    case "xcChartData":
        //User explicitly cleared binding,
        //do not create another.
        if ((bindingID == null) || (bindingID == ""))
        {

            //Fill the chart with an empty dataset
            propertyValues = proxy.getProperties([propertyName]);
            propertyObject = propertyValues[0];
            //Make sure we set the property
            //on the component as well.
            proxy.setProperty(propertyName, propertyObject.value);
            return;
        }
        //Display the range address.
        tiChartData.text = proxy.getBindingDisplayName(bindingID);

        proxy.bind("xcChartData",
            null, bindingID,
            BindingDirection.OUTPUT, "",
            OutputBindings.ARRAY2D);
        break;
    case "xcChartSeries":
        //User explicitly cleared binding,
        //do not create another.
        if ((bindingID == null) || (bindingID == ""))
        {
            propertyValues = proxy.getProperties([propertyName]);
            propertyObject = propertyValues[0];
            proxy.setProperty(propertyName, propertyObject.value);
            return;
        }

        // Display the range address.
        tiChartSeries.text = proxy.getBindingDisplayName(bindingID);
```

```
            proxy.bind("xcChartSeries",
            null, bindingID,
            BindingDirection.OUTPUT, "",
            OutputBindings.ARRAY2D);
            break;
        default:
            break;
    }
}
]]>
</mx:Script>
<mx:Canvas minWidth="268"  minHeight="350"
           width="100%" height="100%"
           backgroundColor="#FFFFFF">
    <mx:Label x="10" y="22" text="Chart Data"/>
    <mx:HRule y="39" height="10" right="10" left="10"/>
    <mx:TextInput id="tiChartData"
        y="57" right="42" left="10"/>
    <mx:Button y="56" right="10"  width="24"
        click="initiateBind('xcChartData');"
        icon="@Embed('com/assets/bind to cell.png')"/>
    <mx:Label x="10" y="87" text="Chart Series"/>
    <mx:Label x="10" y="152" text="Chart Color"/>
    <mx:HRule y="104" height="10" right="10" left="10"/>
    <mx:TextInput id="tiChartSeries" y="122"
                  right="42" left="10"/>
    <mx:Button y="121" right="10"  width="24"
               click="initiateBind('xcChartSeries');"
               icon="@Embed('com/assets/bind to cell.png')"/>
    <mx:ColorPicker left="19" top="175"
                    id="cpChartColor"
                    change="proxy.setProperty('xcChartColor',
                    cpChartColor.selectedColor)"/>
</mx:Canvas>
</mx:Application>
```

11.1.6 Proxy.Bind Explained

The SAP BusinessObjects Dashboards SDK `bind` function currently accepts ten parameters, the last five of which are optional. Let's take a look at the fundamental first six parameters, in order.

```
proxy.bind(
    "xcChartData",           <- Property
    null,                    <- Chain
    bindingID,               <- BindingID
    BindingDirection.OUTPUT, <- Direction
    "",                      <- InputMap
    OutputBindings.ARRAY2D   <- OutputMap
);
```

- **Property**
  The name of the custom component property to be bound.
- **Chain**
  An alternate array argument usually left blank for normal binding scenarios. This chain is usually reserved for subelement binding.
- **BindingID**
  The SAP BusinessObjects Dashboards-generated `BindingID` of the custom component property to be bound.
- **Direction**
  The `BindingDirection` constant that indicates whether the property is read, write, or read and write.
- **InputMap**
  The `InputBindings` constant that indicates whether the property is:
  `SINGLETON`, `ARRAY`, or `ARRAY2D`
- **OutputMap**
  The `OutputBindings` constant that indicates whether the property is:
  `SINGLETON`, `ARRAY`, `ARRAY2D`, `ARRAY2DXYOBJECTS`, or `SORTED_OBJECT`

11.2 Developing Basic Add-On Components

Now that we have a basic property sheet implemented and connected to the properties exposed by the custom component, we'll take a look at how the custom component's public properties are exposed, consumed, and set, and how they process their data output flow from SAP BusinessObjects Dashboards, where applicable. The full component code is listed in breakout form of the code and accompanying code explanations in order from top to bottom.

This custom component is designed as a basic example that leverages a base Flex charting component, `CartesianChart`, to expose a simple yet powerful and highly adaptable chart that allows for multiple series types to be combined in a single chart implementation.

Let's walk through the different custom component codes.

11.2.1 Main Component Initialization Event Handler and Import Statements

On line 2 below, we watch for the native Flex canvas event of `creationComplete`. When this event is triggered, in the function `buildSeries()` we perform the fundamental operation of the component, which is to build up the chart's data and series. The complete import code is listed below.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Canvas verticalScrollPolicy="off" creationPolicy="all"
  creationComplete="buildSeries()"
  horizontalScrollPolicy="off"
  xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="400">
 <mx:Script>
 <![CDATA[
    import mx.charts.series.ColumnSet;
    import mx.graphics.RadialGradient;
    import mx.graphics.LinearGradient;
    import mx.managers.ToolTipManager;
    import mx.charts.renderers.LineRenderer;
    import mx.charts.renderers.CircleItemRenderer;
    import mx.charts.series.AreaSeries;
    import mx.charts.series.LineSeries;
    import mx.graphics.Stroke;
    import mx.charts.series.ColumnSeries;
    import mx.graphics.GradientEntry;
    import mx.collections.ArrayCollection;
```

11.2.2 Private Variables

The private variable `_columnSet` is responsible for clustering any specified `ColumnSeries` for the Cartesian chart. The private variable `_xcChartData` is a bindable `ArrayCollection` that serves as the Cartesian chart's data provider. In the MXML code listed last in this section in the subsection "MXML Markup: Grid

Lines and Cartesian Chart," the Cartesian chart's `dataProvider` property is bound to the `_xcChartData` variable. Listed below are the private variables for the basic component.

```
    private var _columnSet:ColumnSet = new ColumnSet();

[Bindable]private var _xcChartData:ArrayCollection = new ArrayCollection();
```

11.2.3 Public Chart Color Variable—xcChartColor

The public variable `xcChartColor` is set up as a read and write variable, and in each operation it is responsible for setting or getting the bindable internal variable `_xcChartColor`. In the MXML code listed last in this section in the subsection "MXML Markup: Grid Lines and Cartesian Chart," the Cartesian chart's color property is bound to this private variable that controls the color property of the chart. The public property function code is listed below.

```
[Bindable]private var _xcChartColor:Number = 0x000000;
public function get xcChartColor():Number
{
    return _xcChartColor;
}
public function set xcChartColor(value:Number):void
{
   _xcChartColor = value;;
}
```

11.2.4 Public Chart Series Variable

The public variable `xcChartSeries` is set up as a read and write variable, and in each operation it is responsible for setting or getting the internal variable `_xcChartSeries`. In the property sheet, this variable is defined as a two-dimensional array consumer, and each time the public variable is set, we cycle through this two-dimensional array variable to build the chart series for the Cartesian chart. Note that the function `buildSeries()` is called every time the `xcChartSeries` public set function is called, enabling the chart to completely redesign itself at SAP BusinessObjects Dashboards runtime any time the SAP BusinessObjects Dashboards dashboard designer decides it should be redesigned, making it highly adaptable. The function code is listed next:

```
private var _xcChartSeries:Array = new Array();
public function get xcChartSeries():Array
{
    return _xcChartSeries;
}

public function set xcChartSeries(value:Array):void
{
  try
  {
      _xcChartSeries = value;
      buildSeries();
   }
   catch(e:Error)
   {
      trace(e.getStackTrace());
  }
}
```

11.2.5 Public Chart Data Variable

The public variable `xcChartData` is set up as a read and write variable, and in each operation it is responsible for setting or getting the internal variable `_xcChartData`. In the property sheet, we have this variable defined as a two-dimensional array consumer, and each time the public variable is set, we cycle through this two-dimensional array variable to build the data for the Cartesian chart. Note that the building of the data is called every time the `xcChartData` public set function is called. Notice that we treat a two-dimensional array of data output from SAP BusinessObjects Dashboards exactly like any other two-dimensional array would be treated in ActionScript. We access the data in SAP BusinessObjects Dashboards by looping over the data, referencing rows and individual cells by using their ordinal position; that is, `_xcChartData[rowPosition][columnPosition]` = a single cell in the SAP BusinessObjects Dashboards Excel data model.

In this case, we always extract the user-specified series name for a given set of data, at position 0 on each row in the data. For brevity's sake, we then dynamically roll through each cell in any given row from position 1–*N*, to extract data values and associated tooltips. The data and tooltip values are specified as colon-separated in each cell in the SAP BusinessObjects Dashboards Excel model, that is, [data:tooltip] or 5,000:My Tooltip. The function code is listed next:

```
public function get xcChartData():Array
{
   return [];
}
/**
*Cycle through the 2D data Array,
*building data for each series
***/
public function set xcChartData(value:Array):void
{
   //Row[0]=SeriesName
   //Row[0-N]=Data:HTMLTooltip
   _xcChartData.removeAll();
   try
   {
        var columnStart:int = 0;
        var columnEnd:int = value[0].length;
        //Append each dataset to a
        //given month (month = key to each row)
        for (var i:int=columnStart; i<columnEnd; i++)
        {
            var data:Object = new Object();
            data.yLabel = value[0][i];
            if(data.yLabel == "" || data.yLabel == null)
                continue;
            //Build data for each month
            for (var i2:int = 1;i2<value.length;i2++)
            {
                //cell content format =
                //[dataValue : toolTip];
                var cell:String = String(value[i2][i]);
                var seriesName:String = value[i2][0];
                var dataValue:Number =
                    Number(cell.substring(
                            0,cell.indexOf(":")));
                var toolTip:String =
                    cell.substring(
                    cell.indexOf(":") + 1, cell.length);

                data[seriesName]= dataValue;
                data[seriesName + data.yLabel] = toolTip;
            }
```

```
            _xcChartData.addItem(data);
        }
    }
    catch(e:Error)
    {
        trace(e.getStackTrace());
    }
}
```

11.2.6 Chart Building Function

The `buildSeries` function is responsible for looping through the publicly set private variable, `xcChartSeries`. In the property sheet, this variable is defined as a two-dimensional array consumer, and each time the public variable is set, we cycle through this two-dimensional array variable to build the series for the Cartesian chart.

Notice that we treat a two-dimensional array of data output from SAP BusinessObjects Dashboards exactly like any other two-dimensional array would be treated in ActionScript. We access the data in SAP BusinessObjects Dashboards by looping over the data, referencing rows and individual cells by using their ordinal position; that is, _ `xcChartSeries[rowPosition][columnPosition]` = a single cell in the SAP BusinessObjects Dashboards Excel data model.

For readability's sake, we have created an intermediary variable, called `series` that is an array. Essentially, the series variable represents a row of data from the SAP BusinessObjects Dashboards model, and each cell in the row provides us with a singular piece of information about the type of series the user wants to see on the chart and how he wants that series to be named and represented through the Cartesian chart. In this case, we require the user to place the series type in position 0, the series name in position 1, the series color in position 2, and the series alpha in position 3 as shown below.

```
var series:Array = _xcChartSeries[i]; RA row of data from Xcelsius
var seriesType:String = series[0];    RA cell (#1) of data from Xcelsius
var seriesName:String = series[1];    RA cell (#2) of data from Xcelsius
var seriesColor:Number = series[2];   RA cell (#3) of data from Xcelsius
var seriesAlpha:Number = series[3];   RA cell (#4) of data from Xcelsius
```

As we cycle through this data, in the switch statement below, we determine the type of series specified, style the series accordingly, and add the series to the Cartesian chart. The function code is listed here:

```
/***
*Cycle through the 2D series Array,
*styling and adding specified series types
***/
private function buildSeries():void
{
    if(chart == null)
        return;
     chart.series=new Array();
     columnSet = new ColumnSet();
     columnSet.type = "clustered";
     var st:Stroke;
     for(var i:int=0;i<_xcChartSeries.length;i++)
     {
        var series:Array = _xcChartSeries[i];
        var seriesType:String = series[0];
        var seriesName:String = series[1];
        var seriesColor:Number = series[2];
        var seriesAlpha:Number = series[3];
        //Add the correct series based on the
        //specified seriesType
        switch (seriesType.toLowerCase())
        {
            case "line":
                var ls:LineSeries = new LineSeries();
                ls.yField = seriesName;
               ls.name = seriesName;
               ls.displayName= seriesName;

               st = new Stroke();
               st.color=seriesColor;
               st.alpha=seriesAlpha;
               st.weight=2;
               st.pixelHinting=false;

                var linePointFill:RadialGradient =
                    new RadialGradient();
                linePointFill.entries =
```

```
            [new GradientEntry(seriesColor,
                               0.33, .5),
            new GradientEntry(seriesColor,
                               0.65, seriesAlpha)]

      ls.setStyle("fill",linePointFill);
      ls.setStyle("lineStroke",st);
      ls.setStyle("stroke",st);
      ls.setStyle("radius",3);
      ls.setStyle("form","curve");
      ls.setStyle("itemRenderer",
            new ClassFactory(
            mx.charts.renderers.CircleItemRenderer));
      ls.setStyle ("lineSegmentRenderer",
            new ClassFactory(
            mx.charts.renderers.LineRenderer));

      //Add the series
      chart.series.push(ls);
   break;
   case "column":
       var cs:ColumnSeries = new ColumnSeries();
       cs.yField = seriesName;
      cs.name = seriesName;
      cs.displayName= seriesName;

       var columnFill:LinearGradient =
            new LinearGradient();
       columnFill.entries =
            [new GradientEntry(seriesColor,
                        0.33, seriesAlpha),
       new GradientEntry(seriesColor,
                        0.65, seriesAlpha)]

      cs.setStyle("fill",columnFill);
      cs.alpha=seriesAlpha;

      //Add the series
      columnSet.series.push(cs);
   break;
       case "area":
       var ar:AreaSeries = new AreaSeries();
```

```
                ar.yField = seriesName;
                ar.name = seriesName;
                ar.displayName= seriesName;

                st = new Stroke();
                st.color=seriesColor;
                st.alpha=1;
                st.weight=1;
                st.pixelHinting=false;

                var areaFill:LinearGradient =
                    new LinearGradient();
                areaFill.entries =
                    [new GradientEntry(seriesColor,
                        0.33, .5),
                    new GradientEntry(seriesColor,
                        0.65, .6)]
                ar.setStyle("areaFill",areaFill);
                ar.setStyle("areaStroke",st);
                ar.setStyle("stroke",st);
                ar.setStyle("form","curve");
                //Add the series
                chart.series.push(ar);
            break;
        }
        chart.series.push(columnSet);
    }
}

private function formatDataTip(obj:Object):String
{
    return obj.item[obj.element.displayName +
        obj.item.yLabel];
}
```

11.2.7 Tooltip Function

The ability to finely control the information that end users see when they mouse over a data point in an SAP BusinessObjects Dashboards chart is an extremely important feature. By leveraging some standard Flex capabilities, you can return custom tooltips to your end users via this simple function that extracts informa-

tion from the `obj:Object` parameter and returns your user-specified tooltip content, which may contain normal text as well as HTML. The function code is listed below.

```
private function formatDataTip(obj:Object):String
{
   return obj.item[obj.element.displayName + obj.item.yLabel];
}
```

11.2.8 MXML Markup: Grid Lines and Cartesian Chart

The MXML specified in this part of the component provides us with the type of grid lines we want to use for the chart as well as the basic implementation of the Cartesian chart's shell.

```
<mx:Array id="chartBg">
    <mx:GridLines alpha=".65" direction="both"/>
</mx:Array>
<mx:CartesianChart
        id="chart"
        width="100%" height="100%" showDataTips="true"
        color="{_xcChartColor}"
        dataTipFunction="formatDataTip"
        dataProvider="{_xcChartData}"
        backgroundElements="{chartBg}">
    <mx:horizontalAxis>
    <mx:CategoryAxis dataProvider="{_xcChartData}"
                     categoryField="yLabel"/>
        </mx:horizontalAxis>
    </mx:CartesianChart>
</mx:Canvas>
```

11.3 Creating Basic Component Packages

As a continuation of the custom chart component highlighted in this chapter, let's build a packager for it as the final step in preparing it for consumption by SAP BusinessObjects Dashboards.

11.3.1 Basic Component Packaging Steps

1. Open the *Packager.exe* application contained in the SAP BusinessObjects Dashboards SDK install directory *{Program Files}/SAP BusinessObjects\Xcelsius 4.0\SDK*.
2. Enter the details related to your component as shown back in Figure 11.5, including the component name.
3. Select the VISUAL COMPONENTS tab and click ADD NEW COMPONENT.
4. Enter the component's class name, including its full package path, as well as any additional information about the component, including the path to the property sheet SWF file, component SWF file, and optional images you may define to represent the component in the SAP BusinessObjects Dashboards Object Browser and component list as shown in Figure 11.6.

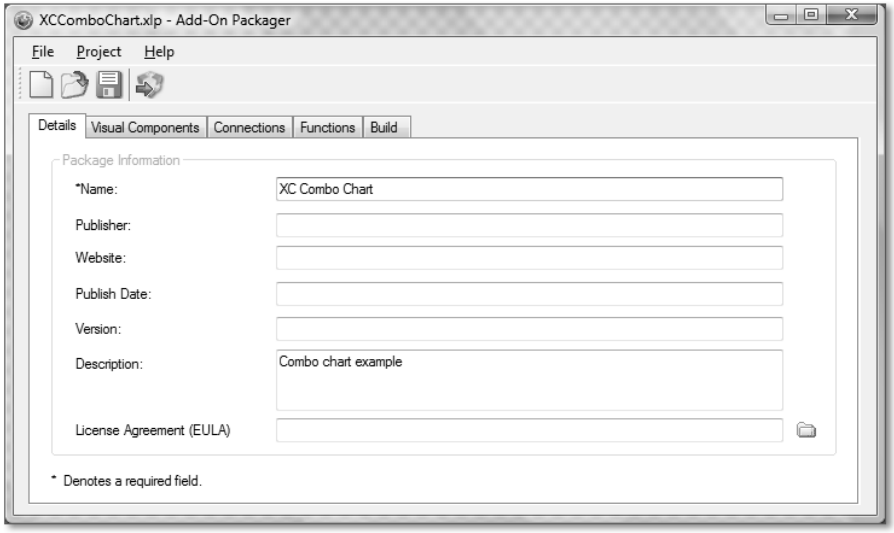


Figure 11.6 Enter your Component Class Name and SWF File Details

5. Save your packager file.
6. Build your XLX installer file and install it using the SAP BusinessObjects Dashboards Add-On Manager (in SAP BusinessObjects Dashboards, FILE • MANAGE ADD-ONS) as shown in Figure 11.7.

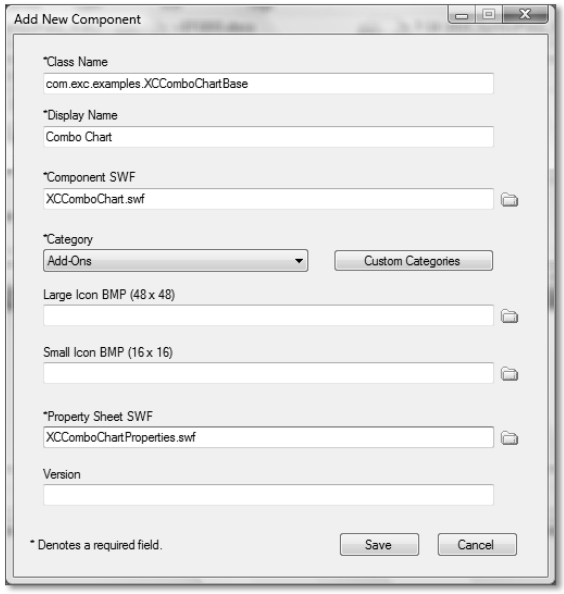


Figure 11.7 Build Your Component XLX Installer File

11.3.2 Packaging for Special Components

If your component isn't a visual component and is a data connection or a function component, use the CONNECTION and FUNCTION tabs accordingly to establish and build your component's installer file as shown in Figure 11.8.

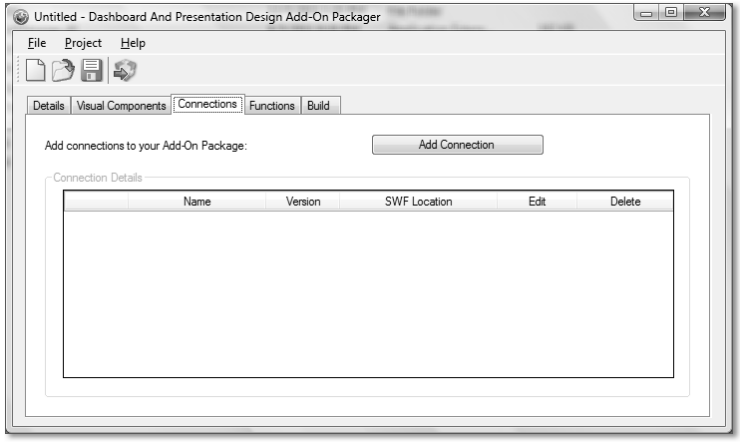


Figure 11.8 Connections Tab

11.3.3 Packaging Best Practices

While the packaging process is a simple act on the home stretch of creating your custom component, there are a handful of best practices you should adhere to during this very important process.

- Use relative paths when linking assets (SWF files, image files, text files) to your package.
- Use only one package per component to avoid installation conflicts in SAP BusinessObjects Dashboards. Each package contains a unique ID value that SAP BusinessObjects Dashboards inspects at install time and expects to be unique on a per-component basis.
- Don't copy components to avoid the conflict behavior described previously.
- Save and backup your packages, especially if you are developing products, because if a package is lost or corrupted, you'll once again run into the installation issue outlined above if you attempt to re-create the lost or corrupted package from the ground up.

11.4 Summary

In this chapter we went through a full property sheet and corresponding custom Flex component for SAP BusinessObjects Dashboards. We also dived deeper into the details of certain core pieces of SDK functionality, highlighted common functions where the SDK can be implemented, and gave an overview of how to develop a basic custom component for your SAP BusinessObjects Dashboards environment. With a functional component under our belts, we'll take a tour of some more advanced yet less advertised, very powerful concepts and utilities exposed by the SDK in Chapter 12.

Index

A

About Dashboard Design, 75
Accordion menu, 255
  example, 258
ActionScript 3, 512
ActionScript class, 563
ActionScript language, 417
Actual value, 263
Add-On Manager, 578, 598
Add-On Packager files, 524
Address validation, 612
Adjust appearance, 509
Adobe Acrobat, 477
Adobe Flash, 417, 475
Adobe Flash Player, 342, 411, 512
Adobe Flex technology, 511
Adobe Integrated Runtime (AIR), 475, 512
Adobe LifeCycle Data Service, 440
Advanced charts, 221
Advanced component features, 566
Advanced custom add-on component features, 557
Advanced data connectivity, 375
Advanced selectors, 255
Advanced single-value components, 273
Aggregation awareness technology, 496
Alert, 151, 158, 199, 464, 523
  method, 465
  thresholds, 153, 465
Alignment, 68, 506
AnyMap component, 300
Appearance, 122, 157
Application level, 577
Area chart, 171
Art, 292
Assist dashboard layout, 210
Auto drill-down, 451
Auto play, 265
Axes tab, 149

B

Background component, 318, 321
Backgrounds, 211
Bar chart, 111, 163
Basic add-on components, 532
Behavior, 119, 138, 157, 166
Benchmarks, 464
Best fit, 320
Best practices, 521
BI, 614
BI Launchpad, 53, 129, 476
  CMC, 636
  InfoView, 60
Bind, 383
  color, 132
  data, 98, 113
  data by range, 252
  display data, 281
Bindable chart data, 581
Bindable selected data, 582
Bindable selected tooltip, 582
Bindable series color, 582
Binding directions, 522
Binding directions and data flow, 528
Binding types, 530
BindingID, 543
Border, 123, 126
Bound property display names, 538
Bubble chart, 168
BubbleChartBase.mxml, 594
Budgets, 464
Build background, 210
Build number, 76
Build query, 90
Bullet chart, 251
  horizontal, 252
BW query, 645
By range, 134
By series, 135

C

Calendar, 270
limits, 272
Candlestick chart, 239
Canvas, 84, 467, 518
container, 210, 458
size, 505
size in pixels, 44
sizing, 66
Canvas and spreadsheet, 66
Cascading combo boxes, 271
Cascading Style Sheet (CSS), 523
Cell, 99
Census data, 621
Chain, 543
variable, 559
Chart area, 122
Chart background, 250
Charts, 361
Checkbox, 185
Child dashboard, 315
Child panels, 329
Color, 131
dynamic, 132
order, 155
picker, 126
Column chart, 111, 133, 258, 492, 495
Columns, 383
Combination chart, 227, 362
Combo boxes, 184
Comparative marker color, 254
Comparative value, 252
Comparison, 25
Complex formatting, 401
Component code, 574
Components browser, 36, 82
Connect to external data, 508
Connection refresh button, 305
Connections, 73
Consumer, 438
Container, 206, 457
Control sheet, 502
Credentials, 456
Cross-tab, 404
Cross-Tab Expert, 406
Crystal Reports Data Consumer connectivity, 400
CSV files, 514
Currency, 129
Custom add-on component, 581
Custom component, 516, 527
development, 557
property values, 562
public variable values, 589
Custom policy file, 380
Customized color scheme, 480, 483

D

Dashboard
data marts, 521
embed, 401
Dashboard Design Departmental edition, 477
Data, 133
binding, 65, 361
cleansing, 612
formatting, 612
insertion, 507
labels, 127, 226
mashups, 612
menu, 37, 71
meters, 519
part, 111
point, 126
processing, 401
sharing component, 598
source, 156
visualization capabilities, 23
visualizations, 518
Data connectivity, 344
basics, 335
capabilities, 27
Data Consumer connection, 401
Data Manager, 341, 347, 360, 377, 387, 402, 412
Data source write-back, 364
Data-change-tracking function, 322
Date/time, 131
Debugging, 522, 668
Default selection, 119
Delivery management, 624

Demographics, 612, 614
Description, 44
Device fonts, 45
Dial, 252
Dial and gauge, 200
Dialog box, 207
Direct data binding, 21, 135, 137, 472
Disable mouse input on load, 356
Disaster management, 623
Discrete, 164
Display several dashboards, 328
Distribute the output, 106
Distribution, 28, 29, 111
Divisions, 146
Document, 46
Document properties, 43
Drill, 384
Drill-down, 100, 102, 115, 441
Drill-down behavior, 137, 361
Dual slider, 273
Dummy data, 501
DuPont Financial analysis system, 317
Dynamic, 178
  data, 98
  visibility, 121, 293, 452, 456, 497

E

Edit menu, 62
Embedded Excel spreadsheet, 85, 336
Embedded files, 293
Embedded JPEG or SWF, 329
Embedded spreadsheet, 98, 346, 350, 383, 449
Enable data animation, 122
Enable data insertion, 115, 443
Enable load cursor, 356
Enable range slider, 140
Enable run-time tools, 139
Enable sorting, 120
Encrypted data repositories, 514
End-user interaction binding operations, 539
Enterprise data warehouse (EDW), 386
Entry effect, 122
Excel, 21, 501
  database and list management, 653
Excel, 21, 501 (Cont.)
  date and time, 657
  financial, 654
  formulas, 65
  functions, 183, 647
  information, 657
  Live Office, 391
  logical functions, 651
  lookup and reference, 653
  math and trigonometry, 648
  options, 49
  statistical functions, 651
  text and data, 656
Excel file, 98
  import, 506
Excel XML Map, 369
Explicitly getting property values, 532
Explicitly setting property values, 531
Exploded doughnut pie chart, 520
Exponential, 319
Export, 53, 72, 474
  preview, 53
  settings, 54
Extensible cell ranges, 664
External data, 103
External data services, 565, 568
External Interface connection, 424, 428

F

File, 36
Fill, 123, 126
Fill types, 296
Filter, 181
  data, 632
  rows, 177
Fisheye menu, 191
Fixed label size, 144
Flash, 56
Flash Data Expert, 406
Flash file, 342
  temporary, 479
Flash movies, 411
Flash Player, 367, 417
Flash trace log, 523
Flash Var, 272, 291, 639

Flash Variables, 407, 411
Flex, 512
  applications, 521
  binding, 515
  class library, 511
  framework, 512
  property sheet, 588
  test container, 597
Flex 2.0.1 Hotfix 3 SDK, 516
Flex Builder, 513, 582
Flex component, 582
  create, 594
Flex project
  create new, 583
Font, 44, 46
Format, 68, 79
  texts, 127
FS Command, 417
Function
  buildChart, 595
  continueBind, 565, 591
  formatDataTip, 596
  getPropertyBindDisplayName, 590
  HLookup(), 326
  init(), 588
  initiateBind, 564, 590
  initValues(), 589
  Rand(), 326
Fundamental SDK import statements, 534

G

Gannt charts, 514
Gauge, 252, 261, 492, 495
  component, 469
General tab, 113
Generic helper functions, 590
Geocoding, 612, 626
Geographical representation, 299
Geography, 614
GIS solutions, 620
Google maps, 618
Graphical background, 293
Greenwich Mean Time (GMT), 385
Grid, 47, 63, 289

H

Hierarchical pie charts, 519
History, 321
Hotfix 3 SDK, 585
HTML, 56
  container, 476
  file, 411
HTTPService, 576
  result, 575

I

Icon, 261
Ignore blank cells, 119, 138, 258
Image component, 292
Images through URLs, 189
Import, 71
  data from an Excel file, 340
  from platform, 72
Import Wizard, 643
Index, 115
Input component, 291, 600
Input parameters, 384
Input variables, 367
InputBindings, 531
InputMap, 543
Insertion, 114, 137, 157
Installation, 20
Insurance underwriting and risk management, 625
Integrated data warehouse (IDW), 386
Integrated development environment (IDE), 35
Interaction options, 118
Interactive analysis, 316
Interactive dashboard, 453
Interactivity, 26

J

Java, 346
Java Servlet, 358, 450
JavaScript code, 417, 428
JavaScript maps, 514

L

Label, 176, 261
Languages, 48
Launch, 59
Law enforcement, 626
Layout, 122
LCDS (Adobe LifeCycle Data Services)
  connections, 313, 431
Legacy bindings, 560
Legend, 125
License manager, 75
Line chart, 155, 362, 492, 497
Linear function, 319
Lines, 127, 298
List, 84
List builder, 192, 193
Live Office
  compatibility, 51, 392
  connection, 388, 490, 501
Load, 349
  status, 356
Local file, 107
Local scenario button, 315
Location intelligence (LI), 611
Logarithmic, 143, 319
Logo, 293
Logon panel, 456
Loosely couple data, 632
LOV, 381

M

Manage add-ons, 56
Manage operations, 625
Manual (Y) axis, 143
Map, 299, 492
Marker overlap, 225
MDX querying capabilities, 521
Measures, 164
Menu, 36
Metadata, 383, 458
Method
  GetReportBlock, 383
  runQueryAsAService, 383
Microsoft Office Excel, 336
Model locator, 599
Mouse click, 101, 118
Mouse events, 213
Mouse sensitivity, 280, 291
Mouse tracking, 203
Multi-dial gauges, 519
Multiple add-ons, 525
Multiple value axes, 240
Multi-select, 457
MXML, 512, 523
  application files, 602
  components, 563
  file, 588
  markup, grid lines and Cartesian chart, 552
My workspace, 66

N

New command, 36

O

Object Browser, 64, 88, 97, 207
OHLC chart, 229, 248
OHLC series, 233
Oil/land management, 626
One-dimensional binding, 473
One-dimensional cell range, 99
OpenDocument, 642, 643
Organizing data in Excel, 501
Outlook, 56, 479
Output component, 601
Output values, 383
OutputBindings, 531

P

Packager and SAP Dashboards XLX add-on, 597
Packaging best practices, 555
Panel container, 208, 462
Panel set, 328
PDF, 477
Percent, 130
Percentage, 24
Performance color, 254

Performance metric, 242
Persisting property sheet values, 560
  retrieve, 561
Picture menus, 188
Pie chart, 96, 99, 110, 492, 500
Plan UI, 492
Plan workflow, 491
Planning the dashboard, 491
Play control, 276
Play selector, 263
Plot area, 123, 250
Polynomial, 319
Portal data, 435
Position, 115
Power, 319
PowerPoint, 56, 478
PowerPoint/Word document, 342
Preferences, 46
Preferred viewing locale (PVL), 129
Preparing data, 493
Preview, 52, 53
Primary measure values, 252
Print button, 324
Private SDK variables, 533
Private variable, 559
Progress bar, 200, 252
Properties, 64
  panel, 112, 371
Property data binding, 528
Property sheet, 36, 85, 517, 527, 563, 603
  code base, 563
  controls, 589
  custom, 522
  data binding, 528
  initializes, 561
  styling, 532
Property value setting/setting, 528
Provider, 440
Proxy.Bind, 542
proxy.getPersist, 562
proxy.unbind, 560
Public component variables, 574
Publish, 58

Q

Query as a Web Service, 376, 403, 477, 638
  methods, 380
  workflow, 377
Query as a Web Service Designer, 383
Query Browser, 64, 89, 638
Query prompt selector, 216
Query refresh button, 215
Quick views, 64

R

Radar chart, 240
  filled, 243
Radio button, 500
Range list, 350
Range slider, 157
Ratio, 110
Region, 299
Region keys, 301
  default, 303
Relative size, 111
Reset button, 324
Resize, 339
Reusable data, 630
Reusable property sheet patterns, 563
Reverse geocoding, 612, 627
Rich Internet application, 431
Row/column, 117
Rows, 383
  maximum number of, 383
RSS feeds, 514
Run locally, 342
Run on a web server, 343
Runtime performance, 293
Runtime tools, 157

S

Samples, 40
SAP BEx query default values, 388
SAP Business Intelligence platform, 38, 107
SAP BusinessObjects, 19
SAP BusinessObjects BI launchpad, 343

SAP BusinessObjects BI platform, 636
export to, 476
SAP BusinessObjects Dashboards
best practices, 662
Departmental Edition, 38
deploy, 643
editions, 659
Enterprise, 660
Personal, 660
Present, 661
workspace, 35
SAP BusinessObjects Dashboards Excel model, 558
SAP BusinessObjects Dashboards Packager, 517
SAP BusinessObjects Dashboards software development toolkit (SDK), 511
SAP BusinessObjects Dashboards SWF Loader component, 577
SAP BusinessObjects Enterprise, 371, 477
SAP BusinessObjects Live Office, 33, 501, 638, 646
connections, 403
SAP BusinessObjects portfolio, 33
SAP BusinessObjects Universe, 638
SAP BusinessObjects Universe Designer, 33
SAP BusinessObjects Web Intelligence, 33, 442, 639
SAP BW query, 473
SAP Crystal Reports, 33, 388, 399, 639
insert SWF object, 404
native cross-tab, 406
SAP GUI, 386
SAP HANA, 386
SAP IK, 647
SAP menu, 58
SAP NetWeaver BW, 58, 107, 384, 644
connection, 385, 386
SAP NetWeaver Portal, 646
Scale, 142, 290
algorithm, 253
options, 140
values, 252
Scenario, 316
Scroll, 187
bars, 209
SDK, 585
Secondary axis, 133
Security, 367
issues related to accessing external data, 341
restriction, 389
Select a single item, 175
Select assets, 613
Selected item, 258
Selectors, 174
Series, 156, 157
Set appearance, 162
Set default currency format, 130
Shape files, 630
Shapes, 294
Shared local objects, 524
Single cell, 99
Single numeric value, 194
Singleton, 599
Skin, 479
Slide show, 312
Slide speed, 257
Slider, 195
Sliding picture menu, 191
Snapshot, 55
SOAP protocols, 639
Sound, 180
Source data, 325, 444
Space evenly, 68
Sparkline chart, 247
Spatial queries, 612
Spinner, 275
Spreadsheet, 99
table, 285
SQL (Structured Query Language), 494
Stacked area chart, 227
Stacked bar, 227
Stacked column chart, 222
visualized, 225
Standard, 78
Start page, 74
Status list, 116
Subelement
array tricks, 566
binding, 558, 566
properties, 558
SVG and vector maps, 618

SWC, 584
SWF document, 401
SWF file, 380, 577, 598, 616
SWF loader, 314
SWF movies, 312

T

Tab set, 209
Table, 280
  grid, 289
  list view, 280
  spreadsheet, 285
Targeted advertising, 624
Targets, 464
Templates, 39
Test container, 522, 597
Text formats, 150
Themes, 79
Themes and colors, 479
Third-party tools, 514
Three-dimensional analysis, 168
Thresholds, 464
Ticker, 186
Ticks, 266, 278
Title area, 124, 250
Titles, 133
Toolbar, 68, 77
  buttons, 80
Tooltip function, 551
Trace statements, 523
Track sales, 623
Transaction
  MDXTEST, 387
Translation Manager, 52
Translation settings, 52, 60
Transparency, 126, 263
  slider, 148
Tree map, 245
Trend analysis algorithm, 318
Trend analyzer, 318
Trend icon, 317
Trends, 497
Trigger connections, 305
Two-dimensional binding, 474
Two-dimensional cell range, 100

U

UI component, 68, 96, 258, 441, 455
  add, 505
  art, 292
  best practices, 665
  web connectivity, 304
UI control logic, 502
UI controls, charts and gauges, 22
UI elements, 22, 39
Undo/redo, 62
Universe connectivity, 214
Universe query, 89, 98, 116, 135, 157, 214, 281, 337, 472
  prompt, 264
  prompts, 117
URL, 344
  button, 307
  reporting, 642
Use current excel data, 55

V

Value, 116, 176
  change trend, 317
  component, 278
Variable values
  save across sessions, 560
Vertical axes, 240
View menu, 63

W

Web 2.0 maps, 618
Web connectivity, 304
Web Intelligence documents, 388
Web Intelligence Rich Client, 34, 90, 214, 376, 383, 640
Web Mapping Services (WMS), 629
Web service connection, 22, 364, 365
Web service URL, 379
What-if analysis, 325
Word, 479
Working with charts, 110
Wrap components, 207
Wrap several components, 206

X

X-axis, 133
X-axis scale by series, 254
.xlf, 38
XLF file, 560
.xls, 36
.xlsx, 36
XML data, 343, 344
XML data connectivity, 360, 450
XML feed, 568
XML file, 346
XML layout, 596
XML layout and controls, 593
XML schemas, 369
XY chart, 164

Y

Y-axis, 133, 142
Y-axis scale, 143
YTD trend, 504