Progress®



OpenEdge® Getting Started:
Introducing the Progress Developer Studio for OpenEdge Visual Designer

OpenEdge®

Copyright

© 2017 Progress Software Corporation and/or one of its subsidiaries or affiliates. All rights reserved.

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

Corticon, DataDirect (and design), DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Deliver More Than Expected, Icenium, Kendo UI, Making Software Work Together, NativeScript, OpenEdge, Powered by Progress, Progress, Progress Software Developers Network, Rollbase, SequeLink, Sitefinity (and Design), SpeedScript, Stylus Studio, TeamPulse, Telerik, Telerik (and Design), Test Studio, and WebSpeed are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries. Analytics360, AppServer, BusinessEdge, DataDirect Spy, SupportLink, DevCraft, Fiddler, JustCode, JustDecompile, JustMock, JustTrace, Kinvey, NativeScript Sidekick, OpenAccess, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro,Sitefinity, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, and WebClient are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

Please refer to the Release Notes applicable to the particular Progress product release for any third-party acknowledgments required to be provided in the documentation associated with the Progress product.

For the latest documentation updates see OpenEdge Product Documentation on Progress Communities: (https://community.progress.com/technicalusers/w/openedgegeneral/1329.openedge-product-documentation-overview.aspx).



October 2015

Last updated with new content: Release 11.6.0

Updated: 2019/03/01

Table of Contents

Preface....9

Purpose....9
Audience....10
Organization....10
Typographical conventions....10

Chapter 1: Features of the Visual Designer....13

What is the OpenEdge Visual Designer?....13
Components of the Visual Designer....14
The Design Canvas....14
The Control Toolbox....15
The Properties view....16
Other Progress Developer Studio for OpenEdge perspectives, editors, and views....17
The ABL Editor....17
The Class Browser....17
Debugger....18
DB Navigator....18
Meta Catalog....18
Tools for Business Logic....18
What you can create with the Visual Designer....19
The ABL Form....19
The ABL Dialog....19
The ABL MDI Form....19
The ABL User Control....19
The ABL Inherited Control....20
Getting help....20
Visual Designer online help....20
Help on .NET controls....20
OpenEdge .NET ABL documentation....20

Chapter 2: Creating the Customer Window....21

Overview....22
Tasks in this chapter....22
Before you begin 22
Adding customerForm.cls to the project....23
Changing the title bar of the form....25
Data binding....25
Adding the ProBindingSource control to a form....25

Defining the schema....27
Defining the recursive relationship....30
Attaching controls to a ProBindingSource....31
Attaching the ProBindingSource control to a data source....34
Adding interface methods....37
Adding event handlers....38
Running customerForm.cls....39

Chapter 3: Creating the Department Window....41
Overview....42
Tasks in this chapter....42
Before you begin 43
Adding departmentForm.cls to the project....43
Adding a ProBindingSource control to the form....43
Adding an UltraTree control to the form....44
Using the ColumnSet Layout Designer....45
Adding the code that completes data binding....48
Running departmentForm.cls....49

Chapter 4: Creating the Purchase Order Window....51
Overview....51
Tasks in this chapter....52
Before you begin 52
Adding purchaseOrderForm.cls to the project....52
Adding a ProBindingSource control....53
Adding a group box control....54
Adding editors....55
Adding a panel and buttons....59
Adding event handlers....61
Adding the interface methods....63
Adding the data binding code....64
Running purchaseOrderForm.cls....65

Chapter 5: Creating the Login Dialog....67
Overview....67
Tasks in this chapter....68
Before you begin 68
Adding LeftBar.cls to the project....68
Adding an image list to LeftBar.cls....70
Adding an UltraExplorerBar to LeftBar.cls....72
Subscribing to events and adding handlers....76
Adding LeftBar.cls to the Toolbox....77
Adding LoginBlock.cls to the Toolbox....79

Adding HelpButton.cls to the Toolbox....79
Adding a HelpButton to the Toolbox....80
Adding LoginDlg.cls to the project....80
Adding interface and login methods to LoginDlg.cls....82
Running LoginDlg.cls....85
Where to go from here....85

Chapter 6: Creating the Sports Window....87

Overview....87
Tasks in this chapter....88
Before you begin 88
Adding sportsForm.cls to the project....89
Adding an UltraExplorerBar to the form....90
Adding event handlers....91
Modifying the functionality of menu and toolbar items....92
Removing the New menu item....93
Modifying the Open menu item....94
Running sportsForm.cls....96

Index....97

Preface

For details, see the following topics:

- Purpose
- Audience
- Organization
- Typographical conventions

Purpose

The purpose of this manual is to provide an introduction to the functionality and features of the OpenEdge® Visual Designer. The OpenEdge Visual Designer is a plug-in for Progress Developer Studio for OpenEdge that allows you to build user interfaces for OpenEdge applications. You create user interfaces with OpenEdge Visual Designer by dragging and dropping .NET controls on a Design Canvas.

In addition to providing a general description of the Visual Designer, this manual includes a tutorial that describes how to use several of the .NET controls that are installed with the Visual Designer. You will see how to set some of the common properties associated with a control, how to associate a control with a data source, and how to implement event handling.

The tutorial is not meant as a guide for the application architecture or the coding practices that you would use to implement a real-world application. Rather, it is meant to show the power and flexibility of the Visual Designer to create a GUI using .NET objects to implement forms and controls.

Audience

The audience for this manual is an experienced OpenEdge ABL developer who is familiar with user-interface design and implementation.

Although the Visual Designer uses .NET objects to implement user-interface elements, you do not need to be familiar with .NET to understand this manual or to use the Visual Designer. The architecture of the Visual Designer allows you to treat .NET objects as if they were native ABL objects.

You should, however, have a general knowledge of object-oriented coding practices and concepts. In addition, you should be familiar with object-oriented programming in ABL, as described in *OpenEdge Development: Object-oriented Programming*.

Organization

- Features of the Visual Designer on page 13

  Provides an overview of the features and functionality of the OpenEdge Visual Designer.

- Setting Up the Sample Application Project

  Shows how to set up an Progress Developer Studio for OpenEdge project for building the sample application.

- Creating the Customer Window on page 21

  Shows how to use an ABL Form that features an UltraGrid control.

- Creating the Department Window on page 41

  Shows how to create an ABL Form that features an UltraTree control.

- Creating the Purchase Order Window on page 51

  Shows how to create an ABL Form that features UltraButtons and a number of editors.

- Creating the Sports Window on page 87

  Shows how to use an ABL MDI form as a container for the other windows in the sample application.

- Creating the Login Dialog on page 67

  Shows how to use an ABL Dialog form that features custom controls.

Typographical conventions

This documentation uses the following typographical and syntax conventions:

Convention	Description
Bold	Bold typeface indicates commands or characters the user types, provides emphasis, or the names of user interface elements.

Convention	Description
Italic	Italic typeface indicates the title of a document, or signifies new terms.
SMALL, BOLD CAPITAL LETTERS	Small, bold capital letters indicate OpenEdge key functions and generic keyboard keys; for example, **GET** and **CTRL**.
KEY1+KEY2	A plus sign between key names indicates a **simultaneous** key sequence: you press and hold down the first key while pressing the second key. For example, **CTRL+X**.
KEY1 KEY2	A space between key names indicates a **sequential** key sequence: you press and release the first key, then press another key. For example, **ESCAPE H**.
Syntax:	
`Fixed width`	A fixed-width font is used in syntax, code examples, system output, and file names.
`Fixed-width italics`	Fixed-width italics indicate variables in syntax.
`Fixed-width bold`	Fixed-width bold italic indicates variables in syntax with special emphasis.
`UPPERCASE fixed width`	ABL keywords in syntax and code examples are almost always shown in upper case. Although shown in uppercase, you can type ABL keywords in either uppercase or lowercase in a procedure or class.
Period (.) or colon (:)	All statements except `DO`, `FOR`, `FUNCTION`, `PROCEDURE`, and `REPEAT` end with a period. `DO`, `FOR`, `FUNCTION`, `PROCEDURE`, and `REPEAT` statements can end with either a period or a colon.
[ ]	Large brackets indicate the items within them are optional.
[ ]	Small brackets are part of ABL.
{ }	Large braces indicate the items within them are required. They are used to simplify complex syntax diagrams.
{ }	Small braces are part of ABL. For example, a called external procedure must use braces when referencing arguments passed by a calling procedure.
|	A vertical bar indicates a choice.
. . .	Ellipses indicate repetition: you can choose one or more of the preceding items.

1

Features of the Visual Designer

This chapter provides an overview of the Visual Designer.

For details, see the following topics:

- What is the OpenEdge Visual Designer?
- Components of the Visual Designer
- Other Progress Developer Studio for OpenEdge perspectives, editors, and views
- What you can create with the Visual Designer
- Getting help

What is the OpenEdge Visual Designer?

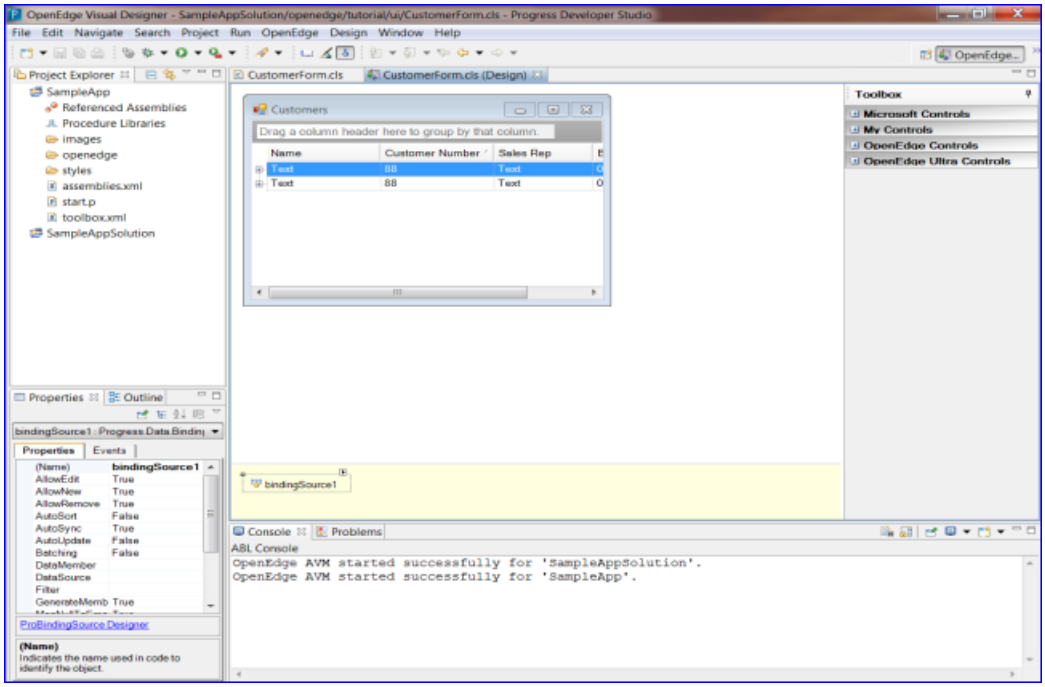
The OpenEdge Visual Designer is a plug-in for Progress Developer Studio for OpenEdge that allows you to build user interfaces for OpenEdge applications. These interfaces are written in ABL but are based on .NET objects, with the result that the GUI is richer, more robust, and more contemporary in look and feel compared to a traditional OpenEdge GUI. The style of user interface built in ABL with .NET forms and controls is referred to as the OpenEdge GUI for .NET.

Support for .NET objects is built into the OpenEdge ABL. ABL developers can use .NET objects without resorting to some other language (like C# or VB .NET). Because the OpenEdge GUI client hosts the Microsoft .NET runtime (CLR) within the ABL Virtual Machine (AVM), .NET controls appear to the developer as if they were native ABL controls. This powerful configuration not only allows .NET forms and controls in an OpenEdge application, but it also dispatches events from the .NET UI to event handling code written in the ABL. Additionally, OpenEdge data objects can easily be used as data sources for .NET controls.

The Visual Designer provides an integrated development environment with quick access to other Progress Developer Studio for OpenEdge tools such as DB Navigator, the ABL Editor, and Tools for Business Logic.

The OpenEdge Visual Designer perspective, shown in the following figure, displays the primary tools used for designing user interfaces. These are the Properties view, the Design Canvas, and the Control Toolbox, which are described in Components of the Visual Designer on page 14. In addition to the primary design tools, the Visual Designer Perspective includes a **Project Explorer** view that shows project resources, and an **Outline** view that helps you to navigate in code files.

Figure 1: The OpenEdge Visual Designer perspective



Components of the Visual Designer

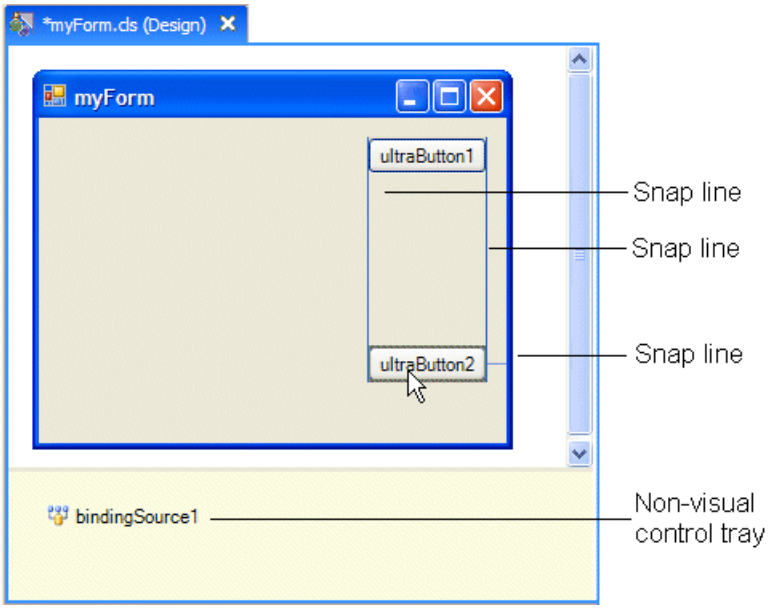
The components of the Visual Designer are the Design Canvas, the Toolbox, and the Properties view. The following sections describe these components.

The Design Canvas

The Design Canvas, shown in the following figure, is a drag-and-drop interface for developing graphical applications. It provides a visualization of how the form will appear as an application window at run time.

Typically, you start the Visual Designer by opening a file that implements one of the ABL form styles. (For more information, see What you can create with the Visual Designer on page 19.) Then, you position and resize various controls on the form. Using snap lines, you can easily align a control within the form, and with respect to other controls. You can also set a preference that displays a grid in the form.

Figure 2: The Design Canvas



There is a panel at the bottom of the Design Canvas where non-visual controls appear as labeled icons. Non-visual controls are not directly manipulated by the end user and have no graphical representation in the GUI at run time. An example of a non-visual control is the ProBindingSource (**bindingSource1** in the above figure), which links controls to data sources.

The Control Toolbox

The Control Toolbox contains a list of the controls that you use to develop a user interface. These controls are organized into three groups:

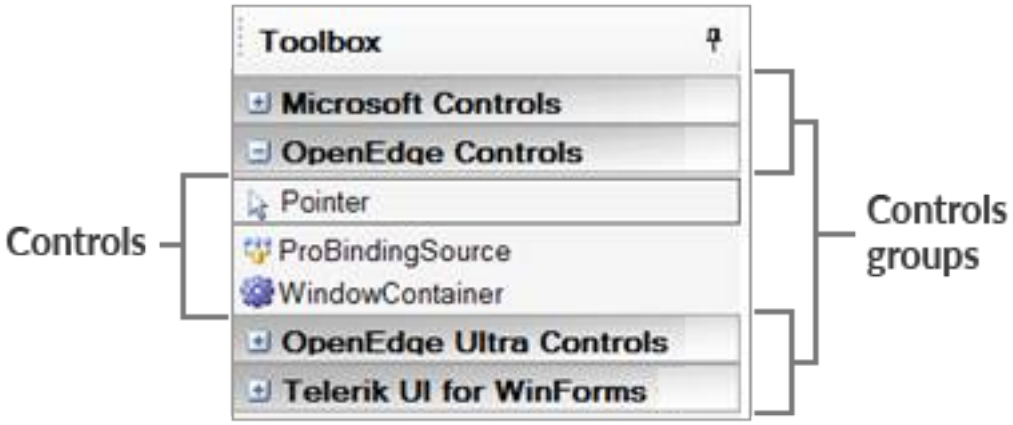
- **Microsoft** — The Microsoft® .NET™ UI Controls that are the basic user interface controls that install with Progress Developer Studio for OpenEdge
- **OpenEdge** — OpenEdge-specific controls for binding to OpenEdge data sources, and for embedding OpenEdge windows that install with Progress Developer Studio for OpenEdge
- **OpenEdge Ultra** — Advanced .NET controls based on Infragistics® NetAdvantage

  Note: OpenEdge Ultra requires the installation of OpenEdge® Ultra Controls for .NET, which is an optional product. OpenEdge Ultra is required if you want to complete the exercises in this manual.

- **Telerik UI for WinForms** — Telerik UI for WinForms controls that you can use to build your business applications. If you need a different version of Telerik, you can download it from the Telerik web site.

When you expand a group, as shown in the following figure, you can select a control, and then drag and drop it on a form in the Design Canvas.

Figure 3: The Control Toolbox

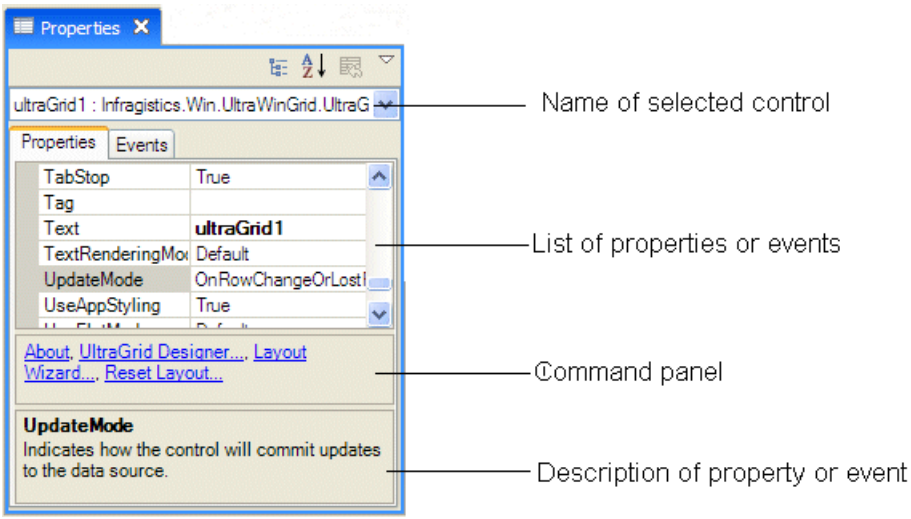


You can create custom controls by combining existing controls or by inheriting from an existing control class. You can also add additional controls that you obtain from third-party control vendors. In addition, you can organize controls into groups that you define, for example the **My Controls** group shown in the above figure.

The Properties view

The Visual Designer Properties view, shown in the following figure, has tabs where you can set properties or subscribe to events for controls and forms. At the top, it displays the control that is currently selected in the Design Canvas. You can edit the values of the properties and events in the list below the control name by selecting respective tabs.

Figure 4: The Properties view



A command panel provides access to the dialogs and wizards that are available for configuring the selected control. The bottom panel gives a brief description of the selected property or event.

Other Progress Developer Studio for OpenEdge perspectives, editors, and views

Since the Visual Designer is a plug-in to Progress Developer Studio for OpenEdge, you have easy access to the other features of the Progress Developer Studio for OpenEdge application development environment. These features are described in the sections that follow.

The ABL Editor

The ABL Editor is the tool you will use most often in conjunction with the Visual Designer. For example, you open the ABL Editor to add the methods that implement event handlers.

While developing a user interface, you might have a `.cls` file open in both the Visual Designer and the ABL Editor. Changes in one are automatically reflected in the other. For example, if you drop a new control on a form in the Visual Designer, the code for that control is automatically generated and appears in the ABL Editor.

The ABL Editor is a standard Eclipse text editor that is customized for working with ABL code. It supports many features to make coding easier, including:

- ABL statement completion
- Drag and drop functionality
- Wizards for creating new files and adding code blocks to files
- Templates (macros)
- Automatic code formatting
- Color Coding for Keywords

When you have a file open in the Visual Designer, right-click on the form and choose **View Source** (or press **F9**) to open the file in the ABL Editor.

The Class Browser

The Class Browser allows you to see the structure and content of the various items that you work with when creating ABL applications.

Whether you are developing with ABL or .NET classes, you can use the Class Browser's three-panel display to easily see details about all your ABL and .NET resources, packages or namespaces, and types. By default, the Class Browser displays this information for all resources, whether within a project or outside of one, in the workspace. Within a project, for r-code based on class files and Progress procedure libraries, the details the Class Browser displays are derived from the PROPATH associated with each open project in the workspace. (Directories and procedure libraries that do not have r-code classes are not displayed by the Class Browser.) For .NET classes, the details come from the assemblies that are associated with each project in the workspace. The same level of information can be displayed for external resources you add to the Class Browser.

The Class Browser is an Eclipse view. The easiest way to start it is to select it from the Fast View menu. The Fast View menu is available by clicking on , which is located at the bottom left of the Progress Developer Studio for OpenEdge Eclipse workbench.

Debugger

The Progress Developer Studio for OpenEdge Debugger provides a full set of tools for testing and debugging ABL programs. The Debugger works directly with the ABL Editor, making it easy to identify and fix problems as you work.

You can launch the Debugger (when debugging is enabled) by setting the focus in an ABL source file and clicking  on the Progress Developer Studio for OpenEdge toolbar.

DB Navigator

DB Navigator is a tool for browsing and maintaining the schema of any SQL-compliant database, including OpenEdge databases. You can view and analyze application data. You can create, modify, and delete tables, columns, indexes, sequences, and other database objects, but you cannot perform administrative tasks, like dumping and loading data.

DB Navigator is a perspective that you can launch from the **Open Perspective** menu. You can access the **Open Perspective** menu by clicking  on the top left of the Progress Developer Studio for OpenEdge workbench.

Meta Catalog

The Progress Developer Studio for OpenEdge Meta Catalog is an index that enables you to find where elements are used in your application. You can find where a temp table is defined and where it is used in your application. You can find all the procedures and functions in your application. You can find where those procedures and functions are called. You can also add your own annotations to the code and have them included in the index. You can use this index to simplify analyzing the impact of proposed changes and carrying out those changes.

You can access the Meta Catalog Explorer view from the Fast View menu. The Fast View menu is available by clicking on , which is located at the bottom left of the Progress Developer Studio for OpenEdge Eclipse workbench.

Tools for Business Logic

Progress Developer Studio for OpenEdge Tools for Business Logic allows you to build logical models that are coded as ABL data objects (temp tables and ProDataSets). Using a graphical editor called the Component Designer, you diagram components, adding elements such as tables, columns, indexes, and relationships. As you build your components, OpenEdge stores them either in a component model file or in a database.

After defining the logical structure of a component, you can generate the corresponding ABL source code. You can include this in an existing application, using the ABL Editor to modify it as necessary. You can also reverse this process, creating model components by extracting them from existing ABL source code.

Tools for Business Logic is a perspective that you can launch from the **Open Perspective** menu. You can access the **Open Perspective** menu by clicking  on the top left of the Progress Developer Studio for OpenEdge workbench.

What you can create with the Visual Designer

With the Visual Designer, you can build user interfaces that are based on three basic windows (ABL Form, ABL Dialog, ABL MDI) and the two types of customized control (ABL User Control, ABL Inherited Control). You instantiate one of these objects on the Design Canvas and then drag and drop various .NET controls (buttons, menus, labels, data binding objects, etc.) on them.

You can change properties to specify the appearance, size, color, and window management features of windows and controls. You can also use the methods of the class to manipulate windows and controls.

Event subscription allows the application to respond to user actions. For example, you can subscribe to the Activated event to do data updates whenever a window is activated.

The ABL Form

You use the ABL Form when you want a non-modal window for your application. Non-modal windows can be active when other windows in the application are active. The parent of an ABL Form cannot be a modal dialog, but an ABL Form can be the parent of a modal dialog.

You will implement ABL Forms in Creating the Customer Window on page 21, Creating the Department Window on page 41, and Creating the Purchase Order Window on page 51.

The ABL Dialog

You use the ABL Dialog when you want a modal window for your application. Modal windows require user action before the user can work with any other window in an application.

Because user response is mandatory, the ABL Dialog automatically implements **OK** and **Cancel** buttons. Event subscription and event logic for a click are pre-coded for each of these buttons.

You will implement an ABL Dialog in Creating the Login Dialog on page 67

The ABL MDI Form

You use the ABL MDI (multiple document interface) Form when you want a parent form for one or more child forms.

The ABL MDI Form includes a menu bar containing common menus (**File**, **Edit**, **View**, **Tools**, **Windows**, **Help**) and a toolbar containing common command buttons (**New**, **Open**, **Save**, **Print**, **PrintPreview**, and **Help**). Some of the menu items and command buttons include pre-coded event subscriptions and logic. The ABL MDI Form also includes a status bar at the bottom.

You will implement an ABL MDI Form in Creating the Sports Window on page 87.

The ABL User Control

The .NET controls in the Progress Developer Studio for OpenEdge Toolbox can be encapsulated into a common container, the ABL User Control. The ABL User Control combines the functionality of one or more controls, with specific properties and behaviors, into a single reusable unit. After you create an ABL User Control, you can add it to the Toolbox, where it is available to all of the forms in your project.

You will implement an ABL User Control in Adding LeftBar.cls to the project on page 68.

The ABL Inherited Control

An ABL Inherited Control is an extension of an existing control class. For example, you might create a control that has specific properties and behavior, and that inherits from the button class. After you create an ABL Inherited Control, you can add it to the Toolbox, where it is available to all of the forms in your project.

For more information about the ABL Inherited Control, see Adding HelpButton.cls to the Toolbox on page 79.

Getting help

This section describes where to locate the various resources for help on the Visual Designer.

Visual Designer online help

The primary documentation for the Visual Designer and all Progress DeveloperStudio for OpenEdge tools is the *Progress Developer Studio for OpenEdge Guide*, an online volume in the Eclipse help system.

Go to the main menu bar of the Eclipse workbench and choose **Help** > **Help Contents**. When the Eclipse help viewer opens, find and expand the Visual Designer section of the *Progress Developer Studio for OpenEdge Guide*.

Help on .NET controls

Progress Developer Studio for OpenEdge includes context-sensitive help for .NET controls. To get help, select the control on the Design Canvas and press **F1**.

The help for .NET controls appears in the Microsoft Document Explorer instead of the browser that displays Progress Developer Studio for OpenEdge help. The Microsoft Document Explorer is automatically installed when you install Progress Developer Studio for OpenEdge.

OpenEdge .NET ABL documentation

For information about using .NET objects in ABL programming, see:

- *OpenEdge Development: GUI for .NET Programming*
- *OpenEdge Development: GUI for .NET Mapping Reference*
- *OpenEdge Development: ABL Reference*
- *OpenEdge Getting Started: GUI for .NET Primer*

For information about object-oriented programming with ABL, see *OpenEdge Development: Object-oriented Programming.*

2

Creating the Customer Window

This chapter describes how to use the Visual Designer to build the Customer window, which is an implementation of an ABL Form.

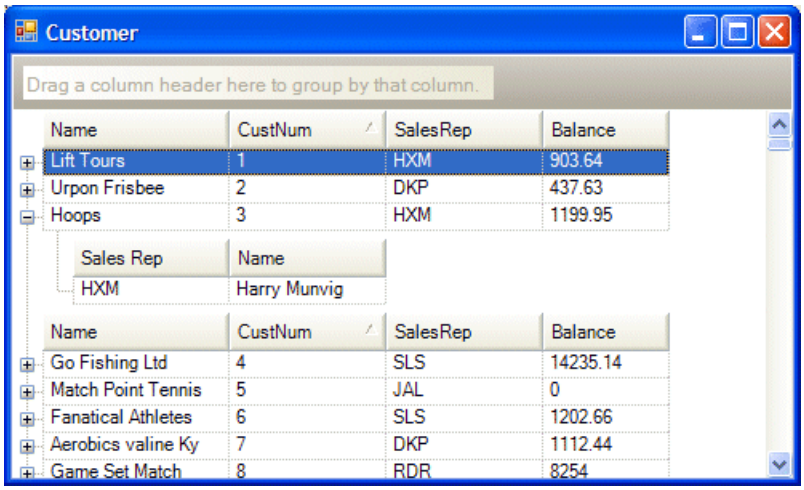
For details, see the following topics:

- Overview
- Adding customerForm.cls to the project
- Data binding
- Adding interface methods
- Adding event handlers
- Running customerForm.cls

Overview

The Customer form, shown in the following figure, features an UltraGrid control, which allows the user to browse through data. The UltraGrid control presents data in a tabular format that can be hierarchical. Data appears in bands (rows) with column heads. The bands are expandable, so you can reveal data from child tables beneath the parent table. By default, the UltraGrid control also allows the user to group the data by any of the column heads.

Figure 5: The Customer form



Tasks in this chapter

When you create the Customer form, you will:

- Add an ABL form to the project
- Implement an UltraGrid control on the form
- Bind the UltraGrid control to a ProBindingSource control
- Bind the ProBindingSource control to a data source
- Add event handlers to the UltraGrid control

Before you begin . . .

- Be sure that you have set up the SampleApp project, created the database, and started the database server as described in Setting Up the Sample Application Project.
- Start Progress Developer Studio for OpenEdge in the workspace that contains the SampleApp project.
- Verify that the SampleApp project runtime has started and is connected to the sports2000 server. (The Console view in the OpenEdge Editor perspective displays messages about project and database startup.)
- Open the OpenEdge Visual Designer perspective. The easiest way to open the OpenEdge Visual Designer perspective is to choose the icon from the tool bar and select **OpenEdge Visual Designer** from the menu.

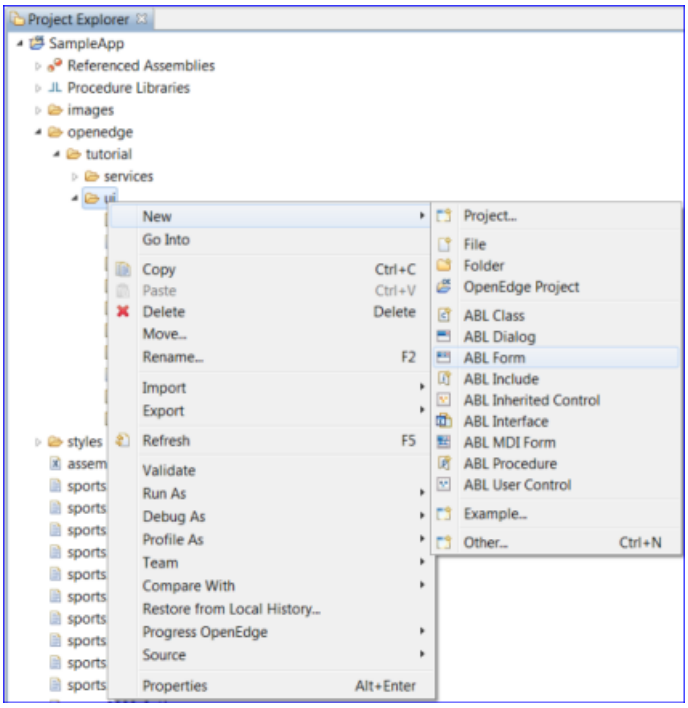
Adding customerForm.cls to the project

The `customerForm.cls` file that you will add to the project in this section is an ABL Form that implements the Customer window in the application.

An ABL Form is one of the visual containers that you can create in the Visual Designer. The ABL Form implements a non-modal window in an application. *Non-modal* means that the window can be active while other windows are also active. The Customer, Department, and Purchase Order windows in the sample application are all non-modal ABL Forms.

To add customerForm.cls to the project:

1. In the **Project Explorer** view, expand the **openedge** node in the SampleApp project.
2. Right-click on the **ui** node in the **tutorial** folder.
3. From the context menu, select **New** > **ABL Form**, as shown in the following illustration:

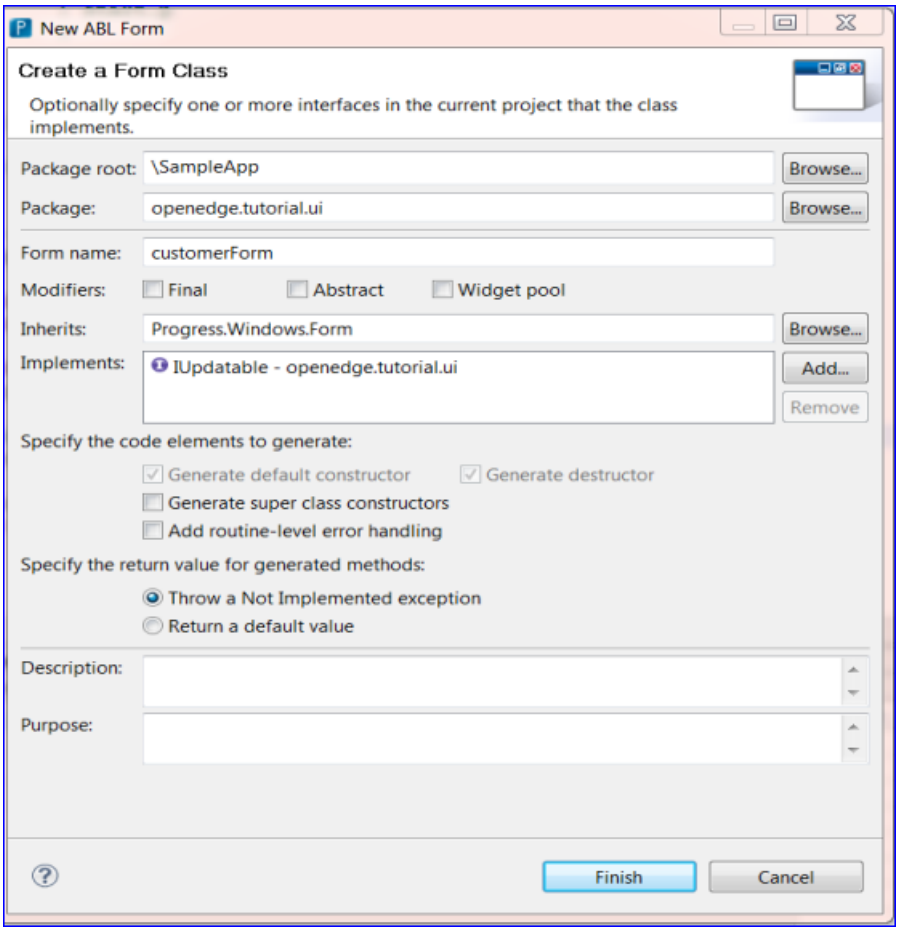


4. In the **New ABL Form** wizard, enter `customerForm` in the **Form name** field.
5. Click **Add** (next to the **Implements** field). The **Interface Selection** dialog appears.
6. Select **IUpdatable.openedge.tutorial.ui** from the **Matching interfaces** list in the **Interface Selection** dialog.

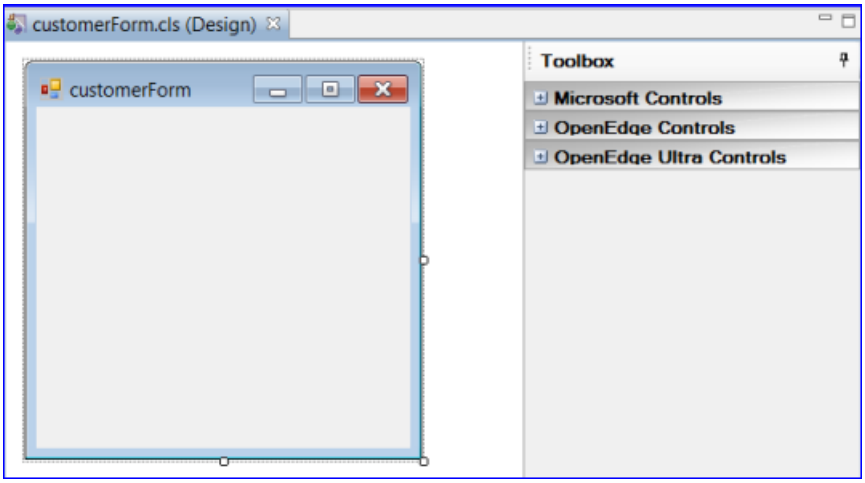
Note: If you do not see **IUpdatable.openedge.tutorial.ui** on the list, compile the project. Right-click on **SampleApp** in the **Project Explorer** view, and choose **Progress OpenEdge** > **Compile**.

`IUpdatable` is an interface. An interface is a `.cls` file that declares prototypes for methods but does not actually define those methods. Since you declared that customerForm implements `IUpdatable` in this step, you will eventually need to define the methods declared in `IUpdatable`. (For more information about interfaces and other object-oriented programming concepts, see *OpenEdge Development: Object-oriented Programming.*)

7. Click **OK** to return to the **New ABL Form** wizard. The **New ABL Form** dialog appears, as shown in the following illustration:



A `customerForm.cls` file is added to the project under the **ui** folder. The file appears in the Design Canvas of the Visual Designer and will look similar to the following illustration:



Note: If you look at `customerForm.cls` in the ABL Editor, you can see the preliminary implementation of the methods specified in the interface file, `IUpdatable.cls`. To see `customerForm.cls` in the ABL Editor, select customerForm in the Visual Designer and press **F9**. The methods specified in `IUpdatable.cls` are `AddRecord()`, `DeleteRecord()`, `SaveRecord()`, and `CancelUpdate()`. Adding interface methods on page 37 explains how to complete the implementation of these interface methods.

Changing the title bar of the form

If you are viewing `customerForm.cls` in the ABL Editor, return to the Visual Designer Editor. Press **SHIFT+F9** or click the Design tab. Notice that the title bar of the form contains the name of the class file, customerForm, by default. The name in the title bar is one of the properties that you can easily change.

To change the name of the form:

1. Select the form in the Visual Designer canvas. When a form or control is selected, an editable list of its properties appears in the **Properties** view.
2. Scroll down to the **Text** property in the **Properties** view.
3. Replace `customerForm` with `Customers`.

Note: Be aware of the difference between the Text property and the Name property. You modify the Text property when you want to change the label of a control. The Name property is the physical name of a resource and is also used in generated code.

Data binding

At this point, you can drag and drop controls from the OpenEdge Toolbox to the form. The first control you will drop is the ProBindingSource control.

The ProBindingSource control is used to connect a control to a data source. This process is known as *data binding*. Data binding is a procedure that requires these steps:

- Adding the ProBindingSource control to a form on page 25
- Defining the schema on page 27
- Defining the recursive relationship on page 30
- Attaching controls to a ProBindingSource on page 31
- Attaching the ProBindingSource control to a data source on page 34

Adding the ProBindingSource control to a form

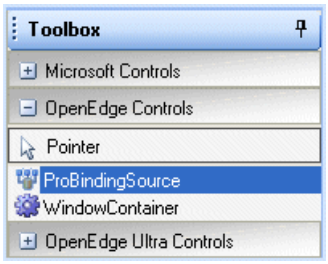
The ProBindingSource control is one of the controls that does not have a visual representation on a user interface. It is not a control that the end user directly interacts with. However, the Visual Designer represents the ProBindingSource (and other non-visual controls) by an icon and a label in a panel (the non-visual control tray) below the form. This panel allows you to see what non-visual controls are associated with a form without intruding on the graphical layout of the form.

Note: In this section, you will add a ProBindingSource to `customerForm.cls` by dragging the ProBindingSource from the Toolbox to the Customer form. An alternative to this procedure is to drag tables from the DB Structure view to the form. Another alternative is to drag an ABL source file containing a schema definition (such as a ProDataSet) to the form. All three methods start the ProBindingSource Designer where you can define the schema of the data source.

If you want to experiment with these alternatives, first read *Creating a binding source object* in the Visual Designer's online help.

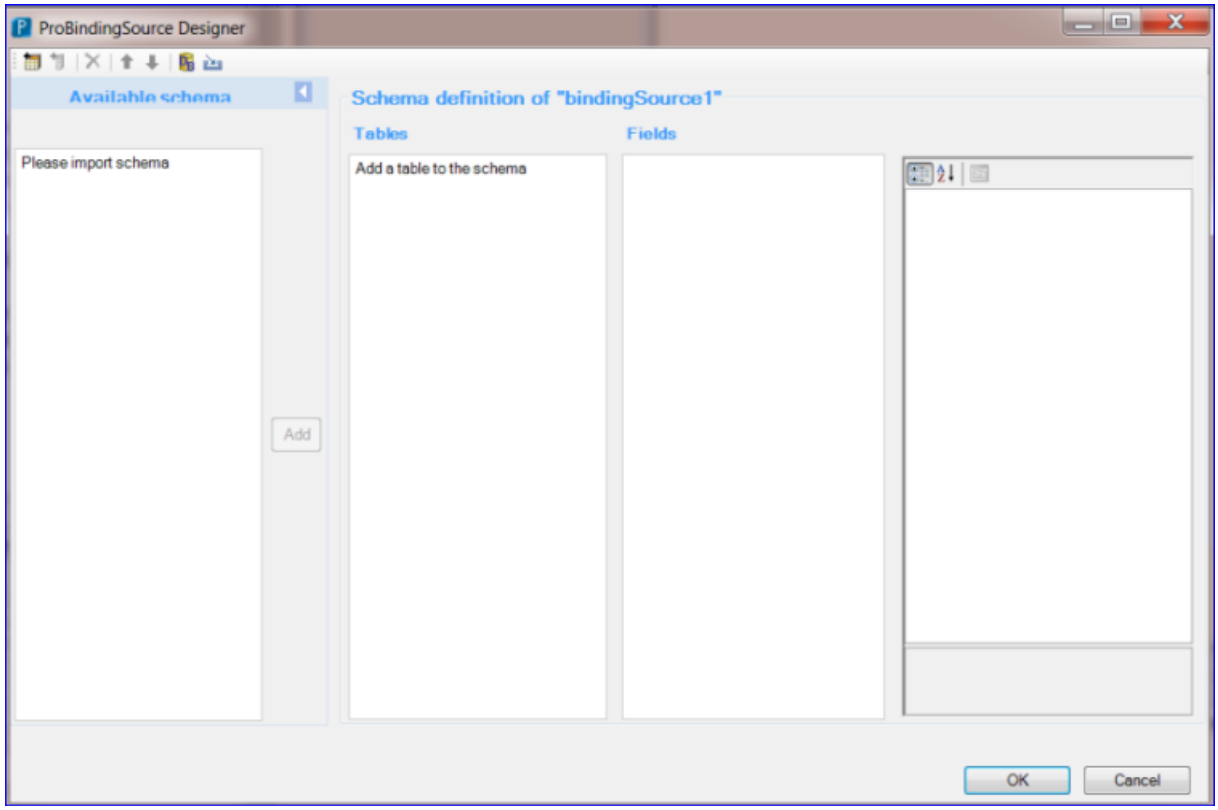
To add a ProBindingSource control to customerForm:

1. With `customerForm.cls` open in the Visual Designer canvas, expand the **OpenEdge Controls** control group in the Toolbox, as shown in the following illustration:



2. Drag and drop **ProBindingSource** on the **Customer** form.

   Because the ProBindingSource is a non-visual control (that is, it does not occupy any real estate in an application window), it appears in a panel below the Customer window. In addition, the ProBindingSource Designer appears, as shown in the following illustration:



Defining the schema

The ProBindingSource Designer allows you to define the schema of a data source in a number of ways. You can:

- Manually define each table and table column
- Import the schema from an ABL file or a schema definition (XSD) file
- Import schema information from a connected database

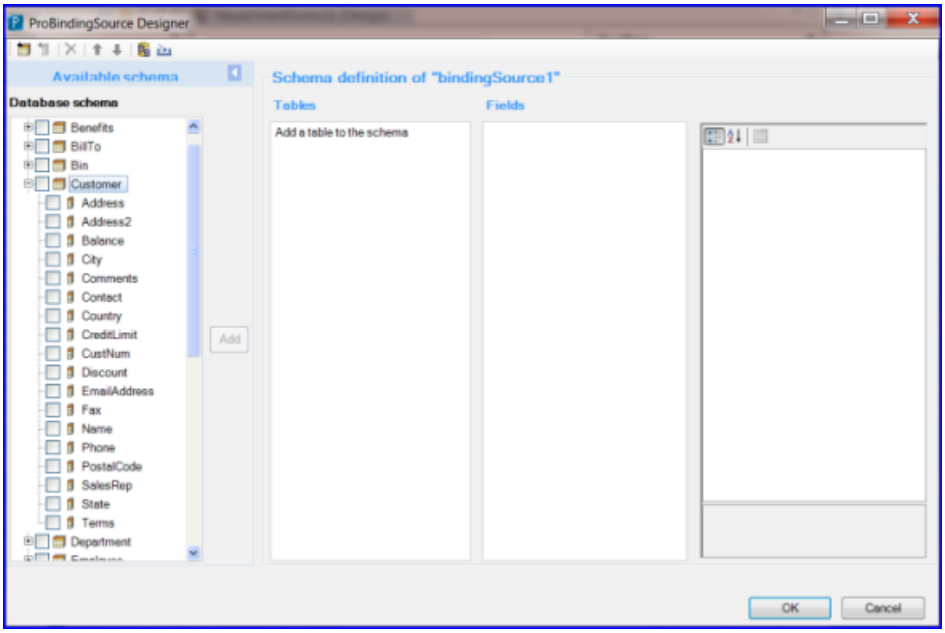
In this exercise, you will import schema from the Customer and Salesrep tables of the sports2000 database. (For an example of using an XSD file, see Adding the ProBindingSource control to a form on page 25.)

For more information about setting up the sports2000 database, see Creating the database and Adding the database connection to the project.

To define the schema in the ProBindingSource Designer:

1. If the ProBindingSource Designer is not already open, right-click on **bindingSource1** and choose **ProBindingSource Designer**.
2. Click the Import Schema from Connected Database icon on the toolbar

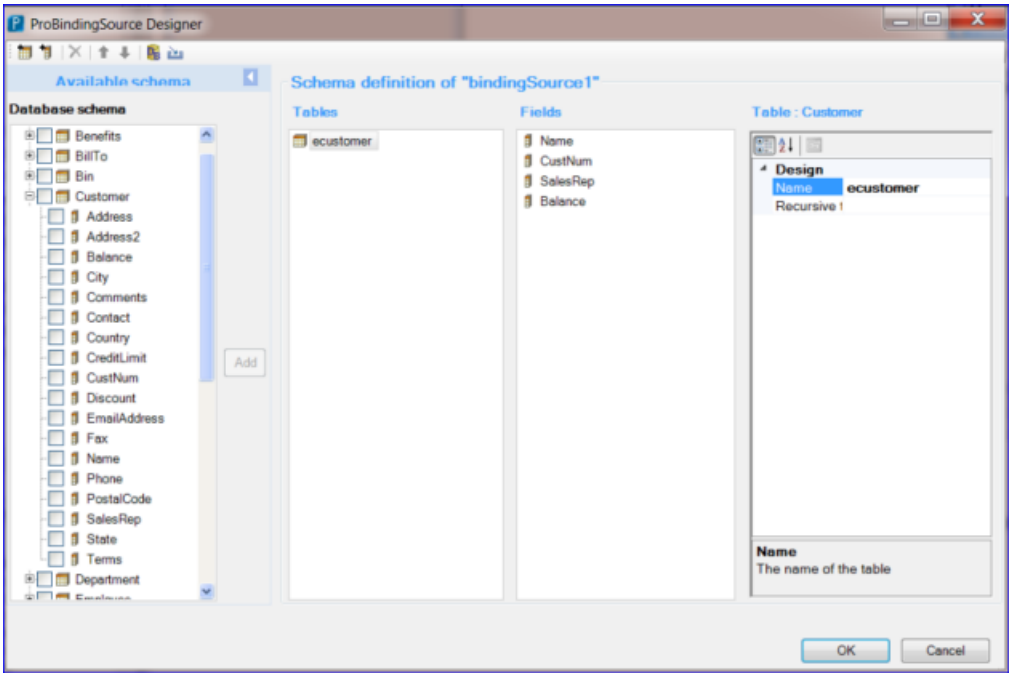
   The database schema appears in the **Available Schema** pane on the **ProBindingSource Designer** dialog. The **AvailableSchema** pane allows you to select schema information from the tables and table columns of databases that are connected to the project.
3. Expand the **sports2000** node and the **Customer** node in the Schema Selection dialog, as shown in the following illustration:



4. Select the table columns **Name**, **CustNum**, **SalesRep**, and **Balance**.
5. Click **Add**. The selected **Customer** table appears in the **Tables** pane. The table columns appear in the **Fields** pane.

Note: You can change the order of the table columns in the **Fields** pane, using **Move Selection Up** and **Move Selection Down** icons on the toolbar.

6. Select **Customer** from the **Tables** pane and change its name to **ecustomer**, as shown in the following illustration:

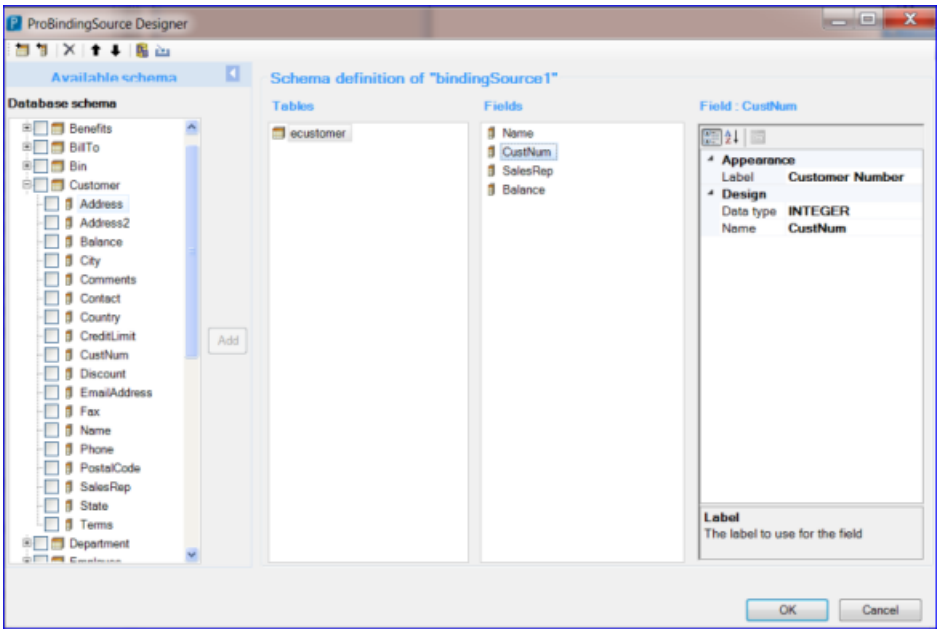


Note: Although sports2000 is the ultimate source of data for the sample application, the data is delivered at run time from business entities through a service layer. In this project, the immediate source of data is the temp-table `ecustomer`, which has the same schema as the Customer table in sports2000. All this is implemented in the `SampleApp/openedge/tutorial/services` folder that you downloaded.

7. Select each table column and change the **Name** and **Label** properties to match the following values:

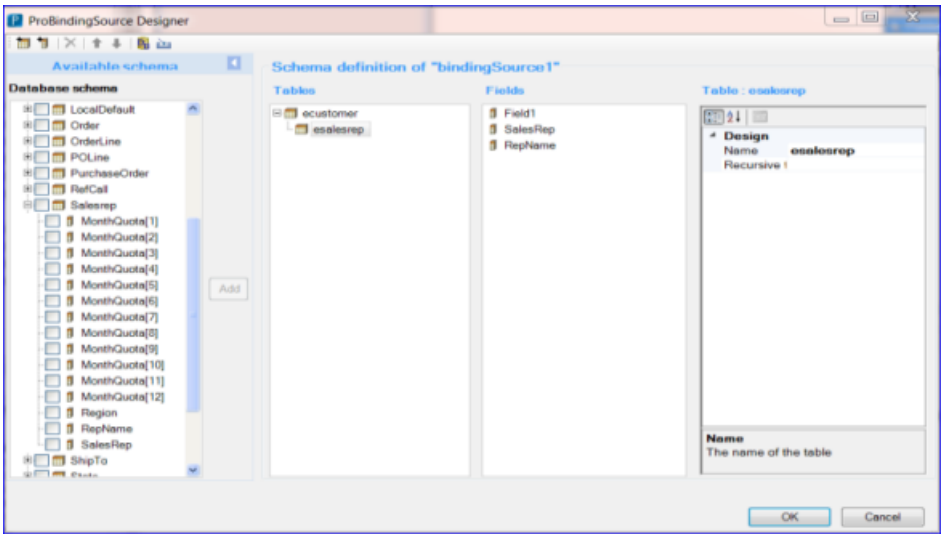
Name	Label
Name	Name
CustNum	Customer Number
SalesRep	Sales Rep
Balance	Balance

The **ProBindingSource Designer** now looks similar to the following illustration:



8. Select the **ecustomer** table in the **ProBindingSource Designer** dialog and add a child table by clicking the **Add Table** icon on the toolbar.
9. Rename the child table from **Table** to **esalesrep**.

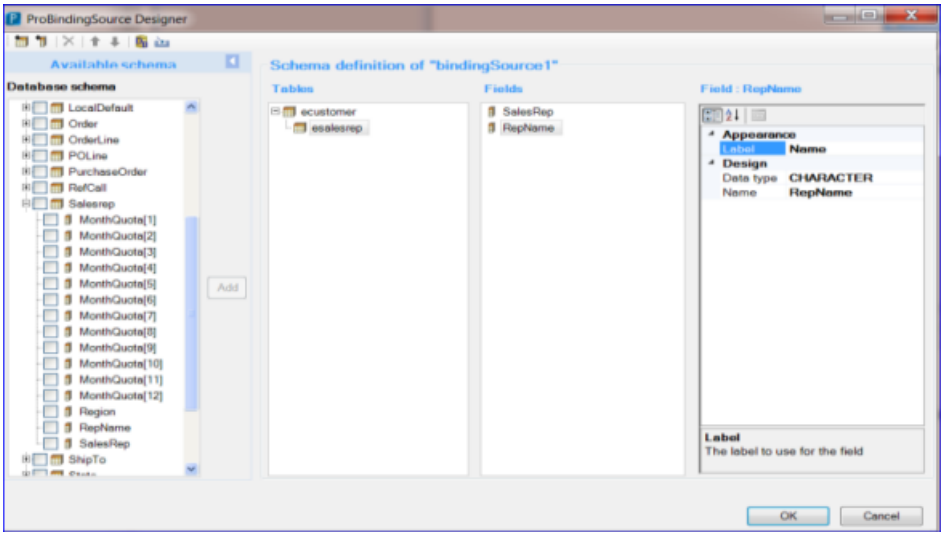
   When you add a child table, the **ProBindingSource Designer** dialog adds a default table column, **Field1** in the **Fields** pane. You will remove the default tablecolumn after you import schema from the database.
10. Select **esalesrep** and expand the **sports2000** node and the **Salesrep** node in the **Available Schema** pane.
11. Select **SalesRep** and **RepName**.
12. Click **Add**. The **ProBindingSource Designer** dialog now looks similar to the following:



13. Select each remaining table column and change the **Name**, and **Label** properties to match the following values:

Name	Label
SalesRep	Sales Rep
RepName	Name

The **ProBindingSource Designer** dialog appears, as shown in the following illustration:



14. Click **OK** to save changes to your schema definition.

Defining the recursive relationship

The ProBindingSource Designer dialog allows you to define recursive relationship between tables using the **Recursive table** property field. When defining a recursive relationship, note of the following:

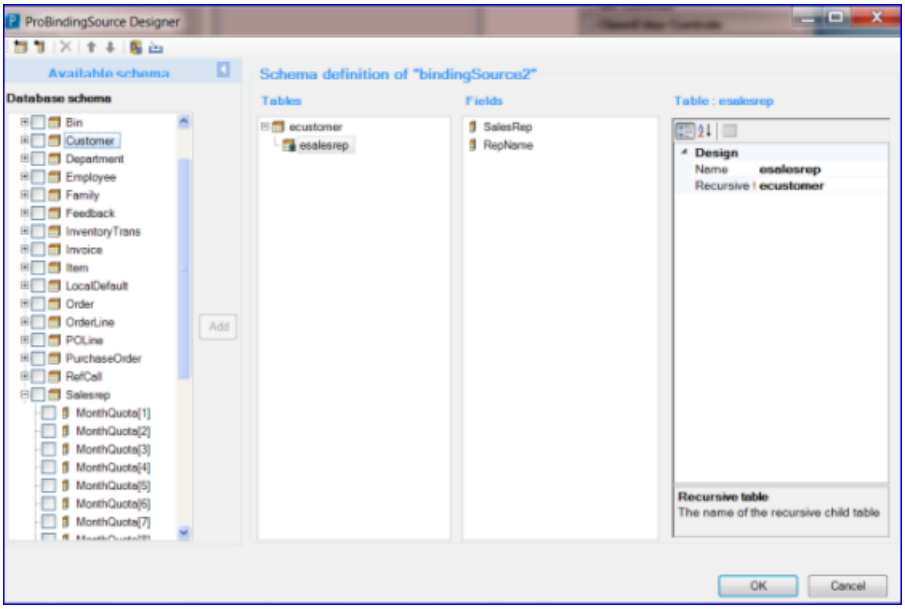
- **Recursive table** lists the name of the selected table and all its parent tables in a hierarchical order.
- A table can have a recursive relationship with either itself or with any of the parent table in its hierarchy.
- You can define only one resursive relationship in a schema definition.

To define a recursive relationship using the ProBindingSource Designer:

1. If the **ProBindingSource Designer** is not already open, right-click on **bindingSource1** and choose **ProBindingSource Designer**.
2. In the **Tables** pane, select **esalesrep**. The **Recursive table** drop-down in the properties pane lists the **esalesrep** and **ecustomer** tables.
3. Do one of the following:
   - Select **ecustomer** table from the **Recursive table** drop-down, if you want to define a recursive relationship between the esalesrep and ecustomer tables.
   - Select **esalesrep** from the **Recursive table** drop-down, if you want to define a recursive relationship with the esalesrep table itself.

Note: If a recursive relationship already exists for the selected schema definition, a message displays stating: "recursive relation already exists".

The **ProBindingSource Designer** dialog looks similar to the following illustration:



4. Click **OK** to save changes to your schema definition.

Attaching controls to a ProBindingSource

By completing the procedure in the previous section, you created an object, `bindingSource1`, that is now available as a data source for any of the controls in the Toolbox that can display or manipulate data. In this section, you will implement one such control, the UltraGrid, and bind it to `bindingSource1`. In addition, you will set some properties for the form and the control.

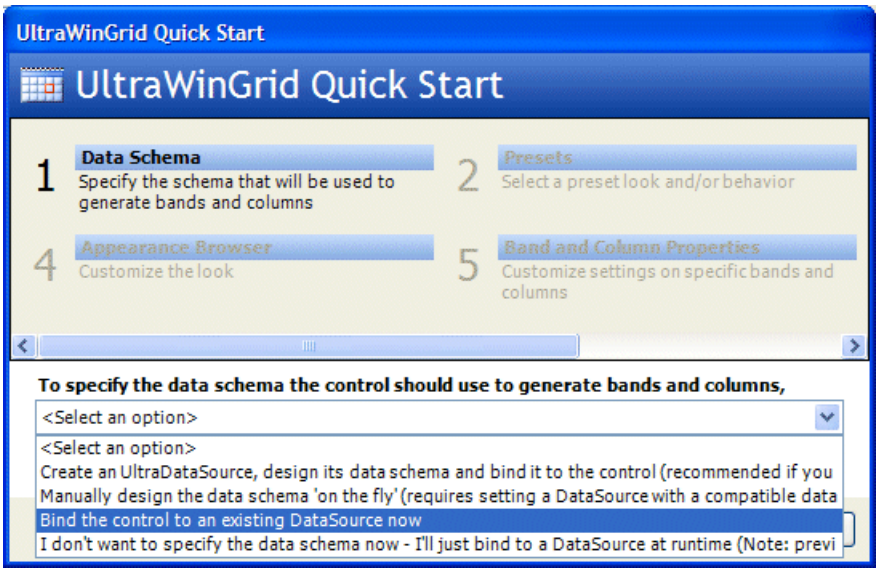
Note: The UltraGrid is similar, in some ways, to the OpenEdge browse widget. The key advantage of the UltraGrid is that it can manage and display multiple parent-child levels of data, such as the ecustomer and esalesrep tables you defined for bindingSource1.

To connect an UltraGrid to bindingSource1:

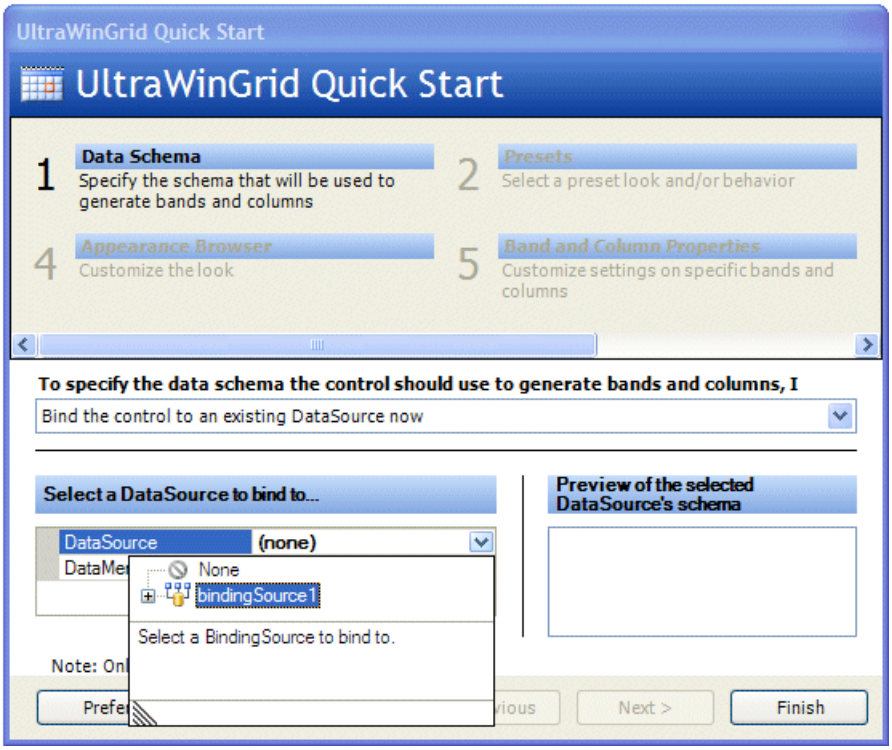
1. With `customerForm.cls` open in the Visual Designer canvas, expand the **OpenEdge Ultra Controls** control group in the **Toolbox**.
2. Drag and drop the UltraGrid control from the **Toolbox** to the Customer form.

   For many of the controls, the default behavior after a drag and drop is to display a wizard that walks you through the process of setting the properties of the control. When you drop the UltraGrid control, the **UltraWinGrid Quick Start** wizard appears.

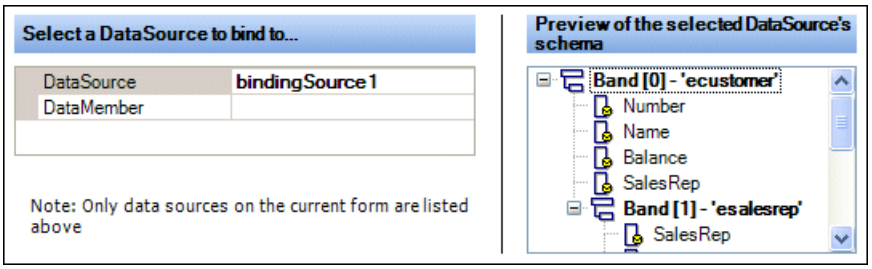
3. On the **Data Schema** page of the **UltraWinGrid Quick Start** wizard, select **Bind the control to an existing DataSource now** from the drop-down menu as shown in the following illustration:



4. Select **DataSource**, and then select **bindingSource1** from the drop-down menu as shown in the following illustration:



After you select **bindingSource1**, a preview of its schema looks similar to the following:



5. Click **Finish**. For the Customer form, it is not necessary to change anything else.

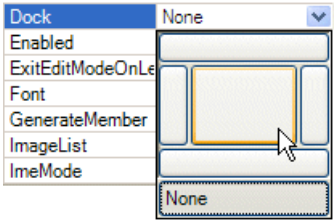
There are many more customizations that you can perform if you continue with the wizard. However, if you are familiar with the properties of a control, it is usually quicker to make changes by editing property values directly in the **Properties** view.

At this point you can align the UltraGrid control with respect to the borders of the form.

Positioning the UltraGrid control

To position the UltraGrid control within the form:

1. Select the **ultragrid1** control on the Design Canvas.
2. Select the **Dock** property in the **Properties** view.
3. From the drop-down list, select the middle option, **Fill**, which binds the control to the four borders of the form as shown in the following illustration:

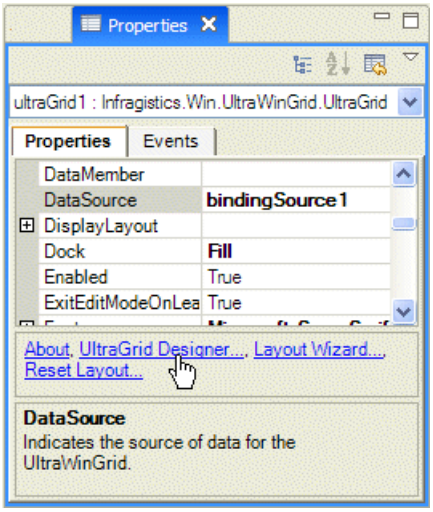


After you set the **Dock** property, the UltraGrid control moves to the appropriate alignment within the form. You might need to adjust the width of the form to see all of the fields.

Notice in the **Properties** view that the **DataSource** property is set to `bindingSource1`. This was set previously in the UltraWinGrid Quick Start wizard. Instead of using the wizard, you can set **DataSource** directly in the **Properties** view.

Also notice in the **Properties** view shown in the following figure, there are links to several tools that can help you set the properties for the control.

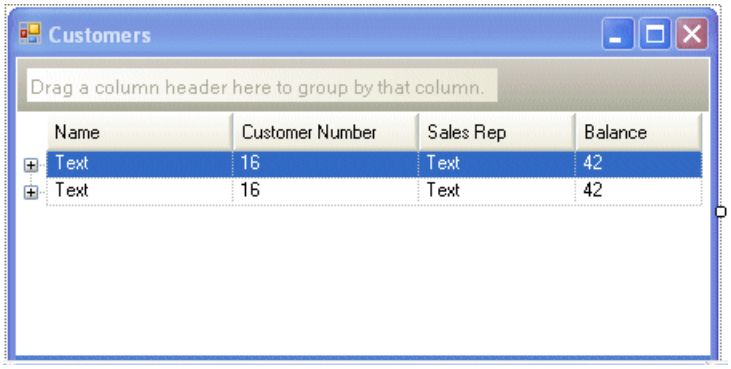
Figure 6: Tools links in the Properties view



Attaching the ProBindingSource control to a data source

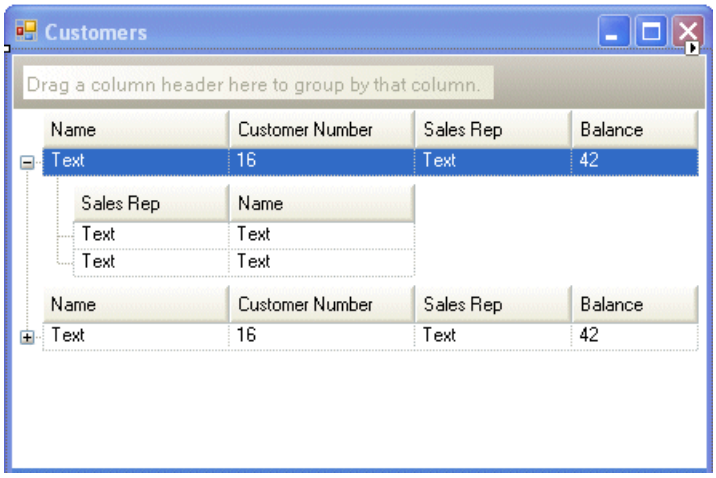
The ProBindingSource control, **bindingSource1**, is now the data source for the UltraGrid control. The UltraGrid control on the Design Canvas (see the following figure) displays bands of data with representative values, based on data type, in various fields. It also shows the label specified for each field.

Figure 7: The UltraGrid control in the Customer form



When you expand the node, bands representing the child table appear, as shown in the following figure. The child table is **esalesrep**, which you specified when you created the ProBindingSource control, **bindingSource1**.

Figure 8: The UltraGrid with the data band expanded



At run time, the UltraGrid is filled with Customer and SalesRep data supplied by the ProBindingSource. Before you can fill the UltraGrid control with actual data, however, you have to attach the ProBindingSource to an ABL data source. You can think of a ProBindingSource as a kind of intermediary between .NET controls and ABL data sources. It facilitates the transfer of data, and it takes care of the mapping of data types between .NET and ABL.

Up to this point, you only have half of the binding in place—the binding between **ultraGrid1** and **bindingSource1**. To complete the binding you need to bind **bindingSource1** to the ABL data source, which, in the Sample Application, is accessed through a service adapter.

Note: Use of a service adapter conforms to the layered approach to data access and application design. The UI (`customerForm.cls`) is separate from the business logic (`beCustMaint.cls`), which it accesses through a service adapter (`serviceAdapter.cls`). The business logic gets its data from the sports2000 database, which is running as a resource to the SampleApp project.

The code for binding **bindingSource1** to the data source is not automatically generated. You must edit `customerForm.cls` to add the code necessary to perform the binding. That code also includes open and close query methods and updates to the `CONSTRUCTOR` and `DESTRUCTOR`.

Caution: Be careful not to modify any of the code that was generated by the Visual Designer. There is a danger you will not be able to reopen a file in the Visual Designer after modifying generated code in the ABL Editor.

In particular:

Do not add, edit, or delete any `VisualDesigner` annotations. Annotations are lines that are preceded by `@` (for example, `@VisualDesigner.FormMember.`).

Do not add, edit, or delete any code in the `InitializeComponent()` method.

Do not delete or edit any code in the `CONSTRUCTOR` or `DESTRUCTOR`. You can, however, add code to these blocks.

To define the query:

1. With `customerForm.cls` open in the Visual Designer Editor, press **F9** to open the file in the ABL Editor.

You can also select `customerForm.cls` in the **Project Explorer** view, and choose **Open With** > **OpenEdge ABL Editor** from the context menu.

Note: The Visual Designer and the ABL Editor are synchronized so that a change in one is reflected in the other. For example, if you add a control to a form by dragging and dropping in the Visual Designer, the code visible in the ABL Editor will be updated.

2. Define a variable for the service adapter by adding the following declaration:

```
DEFINE PRIVATE VARIABLE oServiceAdapter AS openedge.tutorial.services.serviceAdapter.
```

You can add it to the other variable declarations, which appear after the `CLASS` declaration at the beginning of the file.

Note: If you are viewing this manual on line, you can copy code snippets from the manual directly to the file in the ABL Editor. The code might be poorly formatted, but it will run.

An alternative is to copy the code from the completed files in the `SampleAppSolution\openedge\tutorial\ui` folder. The `SampleAppSolution` folder should be in the same workspace as the `SampleApp` folder.

3. Define the open and close query methods by adding the following code before the `DESTRUCTOR` statement near the end of the file:

```
/* openQuery */
   METHOD PRIVATE VOID openQuery():
        bindingSource1:handle = oServiceAdapter:getDatasetHandle().
    END METHOD.

  /* closeQuery */
    METHOD PRIVATE VOID closeQuery():
        oServiceAdapter:closeQuery().
    END METHOD.
```

Note: You can add methods with the Add Methods dialog (**Source** > **Add Method**), which provides a standardized template. However, in this tutorial, it is easier to copy the code samples from this book and paste them into your source files.

4. Press **CTRL+S** to save your changes.

Updating the existing constructor and destructor

To update the existing constructor and destructor:

1. Find the `CONSTRUCTOR` declaration in the file.
2. Add the code shown in bold:

```
CONSTRUCTOR PUBLIC  customerForm (  ):

SUPER().
oServiceAdapter = NEW
openedge.tutorial.services.ServiceAdapter("CustMaint").
InitializeComponent ( ).
OpenQuery().
CATCH e AS Progress.Lang.Error:
            UNDO, THROW e.
END CATCH.
END CONSTRUCTOR.
```

   This step implements data access by creating a service adapter that uses a ProDataSet (`CustMaint`) that is defined in `BECustMaint.cls`.

3. Find the `DESTRUCTOR` declaration in the file.
4. Add the code shown in bold:

```
DESTRUCTOR PUBLIC customerForm ( ):

closeQuery().

IF VALID-OBJECT(components) THEN DO:
CAST(components, System.IDisposable):Dispose().
END.

END DESTRUCTOR.
```

5. Press **CTRL+S** to save your changes.

Adding interface methods

When you created `custormerForm.cls`, you specified that it implements the interface, `IUpdatable.cls`. `IUpdatable.cls` defines a number of methods for adding, deleting, saving, and canceling updates to database records. Stub code for each of the interface methods is automatically added to the file by default. The stub code has the same signature as the methods specified in the interface file, but the methods themselves are not implemented.

In this exercise, you will replace the `DeleteRecord` and `SaveRecord` stub code with methods that actually do something.

You will not change the stub code for the AddRecord and `CancelUpdate` methods. They are not used by any control in the Customer form, so it is not necessary to implement them. However, methods with the same signatures as those defined in the interface file are required. If methods with identical signatures do not exist, you will get compiler errors. Therefore, you should retain the generated stub code, which looks like the following:

```
METHOD PUBLIC VOID AddRecord(  ):
UNDO, THROW NEW Progress.Lang.AppError("METHOD NOT IMPLEMENTED").
END METHOD.

METHOD PUBLIC VOID CancelUpdate(  ):
UNDO, THROW NEW Progress.Lang.AppError("METHOD NOT IMPLEMENTED").
END METHOD.
```

To update the DeleteRecord and SaveRecord interface methods in customerForm.cls:

1. Open `customerForm.cls` in the ABL Editor.
2. Replace the stub code of the `DeleteRecord` and `SaveRecord` methods with the following:

```
METHOD PUBLIC VOID DeleteRecord(  ):
    oServiceAdapter:RemoveRecord
    (bindingSource1:Handle:get-buffer-handle('ecustomer')).
    oServiceAdapter:SaveData().
END METHOD.

METHOD PUBLIC VOID SaveRecord(  ):
     IF bindingSource1:RowModified THEN
     DO:
        bindingSource1:Assign().
        oServiceAdapter:SaveData().
     END.
END METHOD.
```

3. Save `customerForm.cls`.

Adding event handlers

Adding event handlers to controls is a two-step process:

1. Subscribe to one of the events that the control supports by choosing it in the **Events** tab of the **Properties** view.
2. Code the event-handling method, using the ABL Editor.

When you choose an event in the **Events** tab of the **Properties** view, the Visual Designer generates the appropriate `SUBSCRIBE` statement in the source code. It also generates the event-handling method. Progress Developer Studio for OpenEdge automatically opens the class file in the ABL Editor (or switches to it if it is already open) and positions the cursor in the method definition.

Note: Most controls have a default event that you can subscribe to by double-clicking the control in the Visual Designer. The `SUBSCRIBE` statement is automatically added to the source code, and the cursor is positioned in the method definition.

In the Customer form you will enable the UltraGrid to save changes made to a field by running SaveRecord() after a field is edited. Also, you will enable the deletion of the contents of a field by running DeleteRecord() when the **DELETE** key is pressed.

To add event handlers to ultraGrid1:

1. Open `customerForm.cls` in the Visual Designer.
2. Select **ultraGrid1** on the Visual Designer canvas, or in the drop-down list at the top of the **Properties** view.
3. In the **Properties** view, select the **Events** tab.
4. Scroll down to the **BeforeRowUpdate** event and double-click it. The Visual Designer adds a template for the event handler and positions the cursor in the method declaration in the source file.
5. Add the code in bold to the event handler declaration:

```
METHOD PRIVATE VOID ultraGrid1_BeforeRowUpdate(INPUT sender ASSystem.Object,
INPUT e AS Infragistics.Win.UltraWinGrid.CancelableRowEventArgs ):
THIS-OBJECT:SaveRecord().
RETURN.
END METHOD.
```

6. In the **Properties** view, scroll to the **BeforeRowsDeleted** event and double- click it.
7. Add the code in bold to the event handler declaration:

```
METHOD PRIVATE VOID ultraGrid1_BeforeRowsDeleted
(INPUT sender AS System.Object,
INPUT e AS Infragistics.Win.UltraWinGrid.BeforeRowsDeletedEventArgs ):
THIS-OBJECT:DeleteRecord().
RETURN.
END METHOD.
```

8. Save `customerForm.cls`.

Running customerForm.cls

To run customerForm.cls:

1. Save `customerForm.cls`.
2. Select `customerForm.cls` in the **Project Explorer** view.
3. Select **Run** > **Run As** > **Progress OpenEdge Application** on the main menu.

When the Customer form runs, you can check its functionality, which includes:

- Displaying data from the **Customer** table of the sports2000 database in the primary or parent band.
- Expanding a parent band to display a child band, which contains related data from the **Salesrep** table of the sports2000 database.
- Editing data fields. The UltraGrid control implements the standard editing features that you would expect. For example, you can modify, copy, and paste the contents of a field. Also, when you right-click on a field, you see a drop-down menu with standard editing commands.

- Changing the sort order by clicking on a column heading.
- Moving columns by dragging the column headers to another position.
- Group columns by dragging column headers to the upper area of the form. You can restore the default view by dragging the column header back down to the lower area of the form.

3

Creating the Department Window

This chapter describes how to use the Visual Designer to build the Department window, an implementation of an ABL Form, to the sample application.

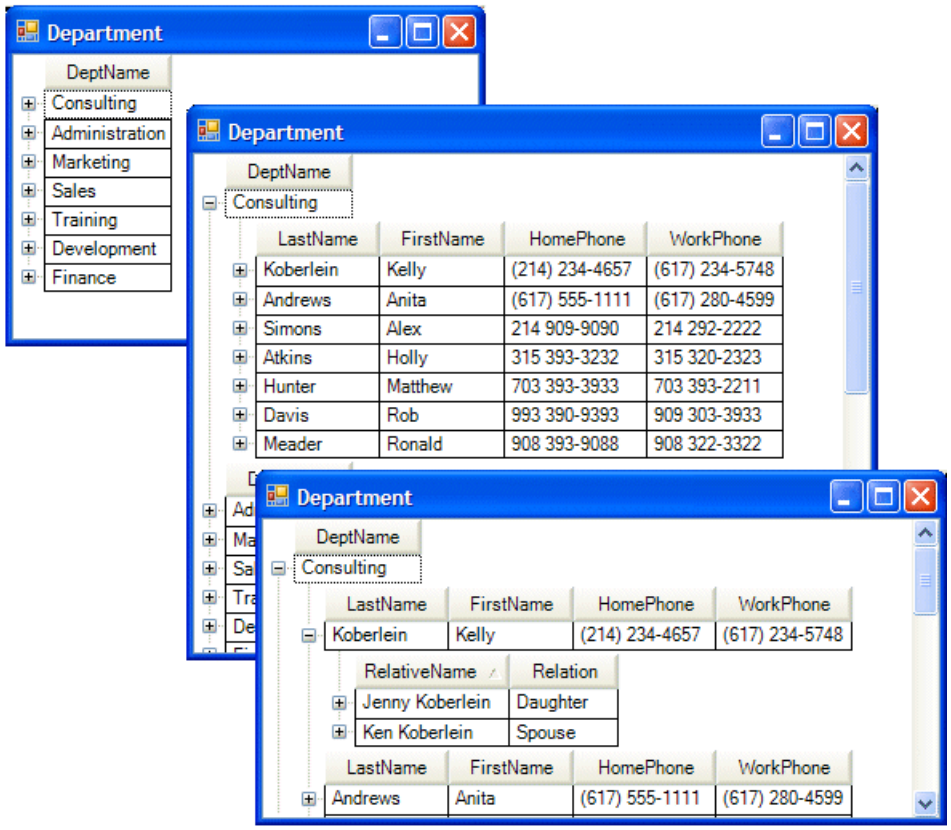
For details, see the following topics:

- Overview
- Adding departmentForm.cls to the project
- Adding a ProBindingSource control to the form
- Adding an UltraTree control to the form
- Adding the code that completes data binding
- Running departmentForm.cls

Overview

The Department form, shown running in the following figure, features an UltraTree control. The UltraTree control is useful for creating lists of data displayed as expandable nodes. When a node is expanded, data can be displayed in a tabular format similar to the UltraGrid.

Figure 9: The Department form



Tasks in this chapter

When you create the Department form, you will:

- Add an ABL form to the project
- Implement an UltraTree control in the form
- Import data source schema from an XSD file
- Bind the UltraTree control to a ProBindingSource control
- Use the UltraTree ColumnSet Layout Designer to define what data is displayed
- Bind the ProBindingSource control to a data source

Before you begin . . .

- Be sure that you have set up the SampleApp project, created the database, and started the database server as described in Setting Up the Sample Application Project.
- Start Progress Developer Studio for OpenEdge in the workspace that contains the SampleApp project.
- Verify that the SampleApp project AVM (runtime) has started and is connected to the sports2000 server.
- Open the OpenEdge Visual Designer perspective.

Adding departmentForm.cls to the project

The `departmentForm.cls` file is an ABL Form that implements the Department window in the application.

This section is a quick summary of how to add `departmentForm.cls` to the SampleApp project. The procedure is essentially the same as the one described in the Adding customerForm.cls to the project on page 23. That section provides more detailed information about the steps in this section.

To add departmentForm.cls to the project:

1. Expand the **openedge** node in the SampleApp project in the **Project Explorer** view.
2. Right-click on the **ui** node in the **tutorial** folder.
3. From the context menu, select **New** > **ABL Form**.
4. In the **New ABL Form** wizard, enter `departmentForm` in the **Form name** field.
5. Click **Finish**.
6. Scroll down to the **Text** property in the **Properties** view, and change `departmentForm` to `Department`.

Adding a ProBindingSource control to the form

To configure the ProBindingSource control in this section, you will import schema information from an XSD file instead of importing schema from a database (which is described in Defining the schema on page 27). Importing schema information from an XSD file is faster and less error-prone than other methods.

See Data binding on page 25 for more details about data binding.

To add a ProBindingSource control to the Department form:

1. With `departmentForm.cls` open in the Visual Designer canvas, expand the **OpenEdge Controls** control group in the OpenEdge Toolbox.
2. Drag **ProBindingSource** and drop it on the **Department** form.
3. When the **ProBindingSource Designer** appears, click the **Import Schema from File** icon  on the toolbar. The **Open** dialog appears.
4. Go to the `openedge\tutorial\services` folder in theSampleApp project and choose the `department.xsd` file.

   The `department.xsd` describes the schema of the `edepartment`, `eemployee`, and `efamily` temp tables.

Note: Creating an XML Schema Definition (XSD) file is a portable and efficient way to describe the schema of a data source. You can create XSD files from databases, temp tables, or ProDataSets with the ABL `WRITE-XMLSCHEMA` method.

5. Click **Open**. The schema from `department.xsd` is added to the **Available Schema**pane in the **ProBindingSource Designer** dialog.
6. Select the tables and columns that you want add to your schema definition from the **Available Schema** pane and click **Add**.
7. Click **OK** in the **ProBindingSource Designer** to save your changes.

When you complete this procedure, you have a ProBindingSource control with the default name **bindingSource1**. This is the name that identifies the control in code. You can change the default name to something more meaningful by changing the **Name** property in the **Properties** view.

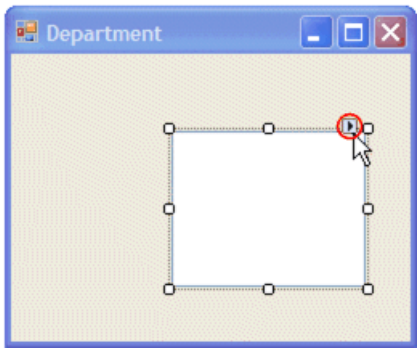
Adding an UltraTree control to the form

This section describes how to add and configure an UltraTree control. You bind the UltraTree control to the ProBindingSource control and then use the UltraTree ColumnSet Layout Designer to define what data is displayed.

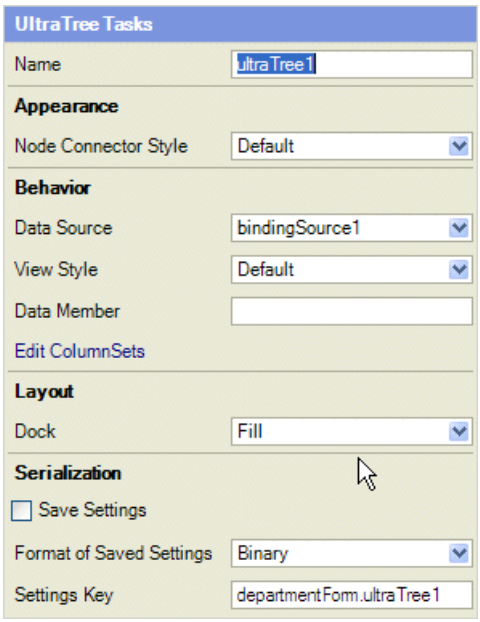
To add and configure an UltraTree control:

1. Find **UltraTree** in the **OpenEdge Ultra Controls** control group in the OpenEdge Toolbox.
2. Drag **UltraTree** to the **Departments** form on the Design Canvas.
3. Select the UltraTree control on the Design Canvas, click the **Dock** property in the **Properties** view and choose the middle option, `Fill`, from the drop-down. The control binds to the four borders of the **Departments** form.
4. Click the Smart Tag arrow on the UltraTree control on the Design Canvas, to open the Smart Tag pop-up panel.

   Smart Tags offer a list of the most frequently used properties for the control. If Smart Tags are available, the control has a small right-arrow button at the top right corner when it is selected,as shown in the following illustration:

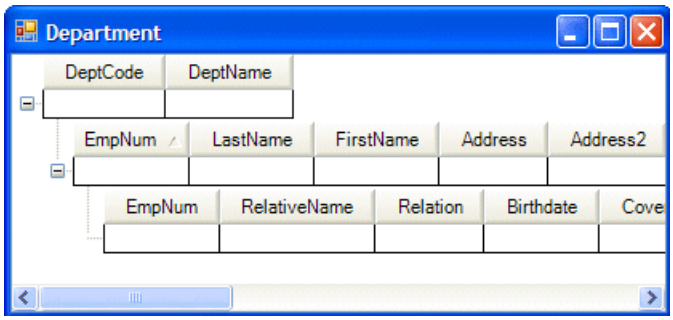


5. In the Smart Tag panel, set **DataSource** to **bindingSource1** as shown in the following illustration:



The following figure shows the UltraTree, after data binding, with all the nodes expanded.

Figure 10: The UltraTree bound to a ProBindingSource



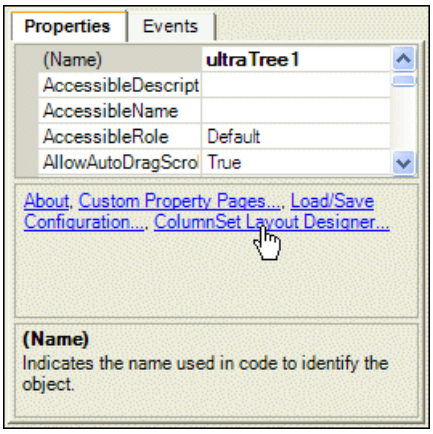
Notice that all the fields in all three tables (`edepartment`, `eemployee`, and `efamily`) are represented. This reflects the schema imported into the ProBindingSource. In a real world application, you might want to hide some of the data. You can eliminate unwanted fields directly in the ProBindingSource control by removing them with the ProBindingSource Designer.

However, you might want the ProBindingSource to include all the data, so it can be bound to a number of controls. For example, one control can implement a browse that supplies names, addresses, and phone numbers and suppresses all other information. Another control, bound to the same ProBindingSource, can implement a browse that displays only employee names, vacation days, and sick days. The ColumnSet Layout Designer of the UltraTree control allows you to specify which fields to show and which fields to hide.

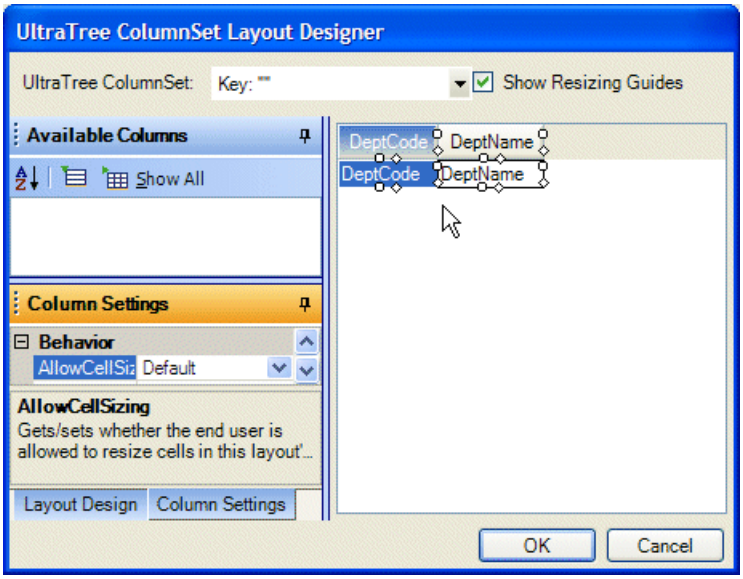
Using the ColumnSet Layout Designer

To use the ColumnSet Layout Designer to hide fields:

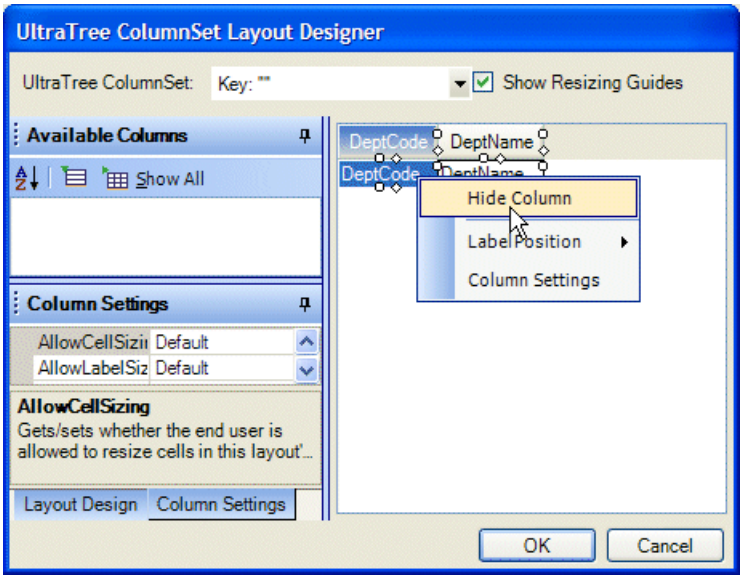
1. Select the UltraTree control in the **Departments** form.
2. Select **ColumnSet Layout Designer** from the bottom of the **Properties** view, as shown in the following illustration:



The ColumnSet Layout Designer displays the fields of the root table in the right pane, as shown in the following illustration:



3. In the ColumnSet Layout Designer, right-click on **DeptCode** and choose **Hide Column**. The dialog looks similar to the following illustration:

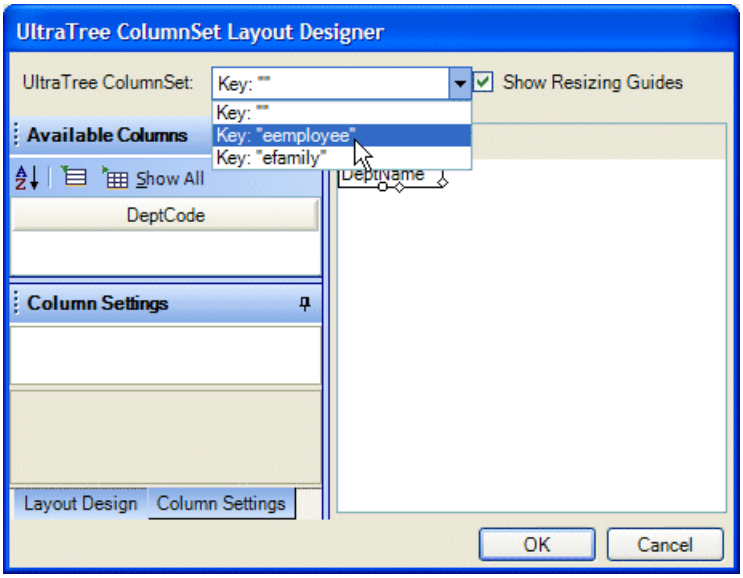


The **DeptCode** column moves to the **Available Column** list, indicating that it is removed from the display.

Note: In addition to hiding fields, you can use the ColumnSet Layout Designer to change a number of other properties, particularly properties that affect run-time behavior. For example, you can specify whether or not a user can reposition fields. If you select the **Layout Design** or **Column Settings** tab, you can explore a list of these properties. Tool tips are available to explain the use of each property.

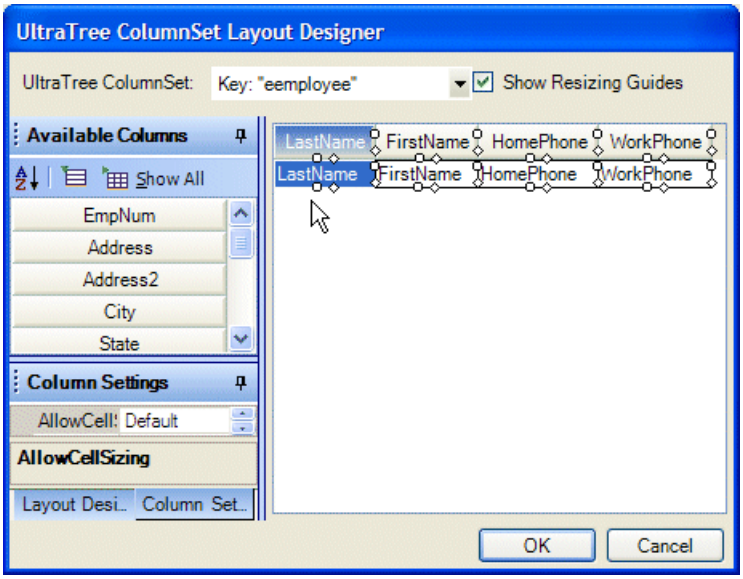
---

4. Select **eemployee** from the **UltraTree ColumnSet** field, as shown in the following illustration:



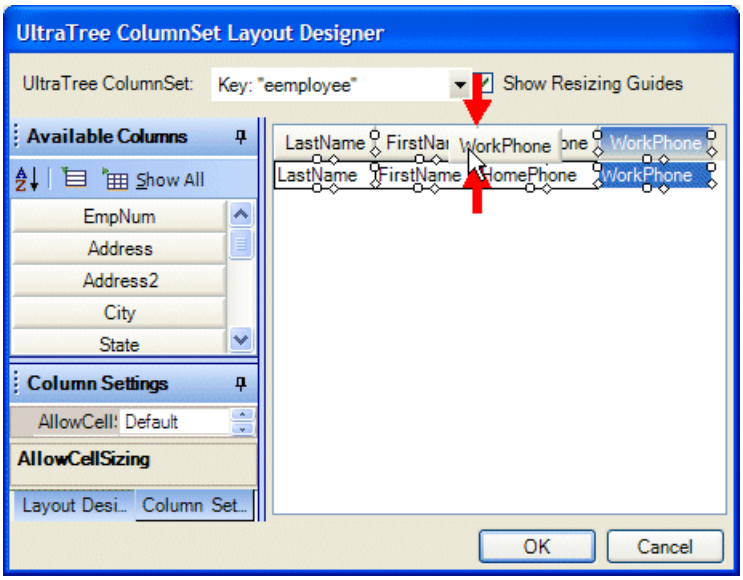
The **ColumnSet Layout Designer** displays the fields of the child table `eemployee` in the right pane.

5. Hide all the columns except **LastName**, **FirstName**, **HomePhone**, and **WorkPhone**. When you finish, the ColumnSet Layout Designer should look something like this:



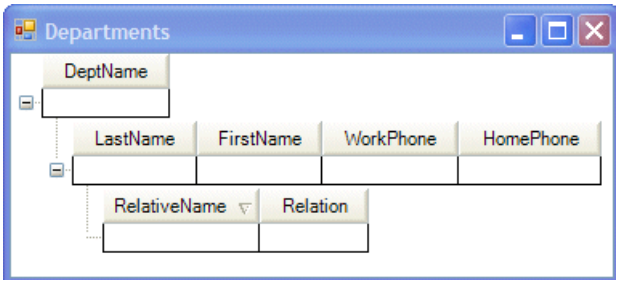
Note that you can reposition the columns by dragging the column heads. For example, drag the **WorkPhone** column head between **FirstName** and **HomePhone**.

Red arrows indicate if the column is repositioned between, above, below, or adjacent to the destination columns. For example, the arrows in the following illustration indicate that **WorkPhone** will be placed between **FirstName** and **HomePhone**:



6. Select **efamily** from the **UltraTree ColumnSet** field.
7. Hide all of the columns except **RelativeName** and **Relation**.
8. Click **OK** to save your changes and exit the ColumnSet Layout Designer.

   After you hide the specified fields, `departmentForm.cls` should look similar to the following in Visual Designer:



Adding the code that completes data binding

At this point, the UltraTree control is bound to **bindingSource1** as its data source. To complete the data binding procedure, you must add the code that binds **bindingSource1** to the ProDataSet, created in `beDepartment.cls`. The code is very similar to the code you added to `customerForm.cls` in Adding the ProBindingSource control to a form on page 25.

To bind bindingSource1 to its data source:

1. With `departmentForm.cls` open in the ABL Editor, define a variable for the service adapter by adding the following declaration:

```
DEFINE PRIVATE VARIABLE oServiceAdapter AS openedge.tutorial.services.serviceAdapter.
```

You can add it to the other variable declarations, which appear after the CLASS declaration at the beginning of the file.

2. Define the open and close query methods by adding the following code before the DESTRUCTOR statement:

```
/* openQuery */
   METHOD PRIVATE VOID openQuery():
        bindingSource1:handle = oServiceAdapter:getDatasetHandle().
    END METHOD.

  /* closeQuery*/
    METHOD PRIVATE VOID closeQuery():
        oServiceAdapter:closeQuery().
    END METHOD.
```

3. Find the CONSTRUCTOR declaration in the file.
4. Add the code shown in bold:

```
CONSTRUCTOR PUBLIC departmentForm (  ):

SUPER().
oServiceAdapter = NEW ServiceAdapter("Department").
InitializeComponent ( ).
openQuery().
CATCH e AS Progress.Lang.Error:
            UNDO, THROW e.
END CATCH.
END CONSTRUCTOR.
```

This step implements data access by creating a service adapter that uses a ProDataSet (Department) that is defined in BEDepartment.cls.

5. Find the DESTRUCTOR declaration in the file.
6. Add the code shown in bold:

```
DESTRUCTOR PUBLIC departmentForm ( ):

closeQuery().
IF VALID-OBJECT(components) THEN DO:
CAST(components, System.IDisposable):Dispose().
END.

END DESTRUCTOR.
```

7. Press **CTRL+S** to save your changes.

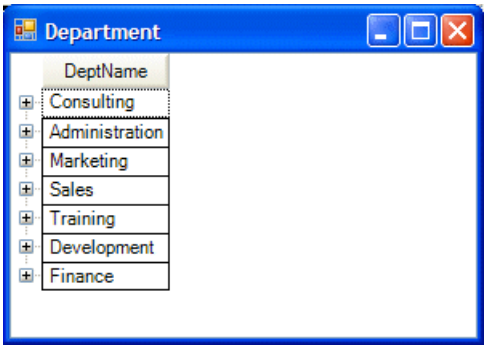
Running departmentForm.cls

To run departmentForm.cls:

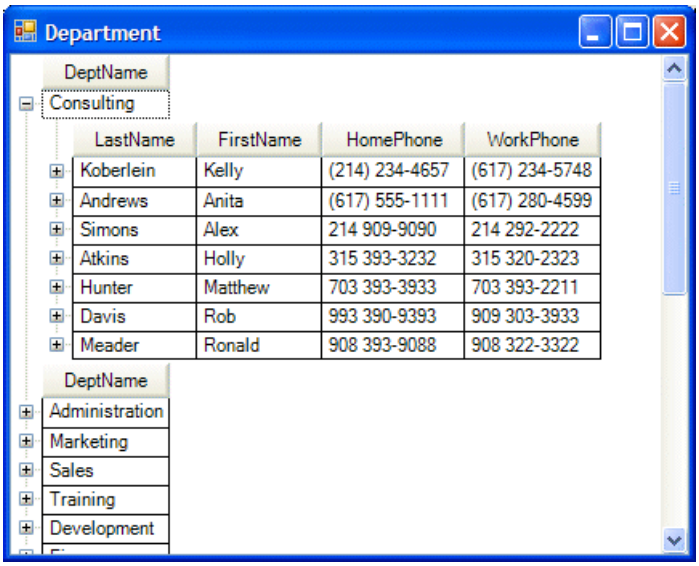
1. Save `departmentForm.cls`, if necessary.
2. Select `departmentForm.cls` in the **Project Explorer** view.
3. Select **Run** > **Run As** > **Progress OpenEdge Application** on the main menu.

When the Department form runs, you can check its functionality, which includes:

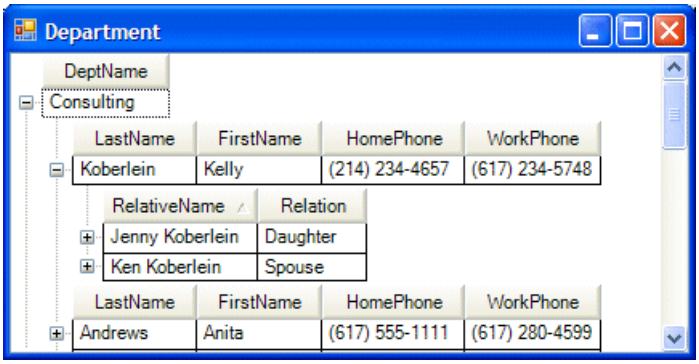
- Displaying DeptNames as nodes in a tree view, as shown in the following illustration:



- Expanding a department node to display employee information for a department, as shown in the following illustration:



- Expanding an employee node to display family information, as shown in the following illustration:



- Changing the sort order by clicking a column heading
- Resizing column widths

4

Creating the Purchase Order Window

This chapter describes how to use the OpenEdge Visual Designer to build the Purchase Order form of the sample application.

For details, see the following topics:

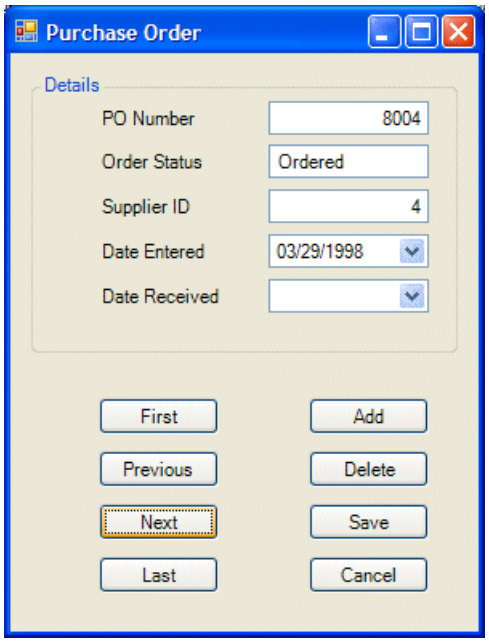
- Overview
- Adding purchaseOrderForm.cls to the project
- Adding a ProBindingSource control
- Adding a group box control
- Adding editors
- Adding a panel and buttons
- Adding event handlers
- Adding the interface methods
- Adding the data binding code
- Running purchaseOrderForm.cls

Overview

This chapter describes how to use the OpenEdge Visual Designer to build the Purchase Order form of the sample application.

The Purchase Order form, shown in the following figure, is the most complicated form in the sample application. It allows the user to navigate through purchase orders, change records, add new records, and delete records. It includes a number of editors that allow the user to enter appropriate data into various fields.

Figure 11: The Purchase Order form



Tasks in this chapter

When you create the Purchase Order form, you will:

- Group related controls on a form
- Resize and align controls
- Use various editor controls to implement user-input and data-display fields
- Implement buttons that subscribe to various Click events

Before you begin . . .

- Be sure that you have set up the SampleApp project, created the database, and started the database server as described in Setting Up the Sample Application Project.
- Start Progress Developer Studio for OpenEdge in the workspace that contains the SampleApp project.
- Verify that the SampleApp project AVM (runtime) has started and is connected to the sports2000 server.
- Open the OpenEdge Visual Designer perspective.

Adding purchaseOrderForm.cls to the project

The `purchaseOrderForm.cls` file is an ABL Form that implements the Department window in the application.

This section is a quick summary of how to add `purchaseOrderForm.cls` to the SampleApp project. The procedure is essentially the same as the one described in the Adding customerForm.cls to the project on page 23. That section provides more detailed information about the steps in this section.

To add purchaseOrderForm.cls to the project:

1. In the **Project Explorer** view, expand the **openedge** node of the SampleApp project.
2. Right-click on the **ui** node in the **tutorial** folder.
3. From the context menu, choose **New** > **ABL Form.**
4. In the **New ABL Form** wizard, enter `purchaseOrderForm` in the **Form name** field.
5. Click **Add** (next to the **Implements** field).
6. Select **IUpdatable -** `openedge.tutorial.ui` from the **Matching Interfaces** list in the **Interface Selection** dialog.
7. Click **OK**.
8. Click **Finish**.
9. Scroll down to the **Text** property in the **Properties** view, and change `purchaseOrderForm` to `Purchase Order`.

Adding a ProBindingSource control

This section summarizes most of the steps described in Data binding on page 25. When configuring the ProBindingSource control in this section, you import schema information from an XSD file, as you did when you created the Department form in Creating the Department Window on page 41.

To add a ProBindingSource control to the Purchase Order form:

1. With `purchaseOrderForm.cls` open in Visual Designer canvas, expand **OpenEdge Controls** in the Toolbox.
2. Drag **ProBindingSource** and drop it on the **Purchase Order** form.
3. When the **ProBindingSource Designer** appears, select the **Import Schema from File**  icon from the toolbar. The **Open** dialog appears.
4. Go to the `openedge\tutorial\services` folder in the SampleApp project and choose the `purchorder.xsd` file.
5. Click **Open**. The schema from `purchorder.xsd` file is added to the Available Schema pane in the ProBindingSource Designer dialog.
6. Select the tables and columns that you want add to your schema definition from the **Available Schema** pane and click **Add**.
7. Click **OK** in the **ProBindingSource Designer** to save your changes.

When you complete this procedure, you have a ProBindingSource control with the default name **bindingSource1**. This is the name that identifies the control in code. You can change the default name to something more meaningful by changing the **Name** property in the **Properties** view.

Adding a group box control

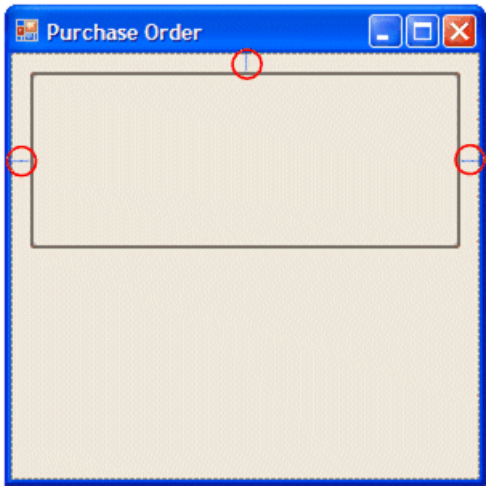
The first control add to the Purchase Order form is a container called an UltraGroupBox. This control is useful because it allows you to group a set of related controls. As you move the group box, all the controls within the group box move with it. In addition, you can make style choices in an UltraGroupBox that are inherited by the controls that it contains.

To add an UltraGroupBox to the Purchase Order form:

1. With `purchaseOrderForm.cls` open in Visual Designer, expand **OpenEdge Ultra Controls** in the OpenEdge Toolbox.
2. Select **UltraGroupBox**, then drag and drop it on the form.

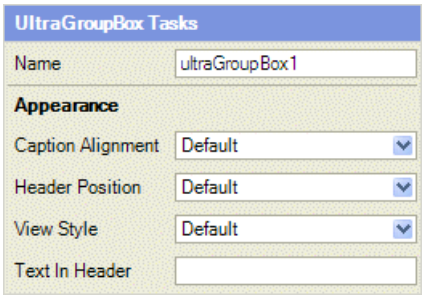
   The UltraGroupBox control, like the Panel control, is a container that you use to group controls. It has additional features, including headers, caption alignment, and a variety of view styles.

3. Use the shape handles to resize the UltraGroupBox, and the control to center it near the top of the form:



   Notice the alignment lines (circled in red) that can help you position the control. Look at the other positioning and alignment options available from the **Design** menu on the main menu. Also, check the layout preferences available from **Window** > **Preferences** > **Progress OpenEdge** > **Visual Designer**.

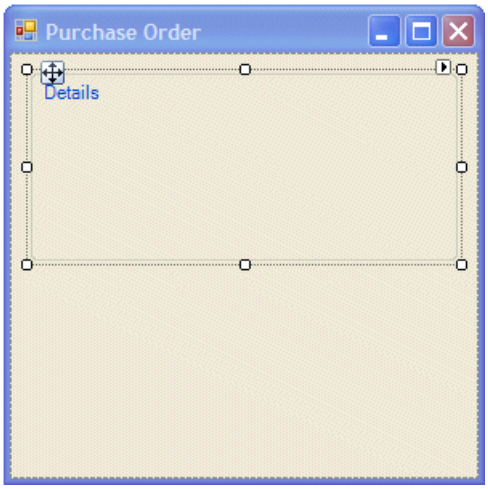
4. Click the Smart Tag arrow on the UltraGroupBox control. The Smart Tag panel appears:



5. Type `Details` in the **Text in Header** field.

6. Change **Header Position** to `TopInsideBorder`. (This is an arbitrary change, included simply to illustrate the options for placing the header. If you prefer, you can leave **Header Position** set to `Default`, which places the header on the border at the top right.)
7. Dismiss the Smart Tag panel by clicking outside of it.

   The form should now look something like the following:



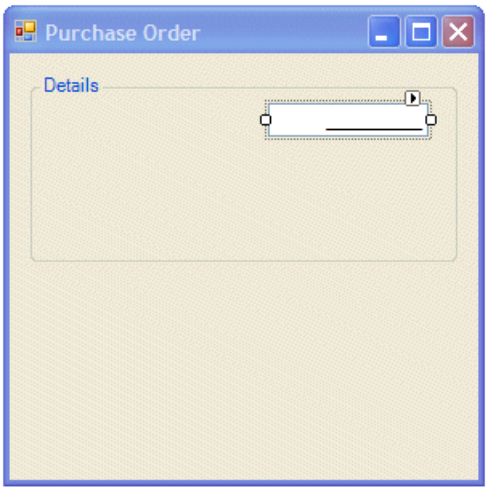
After you add some controls to the **Details** group box (see Adding editors on page 55), you can experiment with style inheritance. In the Smart Tag panel of the UltraGroupBox, you can change the values of the **View Style** property.

Adding editors

In this section, you will add several editors to the UltraGroupBox. Editors are user-input fields and/or display fields that you can customize to show particular data types in a particular format. After you add an editor to a form or a container, you usually add an UltraLabel to identify the editor's purpose. Finally, you align the controls with respect to each other. The Visual Designer supports most of the functionality for moving, aligning, and resizing objects that is standard on graphical editors on Windows platforms.

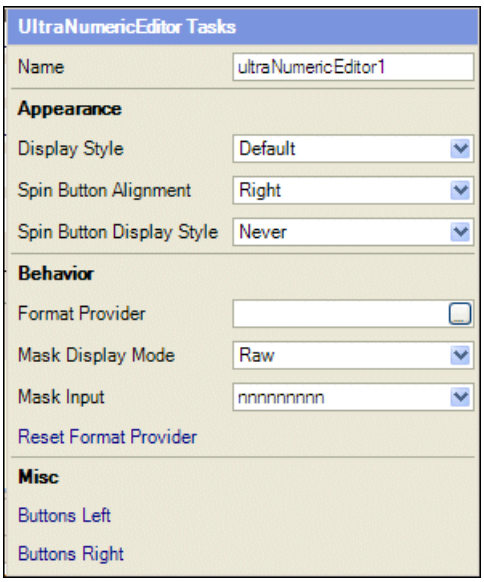
To add an editor to ultraGroupBox1:

1. With `purchaseOrderForm.cls` open in the Visual Designer canvas, expand **OpenEdge Ultra Controls** in the Toolbox.
2. Select **UltraNumericEditor**, then drag and drop it on the **Details** container:



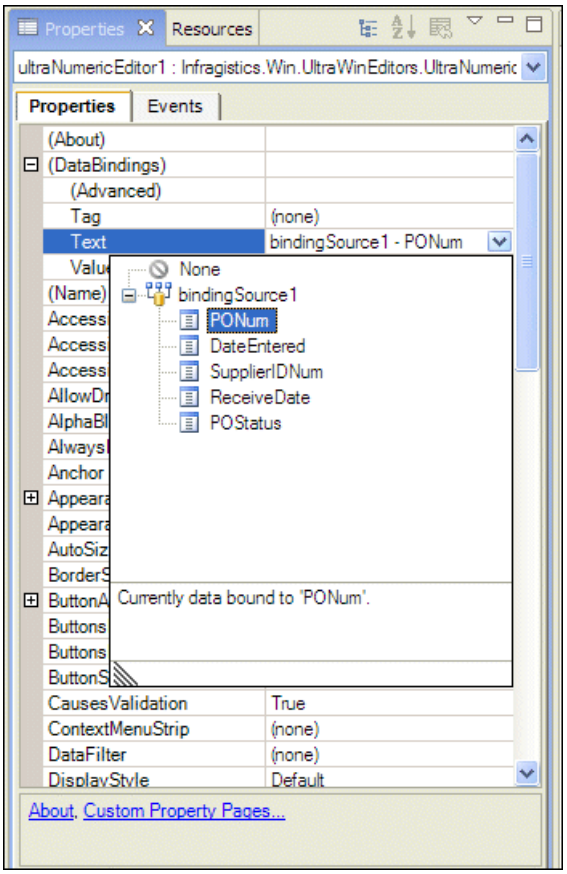
Note: The UltraNumericEditor control supports `INTEGER` and `INT64` data types. It also supports feature masking to format data input and display.

3. Open the Smart Tag panel by clicking the Smart Tag arrow on the UltraNumeric Editor control.
4. In the Smart Tag panel, change **Mask Display Mode** to `Raw` and **Mask Input** to `Integer 1` (`nnnnnnnnn`):



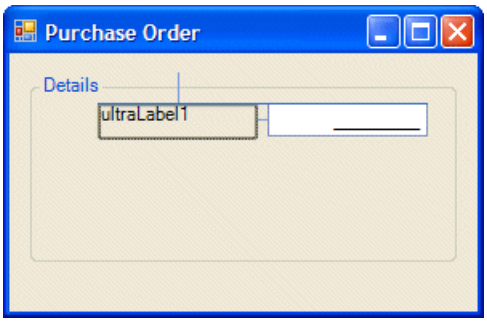
Note: In this tutorial, you leave the default names on controls (**ultraNumericEditor1**, **ultraNumericEditor2**, etc.). You can easily give controls more meaningful names by changing the **Name** property. However, do not change the default names in these exercises because the code for the sample application depends on the defaults.

5. Dismiss the Smart Tag panel by clicking outside of it.
6. With **ultraNumericEditor1** selected, find and expand **(DataBindings)** in the **Properties** view.
7. Under **bindingSource1**, choose **PONum**, from the drop-down menu in the **Text** field as shown in the following illustration:

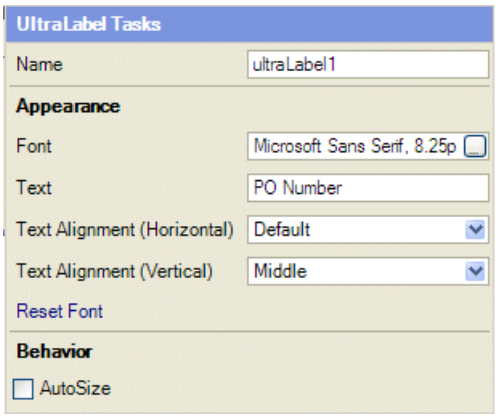


This step binds the control to the correct field in the ProBindingSource.

8. From the **OpenEdge Ultra Controls** control group in the Toolbox, select **UltraLabel1** and drop it to the left of the ultraNumericEditor1 control:

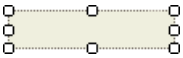


9. In the Smart Tag panel of ultraLabel1, change **Text** to `PO Number` and **Text Alignment (Vertical)** to `Middle`:



10. Change the size of UltraLabel1 and UltraNumericEditor1, if necessary.

    When selected, objects on the Design Canvas display the shape handles that are commonly used in graphical editors to adjust the size of an object. Shape handles are squares distributed along the perimeter of a graphical object, as shown in the following illustration:

    When you hover over a shape handle, an arrow appears. The arrow indicates the direction that you can expand or shrink a shape.

11. Select both ultraLabel1 and ultraNumericEditor1 on the Design Canvas. You can do this by clicking on UltraLabel1, and then **CTRL**-clicking on UltraNumericEditor1. Or, you can place the cursor above and to the left of the two controls, hold down the primary mouse button and draw a marquee around the two controls.
12. With both controls selected, choose **Design** > **Align** > **Vertical Centers** from the main menu bar. This step centers the text of ultraLabel1 with the rectangle created by ultraNumericEditor1.
13. Save `purchaseOrderForm.cls`.

As you add more controls to the Purchase Order form, you will probably need to adjust the size and alignment of the controls within the form. Use your own judgment to give the user interface an uncluttered, balanced, and attractive look. You can also experiment with the various properties that are available for each control, such as appearance, color, font, and style.

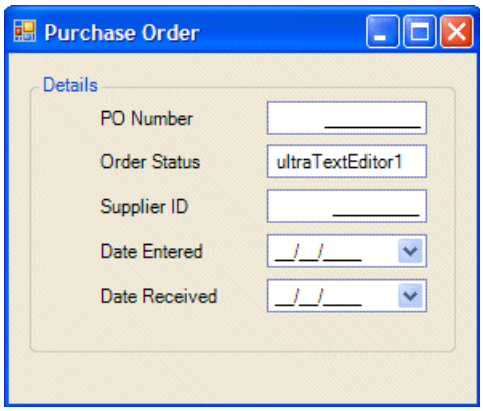
The following table lists the remaining editors and labels to add to ultraGroupBox1.

Table 1: Remaining editors for the Details section

Editor	(Databinding)/Textproperty	MaskInput property	ultraLabel Text property
UltraTextEditor1	POStatus	NA	Order Status
UltraNumericEditor2	SupplierIDNum	Integer 1	Supplier ID
UltraDateTimeEditor1	DateEntered	Date 1	Date Entered
UltraDateTimeEditor2	ReceiveDate	Date 1	Date Received

After you add and align all the editors and labels, the **Details** section of the Purchase Order form should look similar to the following figure.

Figure 12: The Details section of the Purchase Order form



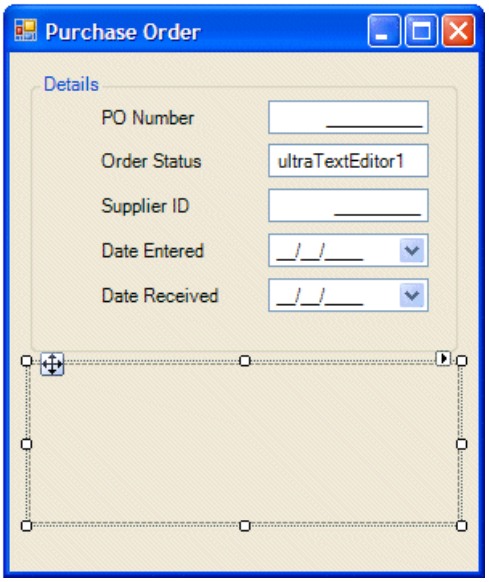
Adding a panel and buttons

In this section, you will add buttons to a panel. A panel is similar to a group box. Like the UltraGroupBox control, the Panel control is a container that allows you to group a number of controls together. Unlike the UltraGroupBox, the Panel control does not have a border or a label. Also note that the controls contained in a panel do not inherit a style from the panel.

There are two sets of buttons: four for navigation and four for database CRUD (create, read, update, and delete) activities.

To add buttons to the purchaseOrderForm:

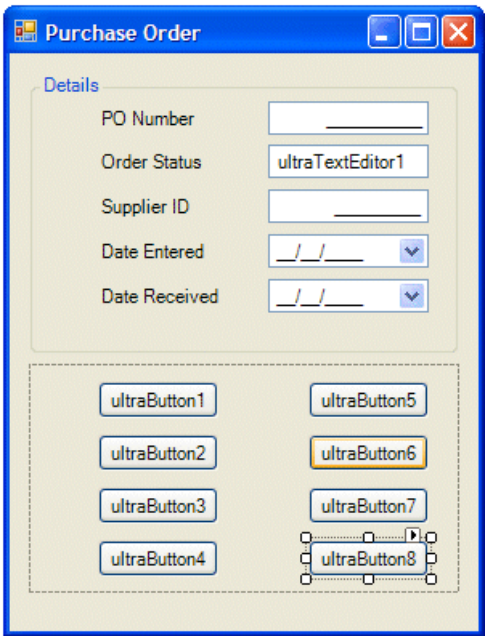
1. With `purchaseOrderForm.cls` open in the Visual Designer, open the **Microsoft** control group in the Toolbox.
2. Select **Panel**, then drag and drop it below the **Details** section of the Purchase Order form as shown in the following illustration:



3. Expand **OpenEdge Ultra Controls** in the Toolbox.
4. Drag and drop eight UltraButtons on the panel container.

   You can create two vertical rows with four buttons in each row. Then you can use the tools under the Design menu to adjust the alignment and spacing of the buttons with respect to each other and with respect to the panel container. The Design menu is on the main menu bar and is also available from the context menu when you right-click on a selected control.

   After you complete this step, the Purchase Order form should look something like the following:



5. Set the properties of the buttons as specified in the following table.

Note: In most of the procedures in this book, the default names for controls are acceptable. For the buttons, however, you must change the Name property for the event handler code to work.

Table 2: Name and Text properties of Purchase Order buttons

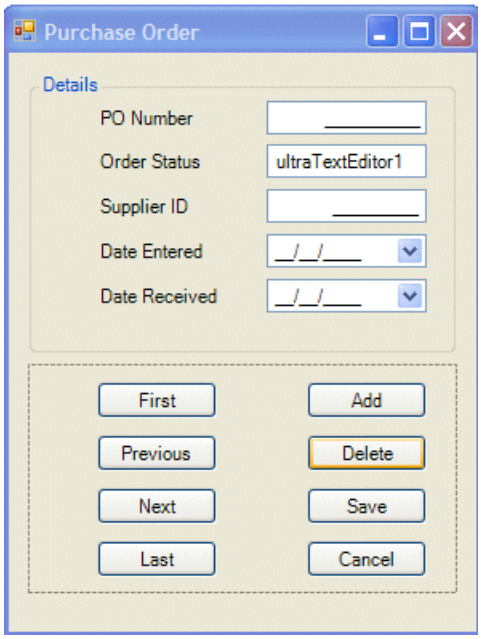
Default Name	Name property	Text property
ultraButton1	btnFirst	First
ultraButton2	btnPrev	Previous
ultraButton3	btnNext	Next
ultraButton4	btnLast	Last
ultraButton5	btnAdd	Add
ultraButton6	btnDelete	Delete

Default Name	Name property	Text property
ultraButton7	btnSave	Save
ultraButton8	btnCancel	Cancel

You can set these properties on the Smart Tag panel for each button. Notice the Smart Tag has a field **Button Style**, where you can experiment with the style of the buttons.

When you complete the steps in this section, the graphical elements of the Purchase Order form are in place. The result in the Visual Designer should be similar to the following figure.

Figure 13: The completed Purchase Order form



Adding event handlers

In this section you will add:

- Methods that execute when you click a button
- The method that executes when you scroll from one record to another in the data source

After adding the code, you will subscribe each button and the ProBindingSource to the appropriate method.

To add event handlers to purchaseOrderForm.cls:

1. Open `purchaseOrderForm.cls` in the ABL Editor. (When a form is open in the Visual Designer canvas, you can open it in the ABL Editor by pressing **F9** or by choosing **View Source** in the context menu.)
2. Add event handler methods to `purchaseOrderForm.cls`. Copy and paste the following code before the `DESTRUCTOR` method to implement the event handler for **bindingSource1**:

```
METHOD PRIVATE  VOID OnPositionChanged (sender AS System.Object,
e AS System.EventArgs ):
  DEF VAR pbs AS Progress.Data.BindingSource NO-UNDO.
     pbs = cast(sender, 'Progress.Data.BindingSource').
     btnPrev:Enabled = pbs:Position GT 0 .    /*Prev*/
     btnFirst:Enabled = btnPrev:enabled.
     btnNext:Enabled = pbs:Position LT (pbs:Count - 1).    /*Next*/
     btnLast:Enabled = btnNext:Enabled.

     DELETE OBJECT e NO-ERROR.
END METHOD.
```

   Copy and paste the following code before the `END CLASS` statement to implement the event handlers for the buttons:

```
METHOD PRIVATE VOID btnFirst_Click (sender AS System.Object,
    e AS System.EventArgs ):
    bindingSource1:Position = 0.
END METHOD.
METHOD PRIVATE VOID btnNext_Click (sender AS System.Object,
    e AS System.EventArgs ):
    bindingSource1:Position = bindingSource1:Position + 1.
END METHOD.
METHOD PRIVATE VOID btnPrev_Click (sender AS System.Object,
    e AS System.EventArgs ):
    bindingSource1:Position = bindingSource1:Position - 1.
END METHOD.
METHOD PRIVATE VOID btnLast_Click (sender AS System.Object,
    e AS System.EventArgs ):
    bindingSource1:Position = bindingSource1:Count - 1.
END METHOD.
METHOD PRIVATE VOID btnSave_Click (sender AS System.Object,
    e AS System.EventArgs ):
    THIS-OBJECT:SaveRecord().
END METHOD.
METHOD PRIVATE VOID btnAdd_Click (sender AS System.Object,
    e AS System.EventArgs ):
    THIS-OBJECT:AddRecord().
END METHOD.
METHOD PRIVATE VOID btnDelete_Click (sender AS System.Object,
    e AS System.EventArgs ):
    THIS-OBJECT:DeleteRecord().
END METHOD.
 METHOD PRIVATE VOID btnCancel_Click (sender AS System.Object,
    e AS System.EventArgs ):
    THIS-OBJECT:CancelUpdate().
END METHOD.
```

   Notice that some of these methods call the methods that are prototyped in the interface file `IUpdatable.cls`. You will implement these interface methods in Adding the interface methods on page 63.

3. Save the file and return to the Visual Designer.
4. Select each button and change the **Click** event in the **Events** tab of the **Properties** view as follows:

Button name	Click value
btnFirst	btnFirst_Click
btnPrev	btnPrev_Click
btnNext	btnNext_Click
btnLast	btnLast_Click
btnAdd	btnAdd_Click
btnDelete	btnDelete_Click
btnSave	btnSave_Click
btnCancel	btnCancel_Click

Note: You can double-click on a control to subscribe to the control's default event. The ABL Editor appears in a template where you can complete the code that runs when the event occurs. However, in this exercise, in which you copied and pasted pre-existing code, it is easier to subscribe to the events for each control in the Properties view after the code is pasted in the file.

5. Select the **bindingSource1** control on the Design Canvas.
6. Add `onPositionChanged` to the **PositionChanged** event in the **Events** tab of the **Properties** view.

Adding the interface methods

In this section you will implement the interface methods prototyped in `IUpdatable.cls`.

To add the interface methods:

1. Add a variable definition for fRecordState and fServiceAdapater with the other variable definitions that appear after the opening CLASS statement. The variable definition appears in bold in the following example:

```
CLASS openedge.tutorial.ui.purchaseOrderForm INHERITS Form IMPLEMENTS
IUpdatable  :
    DEF PRIVATE VAR fRecordState AS CHARACTER NO-UNDO.
    DEF PRIVATE VAR components AS System.ComponentModel.IContainer.
    DEF PRIVATE VAR fServiceAdapter AS
openedge.tutorial.services.ServiceAdapter NO-UNDO.
```

2. With purchaseOrderForm.cls open in the ABL Editor, replace the addRecord(), deleteRecord(), saveRecord(), and cancelUpdate() methods with the following code:

```
METHOD PUBLIC VOID AddRecord():
        fServiceAdapter:addRecord
        (bindingSource1:Handle:get-buffer-handle('ePurchaseOrder')).
        fRecordState = 'ADD':u.
        btnSave:Enabled = TRUE.
        btnCancel:Enabled = TRUE.
END METHOD.
METHOD PUBLIC VOID DeleteRecord():
        fServiceAdapter:removeRecord
        (bindingSource1:Handle:get-buffer-handle('ePurchaseOrder')).
        fServiceAdapter:SaveData().
        bindingSource1:RefreshAll().
END METHOD.
METHOD PUBLIC VOID SaveRecord():
        IF fRecordState NE '':u OR bindingSource1:RowModified THEN
        DO:
           IF bindingSource1:RowModified THEN
            bindingSource1:Assign().
            fServiceAdapter:SaveData().
            btnSave:Enabled = FALSE.
            btnCancel:Enabled = FALSE.
        END.
END METHOD.
METHOD PUBLIC VOID CancelUpdate():
       fRecordState = '':u.
        fServiceAdapter:CancelUpdate
       (bindingSource1:Handle:get-buffer-handle('ePurchaseOrder')).
       bindingSource1:RefreshAll().
       btnSave:Enabled = FALSE.
       btnCancel:Enabled = FALSE.
END METHOD.
```

This step replaces the stub code for the interface methods defined in IUpdateable.cls. Stub code for interface methods is automatically generated. This allows class files that implement interfaces to compile even though you have not implemented the interface methods.

Adding the data binding code

The data binding code includes open and close query methods, updates to the CONSTRUCTOR and DESTRUCTOR, and also a variable definition for the service adapter.

To add the data binding code:

1. Add the open and close query methods by copying and pasting the following code before the `DESTRUCTOR` statement:

```
METHOD PRIVATE VOID openQuery():
bindingSource1:handle = fServiceAdapter:getDatasetHandle().
btnFirst:Enabled = FALSE.
btnPrev:Enabled = FALSE.
btnNext:Enabled = TRUE.
btnLast:Enabled = TRUE.
btnSave:Enabled = FALSE.
btnCancel:Enabled = FALSE.
END METHOD.
METHOD PRIVATE VOID closeQuery():
fServiceAdapter:closeQuery().
END METHOD.
```

2. Update the existing `CONSTRUCTOR` and `DESTRUCTOR` methods by adding the code shown in bold:

```
CONSTRUCTOR PUBLIC  purchaseOrderForm (  ):
SUPER().
fServiceAdapter = NEW openedge.tutorial.services.ServiceAdapter("PurchOrder").
InitializeComponent ( ).
OpenQuery().
CATCH e AS Progress.Lang.Error:
    UNDO, THROW e.
END CATCH.
END CONSTRUCTOR.

DESTRUCTOR PUBLIC purchaseOrderForm ( ):
closeQuery().
IF VALID-OBJECT(components) THEN DO:
CAST(components, System.IDisposable):Dispose().
END.

END DESTRUCTOR.
```

3. Save the file.

Running purchaseOrderForm.cls

To run purchaseOrderForm.cls:

1. Save `purchaseOrderForm.cls`, if necessary.
2. Select `purchaseOrderForm.cls` in the **Project Explorer** view.
3. Select **Run** > **Run As** > **Progress OpenEdge Application** on the main menu bar.

When the Purchase Order form runs, you can check its functionality, which includes:

- Displaying data from the first record of the `PurchaseOrder` table of the sports2000 database in the **Details** fields.
- Navigating through the records using the **First**, **Previous**, **Next**, and **Last** buttons.
- Creating a new record by clicking **Add**, and entering data. Click **Save (**or **Cancel** if you do not want to save**)** after entering data.

Notice that the **Date Entered** and the **Date Received** fields have drop-down calendars that allow you to click on a date.

- Deleting a record by navigating to it and clicking **Delete**.

5

Creating the Login Dialog

This chapter describes how to use Visual Designer to build the Login dialog for the sample application.

For details, see the following topics:

- Overview
- Adding LeftBar.cls to the project
- Adding LoginBlock.cls to the Toolbox
- Adding HelpButton.cls to the Toolbox
- Adding LoginDlg.cls to the project
- Running LoginDlg.cls
- Where to go from here

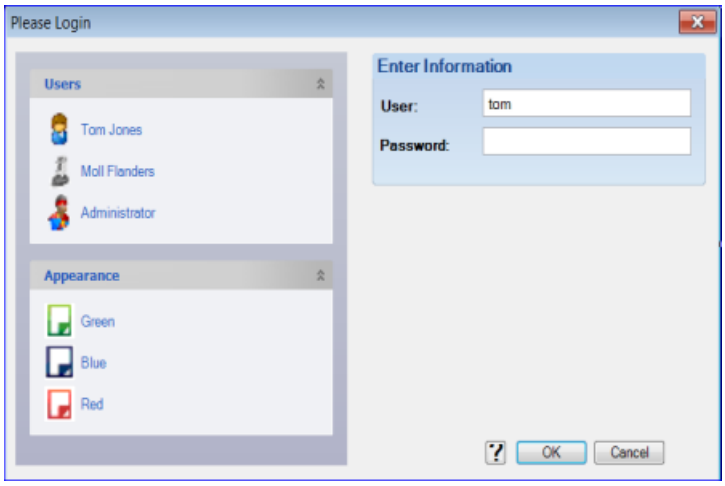
Overview

This chapter describes how to use Visual Designer to build the Login dialog for the sample application.

The Login dialog, shown in the following figure, is the first form users see when they start the sample application. When users enter valid identification information in the form, the Login dialog closes and the Sports form opens.

This is a simplified version of a real login screen. User ID and password authentication is simulated. The focus is on the GUI elements of the dialog and not the security functionality that supports user authentication in real-world applications.

Figure 14: The Login dialog



Tasks in this chapter

When you create the Login form, you will:

- Implement and deploy an ABL Dialog
- Implement and deploy ABL User Controls
- Deploy an ABL Inherited Control
- Use an UltraExplorerBar

Before you begin . . .

- Be sure that you have set up the SampleApp project, created the database, and started the database server as described in Setting Up the Sample Application Project.
- Start Progress Developer Studio for OpenEdge in the workspace that contains the SampleApp project.
- Verify that the SampleApp AVM has started and is connected to the sports2000 server.
- Open the OpenEdge Visual Designer perspective.

Adding LeftBar.cls to the project

One or more of the .NET controls in the Progress Developer Studio for OpenEdge Toolbox can be encapsulated into a reusable unit called the ABL User Control. ABL User Controls are sometimes referred to as composite controls because they are groupings of existing controls. They are useful when you have a specific grouping of controls that is repeated in your application. With an ABL User Control, you can drag and drop the same group on any number of forms in your project.

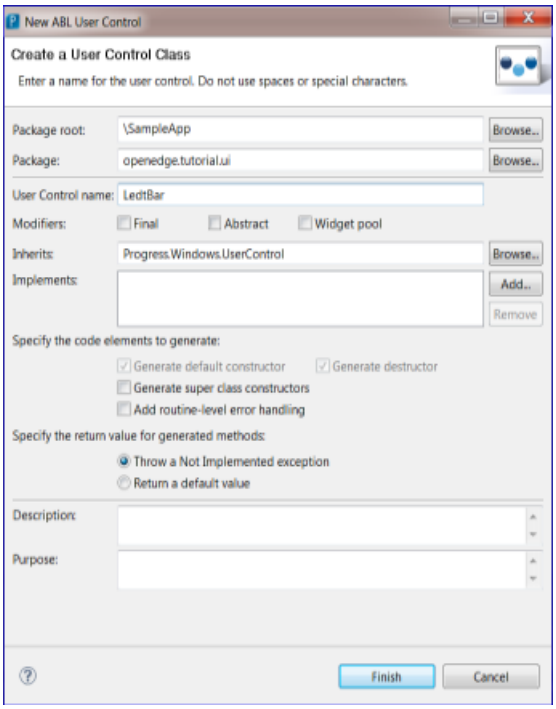
Like the other forms in the Visual Designer, the ABL User Control is a selection on the **File** > **New** menu. After you add controls to the form (and define their properties, data bindings, event logic, etc.), you add the user control to the Toolbox. The user control is then available to be dragged and dropped on other ABL forms in the project.

Note: There is another type of custom control supported by the Visual Designer—the inherited control. Inherited controls are described in Adding HelpButton.cls to the Toolbox on page 79.

`Leftbar.cls` is one of the building blocks of the login dialog. It is an ABL User Control that allows you to choose a user and set the color scheme for the application.

add LeftBar.cls to the project:

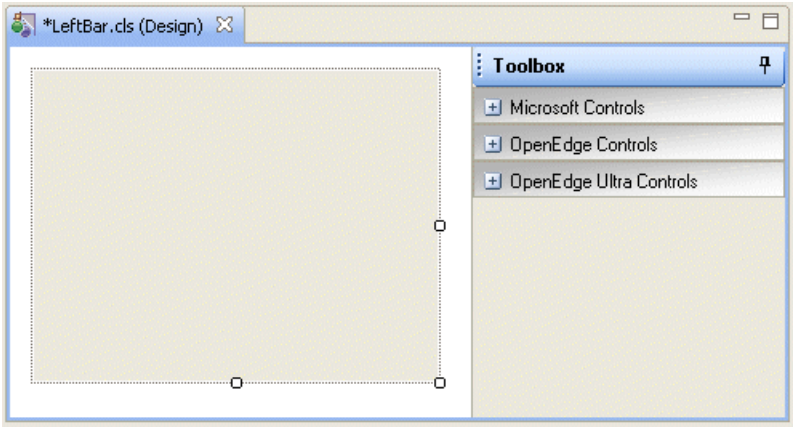
1. Expand the `openedge` node in the SampleApp project.
2. Right-click on the `ui` node.
3. Choose **New** > **ABL User Control**.
4. In the New ABL User Control wizard, type `LeftBar` in the **User Control name** field as shown in the following illustration:



   Notice that the **Inherits** field indicates that the control, `LeftBar`, is an instance of the `Progress.Windows.UserControl` class.

5. Click **Finish**.

`LeftBar.cls` appears with a resizable container displayed in the Visual Designer Canvas as shown in the following illustration:

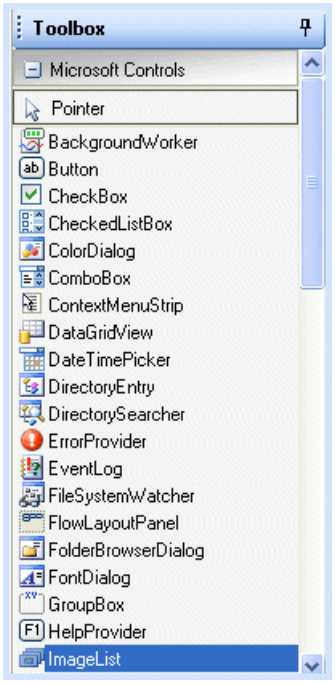


Adding an image list to LeftBar.cls

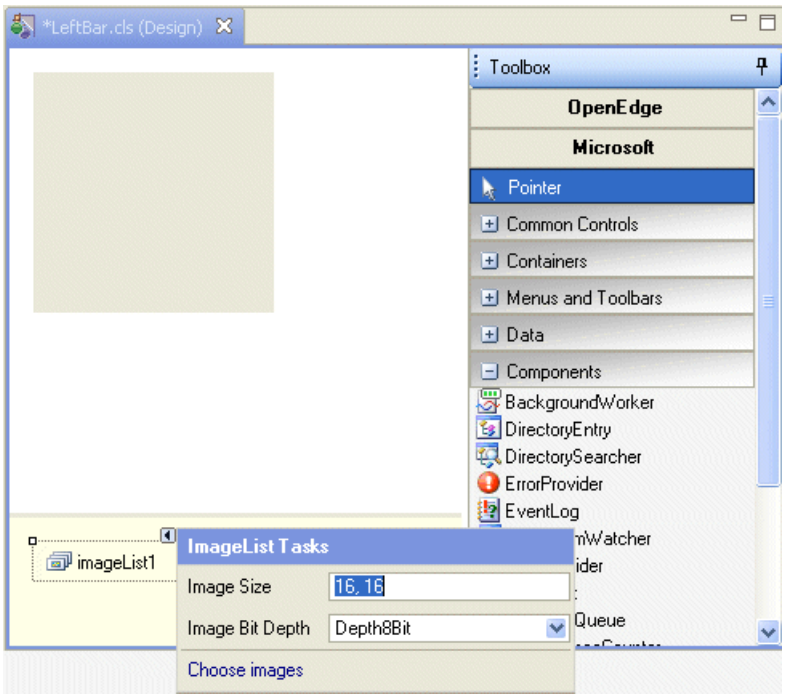
An image list is a non-visual control that manages icons and other graphics, making them available to other controls in a container.

To add an image list:

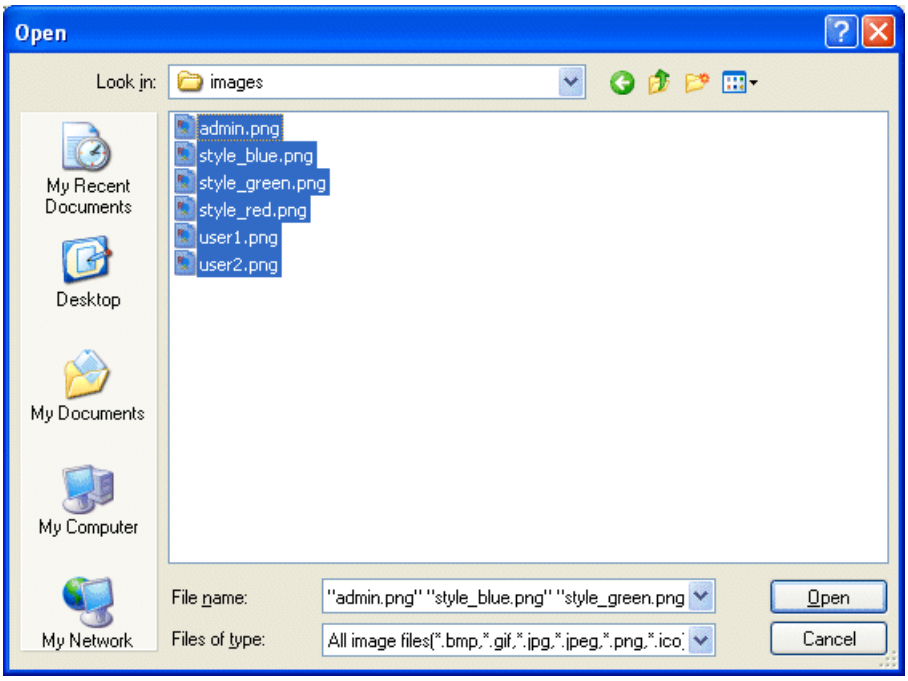
1. In the Toolbox, expand **Microsoft Controls** as shown in the following illustration:



2. Select **ImageList** and drop it on LeftBar. The control, **imageList1**, appears in the non-visual control tray.
3. Click the Smart Tag arrow on **imageList1** to display the **ImageList Tasks** dialog, as shown in the following illustration:



4. Select **Choose images**. The **Images Collection Editor** appears.
5. Click **Add**.
6. Navigate to the `images` folder in the SampleApp project.
7. Select all the image files and click **Open**, as shown in the following illustration:



8. Click **OK** in the Images Collection Editor.

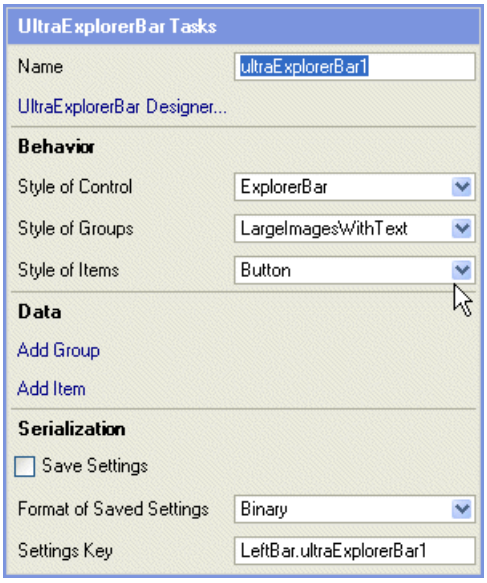
   The images that you added to **imageList1** are icons and are now available for use in `ListBar.cls`.

Adding an UltraExplorerBar to LeftBar.cls

The UltraExplorerBar is a versatile control that allows you to implement a number of navigation items, styles, and groups. When you use an UltraExplorerBar, you might not need to add any additional buttons, labels, or other controls.

To add an UltraExplorerBar:

1. In the Toolbox, expand **OpenEdge Ultra Controls**.
2. Select **UltraExplorerBar** and drop it on LeftBar.
3. Select the **Dock** property in the **Properties** view. From the drop-down menu, select the middle option, **Fill**, which binds the control to the four borders of the form.
4. Click the Smart Tag arrow on ultraExplorerBar1 to display the **UltraExplorerBar Tasks** dialog, as shown in the following illustration:



5. Set the behavior settings listed in the following table:

Behavior setting	Value
Style of Control	ExplorerBar
Style of Groups	LargeImagesWithText
Style of Items	Button

These style properties set the general look and feel of the control, groups and items. After you complete this section, you can return to this step and experiment with other style properties.

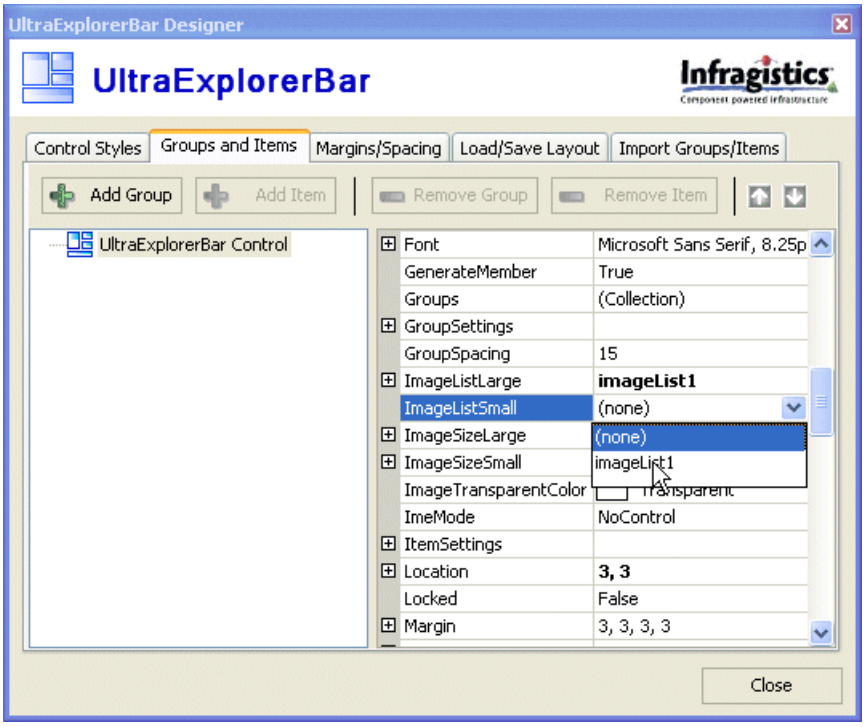
6. With **ultraExplorerBar1** selected, right click and choose **UltraExplorerBar Designer**.

The UltraExplorerBar Designer is an interface for setting many of the properties of the UltraExplorerBar control. In this exercise you specify the source for icon images (imageList1) and you can create two functional groups (Users and Appearance). The items in each group appear as buttons, because you set the **Style of Items** property to **Button** in the last step.

7. Select the **Groups and Items** tab.

   The Groups and Items interface appears. The root node of the element is shown in a tree view on the left. The property sheet on the right displays properties that apply to this top-level node (i.e, the element itself). Buttons at the top of the tab provide the ability to add Groups and Items.

8. Set **ImageListSmall** and **ImageListLarge** properties to **imageList1**, as shown in the following illustration:



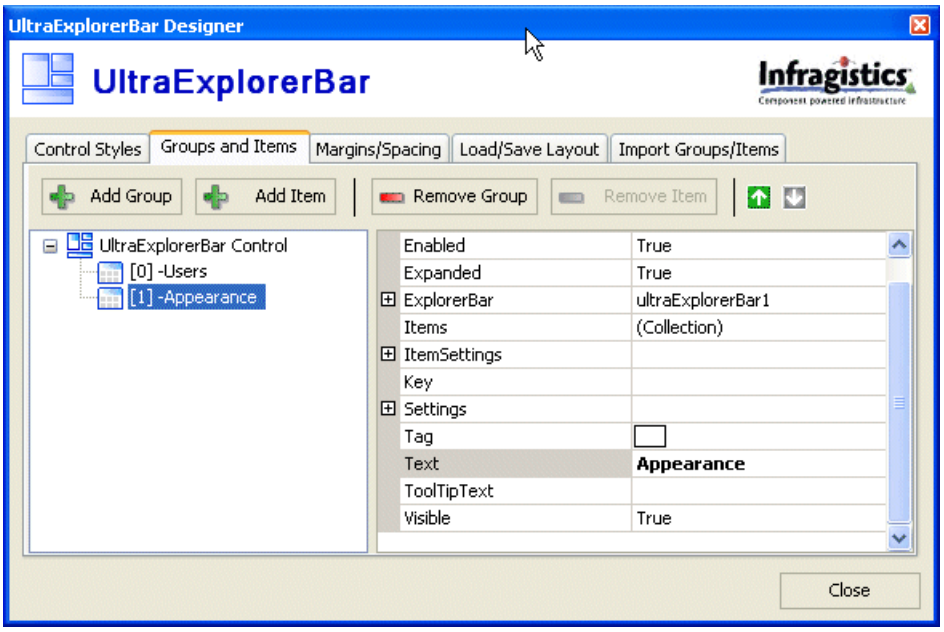
   This step makes the graphics files managed by **imageList1** (the files located in the `SampleApp/images` folder) available to **ultraExplorerBar1**.

9. Click **Add Group** twice.

   Two new groups appear in the tree view. When a group is selected, the property sheet changes to reflect the available properties for groups.

10. Rename the two new groups **Users** and **Appearance**.

    To rename a group, set the Text property of the group, as shown in the following illustration:



11. Select the **Users** group and check the **Expanded** property.

    By default, it is set to **True**. This causes a group to appear in the expanded state, where all items are visible at startup. If set to **False**, the group appears in a collapsed state, where all items are initially hidden

12. With **Users** group selected, click **Add Item** three times.

    The three items will behave like buttons (because the Item Style property was set to **Button** in step 5 on page 72) that supply a user name and password to another user control, `LoginBlock.cls`.

    Remember that this SampleApp is meant to show some of the capabilities of Visual Designer and .NET controls. It is not meant to be a model for real-world solutions. For example, you do not see many login dialogs where you can click a button and get valid user names and passwords. `LeftBar.cls` only simulates the processing of user names and passwords in order to avoid getting entangled in login and security issues.

13. Modify the Users group items by setting the properties shown in the following table:

Item	Property			
	Key	**Text**	**Expand the node Settings > AppearancesLarge > Appearance > Image**	**Expand the node Settings > AppearancesSmall > Appearance > Image**
[0]	tom	Tom Jones	user1.png	user1.png
[1]	moll	Moll Flanders	user2.png	user2.png
[2]	admin	Administrator	admin.png	admin.png

    The value in the key field is passed as a parameter to an event handler when one of the items is selected. See Subscribing to events and adding handlers on page 76 for more information. The value is used in the **User** and **Password** fields that are implemented in Adding LoginBlock.cls to the Toolbox on page 79.

14. Select the **Appearance** group and verify that the **Expanded** property is set to **True**.

    The Appearance group contains buttons that change the color scheme of the windows in the sample application.

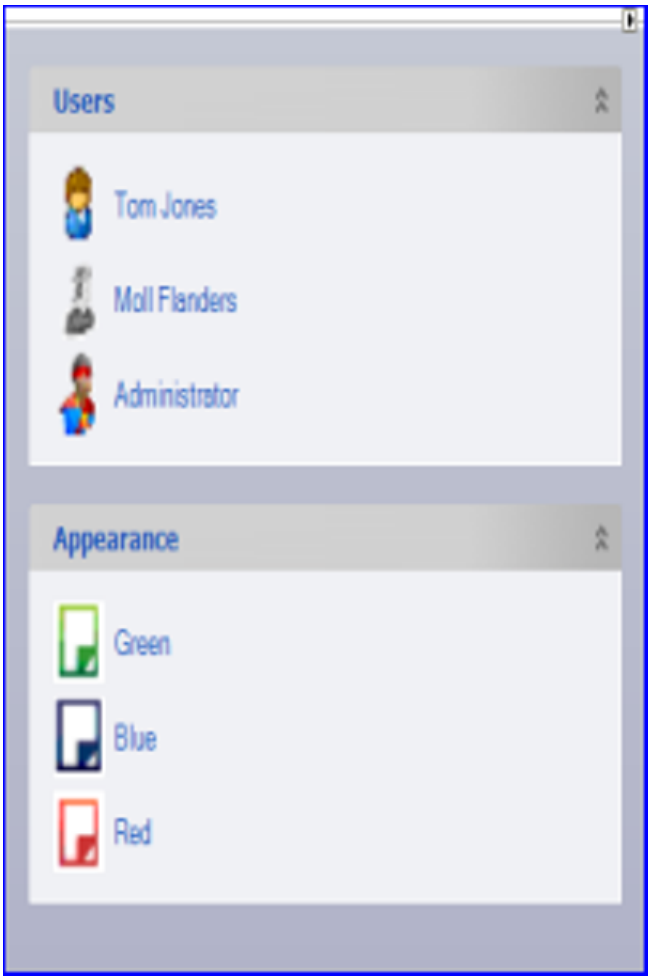
15. With **Appearance** group selected, click **Add Item** three times.
16. Modify the Appearance group items by setting the properties shown in the following table:

Item	Property		
	Key	**Text**	**Settings > AppearancesLarge > Appearance > Image**
[0]	styles\FlatNature.isl	Green	style_green.png
[1]	styles\TheBlues.isl	Blue	style_blue.png
[2]	styles\Red_No_pill.isl	Red	style_red.png

The value in the key field is passed as a parameter to an event handler when one of the items is selected. See Subscribing to events and adding handlers on page 76 for more information. The value is a path to a style library that is used by the AppStylistRuntime control to apply a skin to the application. AppStylistRuntime is a non-visual control that is added in Adding LoginDlg.cls to the project on page 80.

At this point, `LeftBar.cls` should look something like the following figure.

Figure 15: The LeftBar.cls form



Subscribing to events and adding handlers

LeftBar.cls requires event handlers to respond when a group is expanded and when an item is chosen. In this section you add the code for, and subscribe to ultraexplorerbar1_GroupExpanding and ultraexplorerbar1_ActiveItemChanged, which respond to group expansion and item selection.

To add GroupExpanding and ActiveItemChanged to LeftBar.cls:

1. Open LeftBar.cls in the ABL Editor.

   If you have LeftBar.cls open in the Visual Designer, press **F9** to open it in the ABL Editor.

2. Add the following to the USING statements near the top of the file:

```
USING openedge.tutorial.ui.*.
USING Infragistics.Win.UltraWinExplorerBar.*.
```

   These statements are necessary so that the class and interface references in the event handlers are valid.

3. Add the following code before the END CLASS statement to implement the event handlers:

```
@VisualDesigner.
METHOD PRIVATE VOID ultraexplorerbar1_ActiveItemChanged
  (INPUT sender AS System.Object,
  INPUT e AS Infragistics.Win.UltraWinExplorerBar.ItemEventArgs):
   IF e:Item:Group:Index = 0 THEN
   CAST(THIS-OBJECT:ParentForm, IUser):SetUser( e:Item:Key, e:Item:Key).
   ELSE DO:
     DEF VAR styleFile AS System.IO.FileInfo.
     styleFile = NEW System.IO.FileInfo( e:Item:Key ).
     CAST(THIS-OBJECT:ParentForm, IStyling):SetStyle( styleFile ).
   END.
END METHOD.

@VisualDesigner.
METHOD PRIVATE VOID ultraExplorerBar1_GroupExpanding
  (INPUT sender AS System.Object, INPUT e AS
  Infragistics.Win.UltraWinExplorerBar.CancelableGroupEventArgs ):

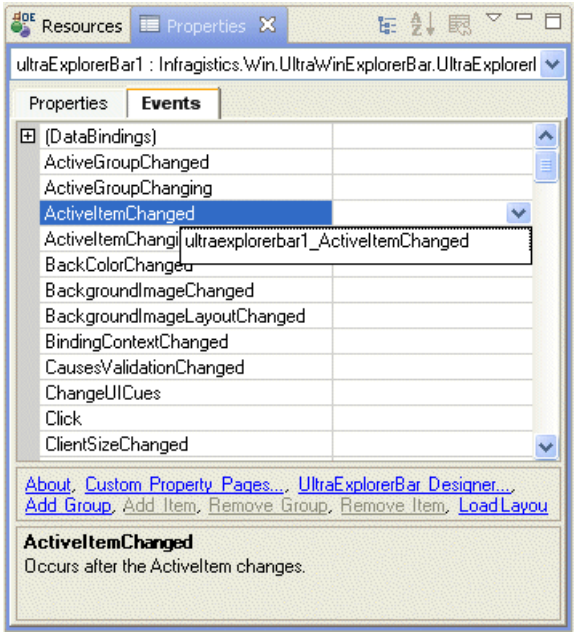
   DEFINE VARIABLE iLoop as integer no-undo.
   DEFINE VARIABLE grp as UltraExplorerBarGroup no-undo.

   DO iLoop = 0 to ultraExplorerBar1:Groups:Count - 1:
    grp = cast(ultraExplorerBar1:Groups:GetItem(iLoop),     UltraExplorerBarGroup).

     /* Must use the Equals() method for objects */
     IF NOT grp:Equals(e:Group) THEN grp:Expanded = False.
   END.
   RETURN.
END METHOD.
```

4. In the Visual Designer, select **ultraExplorerBar1**.
5. Open the **Events** tab in the **Properties** view.
6. Set the **ActiveItemChanged** event to ultraExplorerBar1_ActiveItemChanged.

   A list of the event handlers that you can subscribe to appears in a drop-down menu, as shown in the following illustration:



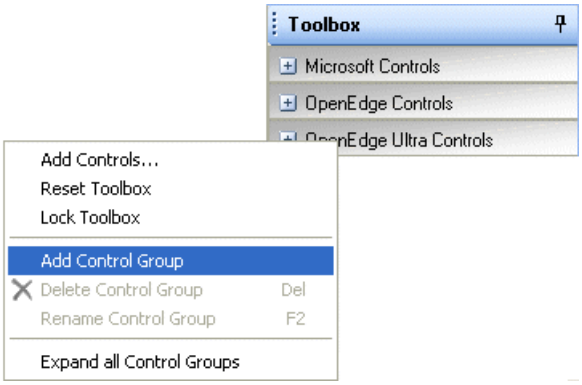
7. Set **GroupExpanding** event to `ultraExplorerBar1_GroupExpanding`.
8. Save and close `LeftBar.cls`.

Adding LeftBar.cls to the Toolbox

When you create a custom control, you must add it to the Toolbox so it will be available to drag and drop on other UI elements in your project.

add Leftbar.cls to the Toolbox:

1. Right-click on **one of** control groups in the Toolbox.
2. Select **Add Control Group**, as shown in the following illustration:



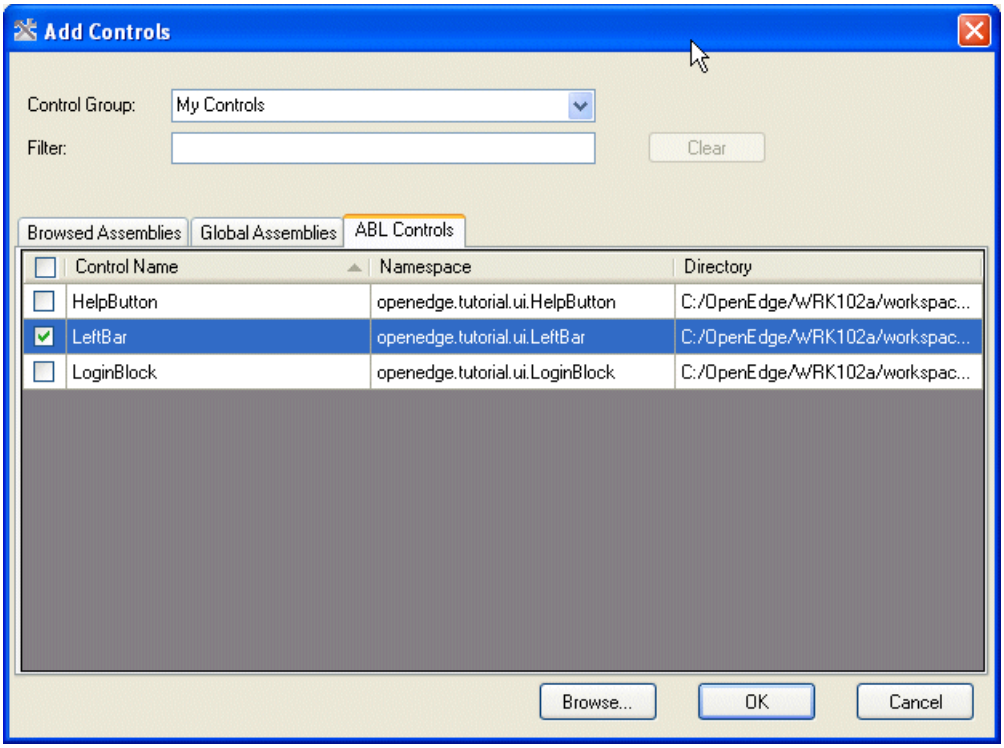
A blank control group title bar appears in the Toolbox.

Note: Although it can be convenient to put your user controls in a separate control group, it is not mandatory. You can add a user control to one of the existing control groups.

3. Give the new control group a name.

   We will call the new control group **My Controls**.

4. Right-click on **My Controls** and select **Add Controls**. The Add Controls dialog appears.
5. Select the **ABL Controls** tab.
6. Choose the **LeftBar** control, as shown in the following illustration:



Note: Notice that in addition to **Leftbar**, **HelpButton** and **LoginBlock** are also listed in the **ABL Controls** tab. The tab automatically lists all the custom controls in the project that are not in the Toolbox. **HelpButton** and **LoginBlock** are custom controls that are among the pre-existing resources in the SampleApp project. See the Adding LoginBlock.cls to the Toolbox on page 79 and Adding HelpButton.cls to the Toolbox on page 79 for more information.

7. Click **OK**.

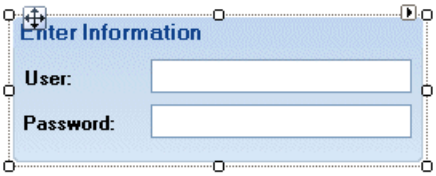
   LeftBar is added to the Toolbox in the **My Controls** group.

Note: **LeftBar** in the Toolbox is not updated if you make future changes to `LeftBar.cls`. To update **LeftBar**, you must remove it and re-import it to the Toolbox.

8. Save and close `LeftBar.cls`.

Adding LoginBlock.cls to the Toolbox

LoginBlock is another ABL user control that you use when building the login dialog for the sample application. It looks like a standard login prompt for user name and password:



However, it differs from a real-world login because it accepts a valid user name and password when a user item is selected in the LeftBar control.

We included `LoginBlock.cls` as one of the pre-existing resources in the SampleApp project. You do not need to create it. However, you can open `LoginBlock.cls` in the Visual Designer or the ABL Editor to see how it is constructed and to examine the properties of its component controls.

Basically, `LoginBlock.cls` is an ABL User Control containing an UltraGroupBox control, which is a container that applies a common style to the controls you drop on it. In this case, the UltraGroupBox has two UltraWinEditors identified by two UltraLabels. The UltraWinEditors will receive the user name and password from the LeftBar control when you run the login dialog in Running LoginDlg.cls on page 85.

Before you can use `LoginBlock.cls` in the project, you must add it to the Toolbox.

To add LoginBlock to the Toolbox:

1. Right-click on **My Controls** and select **Add Controls.** The **Add Controls** dialog appears.
2. Select the **ABL Controls** tab.
3. Choose the **LoginBlock** control.
4. Click **OK**. **LoginBlock** appears under **My Controls**.
5. Close `LoginBlock.cls` if it is open.

Adding HelpButton.cls to the Toolbox

In addition to user controls, Visual Designer also supports inherited controls. Where a user control is a grouping of existing controls, an inherited control is a class that is an extension of an existing control class. Inherited controls are useful if, for example, you want a control with specific properties and behavior that you can use in multiple forms.

To create an inherited control:

1. From the main menu bar, choose **File** > **New** > **ABL Inherited Control**.
2. Enter a value in the **Inhertited Control name** field.
3. In the **Inherits** field, enter the full class path (beginning with `System.Windows.Forms.`) of the control class on which the new inherited control is to be based.

   The remaining fields are optional, with the exception of **Package root** (required).

4. After setting appropriate values in the **Properties** view, click **OK**. Progress Developer Studio for OpenEdge creates a new class file with the same name that you assigned to the control.

A design window appears, but it is not a visual environment like the Design Canvas. You can drag and drop controls on the window, and you can use the **Properties** view to set properties and subscribe to events. However, it is not possible to display a visualization of the control. You must drop inherited controls on a form in order to see them.

`HelpButton.cls` is an inherited control and is one of the pre-existing resources of the SampleApp project. It inherits from the `System.Windows.Form.Button` class. No other controls were dropped on its design window. The font and size properties were changed to make it a square button that accepts a large, bold question mark as a text label. It has a callback to the parent form, which is expected to implement an interface that invokes a help volume. In the sample application, the help button opens the master OpenEdge help system when clicked at runtime.

Like user controls, inherited controls must be added to the Toolbox to make them available for use.

Adding a HelpButton to the Toolbox

To add HelpButton to the Toolbox:

1. Right-click on **My Controls** and select **Add Controls.** The **Add Controls** dialog appears.
2. Select the **ABL Controls** tab.
3. Choose the **HelpButton** control.
4. Click **OK**. **HelpButton** appears under **My Controls**.
5. Close `HelpButton.cls` if it is open.

Adding LoginDlg.cls to the project

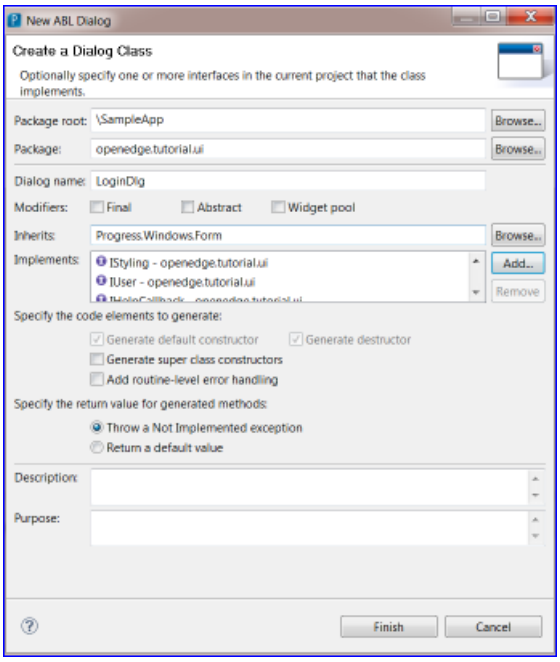
`LoginDlg.cls` is an implementation of an ABL Dialog form. An ABL Dialog is a modal dialog that has **OK** and **Cancel** buttons that are pre-coded with event logic.

To add LoginDlg.cls to the project:

1. Expand the `openedge` node in the SampleApp project.
2. Right-click on the `ui` node.
3. Choose**New** > **ABL Dialog**.
4. In the **New ABL Dialog** wizard, type `LoginDlg` in the **Dialog name** field.
5. Click **Add** (next to the **Implements** field).
6. Add **IStyling -** `openedge.tutorial.ui`, **IUser - openedge.tutorial.ui**, **IHelpCallback - openedge.tutorial.ui** from the **Matching types** list in the **Interface Selection** dialog.

   These interfaces add the templates of the methods required to support the custom controls in the project. You will add these custom controls (HelpButton, LeftBar, and LoginBlock) to the login dialog.

7. Click **OK**. The **ABL Dialog** wizard appears and should look similar to the following:



8. Click **Finish**.
9. In the **Properties** view of `LoginDlg.cls`, set **Text** to `Login`.
10. In the **Properties** view of `LoginDlg.cls`, set **Size** to `595, 320`.

    As you add custom controls to `LoginDlg.cls`, you might want to adjust the size again.

11. From the **OpenEdge Ultra Controls** group in the Toolbox, drag and drop **AppStylistRuntime** on the `LoginDlg.cls` form.

    AppStylistRuntime is a non-visible control that applies styles for an application at runtime. These styles are defined in the three style library (`.ils`) files that are located in the `/styles` folder of the SampleApp project.

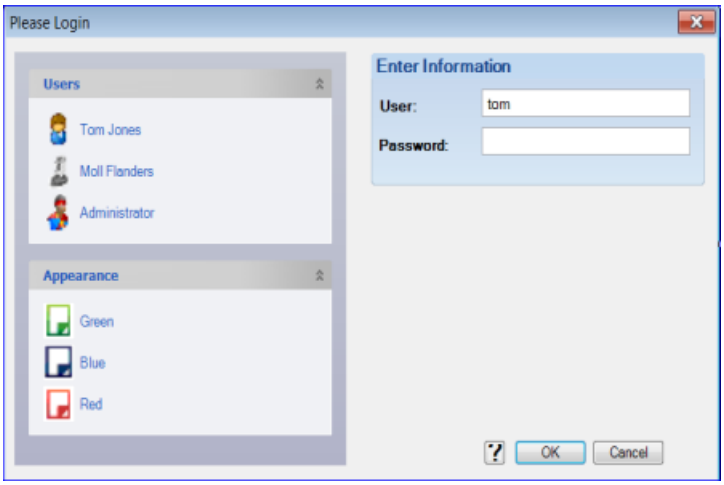
    You can create style libraries with the NetAdvantage® AppStylist® tool. For more information, see:

    http://www.infragistics.com/dotnet/netadvantage/appstylist.aspx

    Remember that each style library file is associated with a particular item in the Appearance group in LeftBar. When you choose an item in the Appearance group at runtime, the style of the application changes.

12. Drag and drop **LeftBar** from the **My Controls** group to the left side of the `LoginDlg.cls` form.
13. Drag and drop **LoginBlock** from the **My Controls** group to the right side of the `LoginDlg.cls` form. Align it to the top of **LeftBar**.
14. Drag and drop **HelpButton** from the **My Controls** group to the bottom of the `LoginDlg.cls` form. Align it to the left of the **OK** button.
15. In the **Properties** view of helpButton1, change **Text** to a question mark (`?`).
16. Adjust the size of the form and the alignment of the controls on the form.

    `LoginDlg.cls` look similar to the following illustration:



Adding interface and login methods to LoginDlg.cls

To complete the login dialog you must add code for the interface methods for the form. When you created `LoginDlg.cls`, you specified that the form would implement the `IStyling`, `IUser`, and `IHelpCallback` interfaces. Templates for the interface methods were automatically added to `LoginDlg.cls`. Now you must complete the templates for those methods.

In addition, you must add some methods (`ValidateLoging` and `LookupLogin`) to complete the "plumbing" that handles logins.

To add interface and login methods to LoginDlg.cls:

1. If `LoginDlg.cls` is open in Visual Designer, press **F9** to view source.

You can also right click on the file name in the **Project Explorer** view and select **Open With** > **OpenEdge ABL Editor**.

2. Find and select the interface method templates in the file:

```
METHOD PUBLIC VOID SetStyle( INPUT styleFile AS System.IO.FileInfo ):
     UNDO, THROW NEW Progress.Lang.AppError("METHOD NOT IMPLEMENTED").
END METHOD.

METHOD PUBLIC VOID SetUser( INPUT name AS CHARACTER, INPUT password AS CHARACTER ):
     UNDO, THROW NEW Progress.Lang.AppError("METHOD NOT IMPLEMENTED").
END METHOD.

METHOD PUBLIC LOGICAL GetHelpContext( OUTPUT pcHelpFilename AS CHARACTER ):
     UNDO, THROW NEW Progress.Lang.AppError("METHOD NOT IMPLEMENTED").
END METHOD.

METHOD PUBLIC LOGICAL GetHelpContext( OUTPUT pcHelpFilename AS CHARACTER, OUTPUT
pcHelpKeyword AS CHARACTER ):
     UNDO, THROW NEW Progress.Lang.AppError("METHOD NOT IMPLEMENTED").
END METHOD.

METHOD PUBLIC LOGICAL GetHelpContext( OUTPUT pcHelpFilename AS CHARACTER, OUTPUT
poHelpNavigator AS System.Windows.Forms.HelpNavigator ):
     UNDO, THROW NEW Progress.Lang.AppError("METHOD NOT IMPLEMENTED").
END METHOD.

METHOD PUBLIC LOGICAL GetHelpContext( OUTPUT pcHelpFilename AS CHARACTER, OUTPUT
poHelpNavigator AS System.Windows.Forms.HelpNavigator,
OUTPUT pcTopicIdentifier AS CHARACTER ):
     UNDO, THROW NEW Progress.Lang.AppError("METHOD NOT IMPLEMENTED").
END METHOD.
```

3. Replace the interface templates with the following code:

```
METHOD PUBLIC VOID SetStyle( INPUT styleFile AS System.IO.FileInfo ):
    Infragistics.Win.AppStyling.StyleManager:Load( styleFile:FullName ).
    appStylistRuntime1:LoadFromStyleManager().
END METHOD.

METHOD PUBLIC VOID SetUser( INPUT name AS CHARACTER, INPUT pwd AS CHARACTER ):
    loginBlock1:loginName:Text = name.
    loginBlock1:loginPwd:Text = pwd.
END METHOD.

METHOD PUBLIC LOGICAL GetHelpContext (OUTPUT pcHelpFilename AS CHARACTER):
    RETURN FALSE.
END METHOD.

METHOD PUBLIC LOGICAL GetHelpContext (OUTPUT pcHelpFilename AS CHARACTER,
OUTPUT pcHelpKeyword AS CHARACTER):
    RETURN FALSE.
END METHOD.

METHOD PUBLIC LOGICAL GetHelpContext (OUTPUT pcHelpFilename AS CHARACTER,
OUTPUT poHelpNavigator AS HelpNavigator):
    RETURN FALSE.
END METHOD.

METHOD PUBLIC LOGICAL GetHelpContext (OUTPUT pcHelpFilename AS CHARACTER,
OUTPUT poHelpNavigator AS HelpNavigator,
OUTPUT pcTopicIdentifier AS CHARACTER):
    /* Example */
    pcHelpFilename = search('prohelp/masteng.chm').
    poHelpNavigator = HelpNavigator:Index.
    pcTopicIdentifier = ''.
    RETURN TRUE.
END METHOD.
```

4. Before the END CLASS statement, add the following code that verifies logins:

```
METHOD PRIVATE LOGICAL ValidateLogin( ):
    IF THIS-OBJECT:loginBlock1:loginName:Text = "" THEN DO:
      MESSAGE "Please enter a login name" VIEW-AS ALERT-BOX.
       THIS-OBJECT:loginBlock1:loginName:Focus().
      RETURN FALSE.
    END.
    IF LookupLogin(THIS-OBJECT:loginBlock1:loginName:Text,
    THIS-OBJECT:loginBlock1:loginPwd:Text) = FALSE THEN DO:
      MESSAGE "Invalid login name and password combination" VIEW-AS ALERT-BOX.
      THIS-OBJECT:loginBlock1:loginPwd:Focus().
      RETURN FALSE.
    END.
    RETURN TRUE.
END METHOD.

METHOD PRIVATE LOGICAL LookupLogin( name AS CHAR, pwd AS CHAR):
    IF name = pwd THEN RETURN TRUE.
    ELSE RETURN FALSE.
END METHOD.
```

5. To ensure that class and interface references are valid, add the following to the `USING` statements section near the top of the file:

```
USING Progress.Windows.*.
USING Progress.Lang.*.
USING openedge.tutorial.ui.*.
USING System.Windows.Forms.*.
```

6. Save and close `LoginDlg.cls`.

Running LoginDlg.cls

You cannot run `LoginDlg.cls` directly. The application must wait for the login to be completed before running the Sports window. Therefore, you must run `start.p`, which contains a `WAIT-FOR` statement that delays the launching of the Sports window.

You can open `start.p` in the ABL Editor to see how the running of the sample application is coded. The `start.p` file is one of the pre-existing resources in the SampleApp project, and it is located in the top-level of the `SampleApp` folder.

To launch start.p:

1. Select `start.p` in the **Project Explorer** view. It is located in the top-level `SampleApp` folder.
2. Select **Run** > **Run as** > **Progress OpenEdge Application** from the main menu.

   When `LoginDlg.cls` runs, a modal login dialog appears, and you can check its functionality which includes:

   - Setting the style of the sample application by selecting one of the three Appearance choices.
   - Selecting one of the three users to set the user name and password.
   - Clicking **OK** to launch the Sports window.

Where to go from here

For the sake of brevity and clarity, many of the features of forms and controls are not implemented in the sample application. For example, we modified only two of the many menu items in the ABL MDI Form.

The intention of the sample application is not to implement every feature. Rather, the intention is to give you an idea of what you can do in the Visual Designer by showing you a representative sample of the forms and controls. You should know how to:

- Use the tools of the Visual Designer to add, arrange, and format various user interface elements
- Use properties and events to modify the behavior of forms and controls
- Implement data binding

The next step is to do some experimenting on your own. You can use this sample application as your starting point.

6

Creating the Sports Window

This chapter describes how to use the Visual Designer to build the Sports window, an implementation of an ABL MDI Form, to the sample application.

For details, see the following topics:

- Overview
- Adding sportsForm.cls to the project
- Adding an UltraExplorerBar to the form
- Adding event handlers
- Modifying the functionality of menu and toolbar items
- Running sportsForm.cls

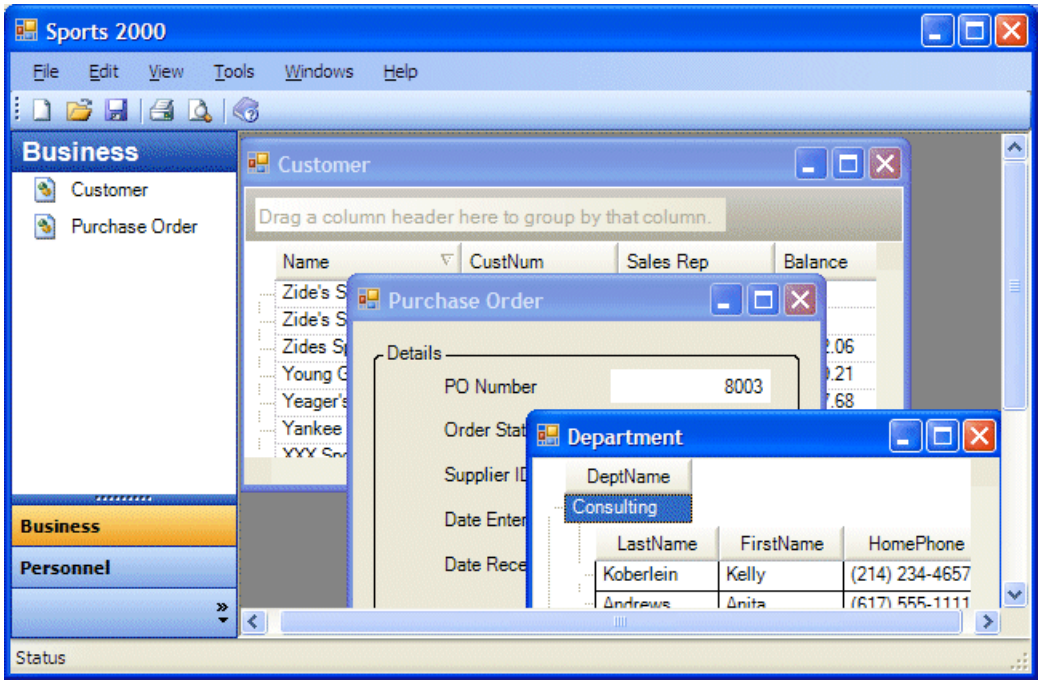
Overview

This chapter describes how to use the Visual Designer to build the Sports window, an implementation of an ABL MDI Form, to the sample application.

The Sports form is an example of an ABL MDI form. MDI (multiple document interface) means it can be a parent for a number of child forms. In the sample application, the Sports form is the parent of the Customer, Department, and Purchase Order forms.

The form, shown in the following figure, includes a menu strip containing common menus (**File**, **Edit**, **View**, **Tools**, **Windows**, **Help**) and a tool strip containing common command buttons (**New**, **Open**, **Save**, **Print**, **Print Preview**, and **Help**). Some of the menus and command buttons are pre-coded with event subscriptions and logic. The form also includes a status bar at the bottom.

Figure 16: The Sports form



Tasks in this chapter

When you create the Sports form, you will:

- Implement an ABL MDI form as a container for other forms
- Implement an UltraExplorerBar control
- Launch child processes from the UltraExplorerBar
- Modify the default behavior of some ABL MDI menu items

Before you begin . . .

- Be sure that you have set up the SampleApp project, created the database, and started the database server as described in Setting Up the Sample Application Project.
- Start Progress Developer Studio for OpenEdge in the workspace that contains the SampleApp project.
- Verify that the SampleApp AVM has started and is connected to the sports2000 server.
- Open the OpenEdge Visual Designer perspective.

Adding sportsForm.cls to the project

The `sportsForm.cls` file that you add to the project in this section is an ABL MDI (multiple document interface) Form that implements the Sports window in the application.

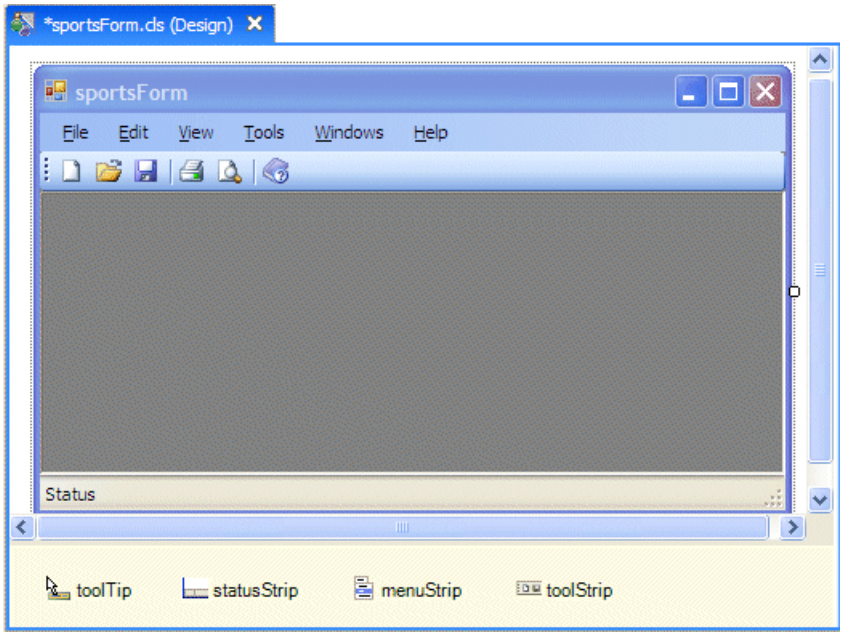
You can use an ABL MDI Form to create a parent form for one or more child forms. In the sample application the Sports window is an implementation of an ABL MDI Form that is the parent window for the Customer, Department, and Purchase Order windows.

The procedure for adding an ABL MDI Form to the project is similar to the procedure for adding an ABL Form to the project.

To add sportsForm.cls to the project:

1. Expand the `openedge` node in the SampleApp project.
2. Right-click on the `ui` node in the `tutorial` folder.
3. Select **New** > **ABL MDI Form**.
4. In the **New ABL MDI Form** wizard, enter `sportsForm` in the **MDI Form name** field.
5. Click **Finish**.

   The ABL MDI form appears in the Visual Designer, as shown in the following illustration:



   Notice that the ABL MDI form has a menu bar, a tool bar, and a status bar by default.

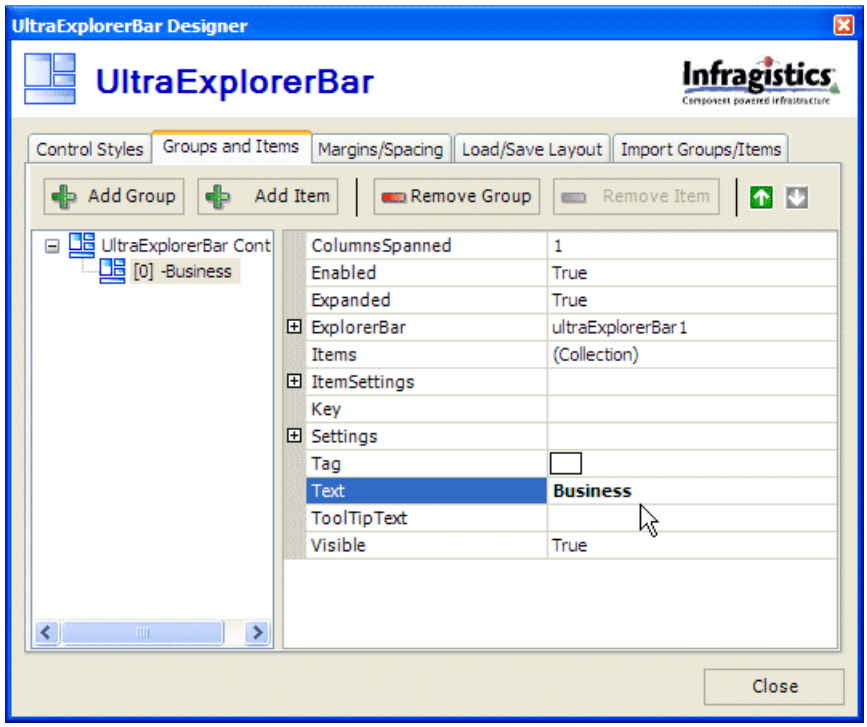
6. In the **Properties** view, set the **Text** property to **Sports 2000**.

Adding an UltraExplorerBar to the form

The UltraExplorerBar is a control that can emulate the look and the functionality of a variety of user interface elements, including explorer bars, list bars, and toolboxes. It allows you to group applications, documents, or other types of objects under labeled headings that can be expanded or minimized. In the sample application, the UltraExplorerBar emulates the navigation pane of Microsoft Office Outlook® 2003. From the UltraExplorerBar, the user can launch the Customer, Purchase Order, and Department forms.

To add the UltraExplorerBar to the sportsForm:

1. With `sportsForm.cls` open in the Visual Designer, expand **OpenEdge Ultra Controls** in the Toolbox.
2. Drag and drop **UltraExplorerBar** on the form.
3. Select the **Dock** property in the **Properties** view. From the drop-down menu, select the **Left** option, which binds the control to the left border of the form.
4. Set the **Style** property to **OutlookNavigationPane**.
5. Right-click on the UltraExplorerBar and choose **UltraExplorerBar Designer** from the drop-down menu.
6. Click the **Groups and Items** tab.
7. Click **Add Group**, then change the **Text** property to `Business`. The UltraExplorerBar Designer looks similar to the following illustration:



8. After clicking **Add Item**:

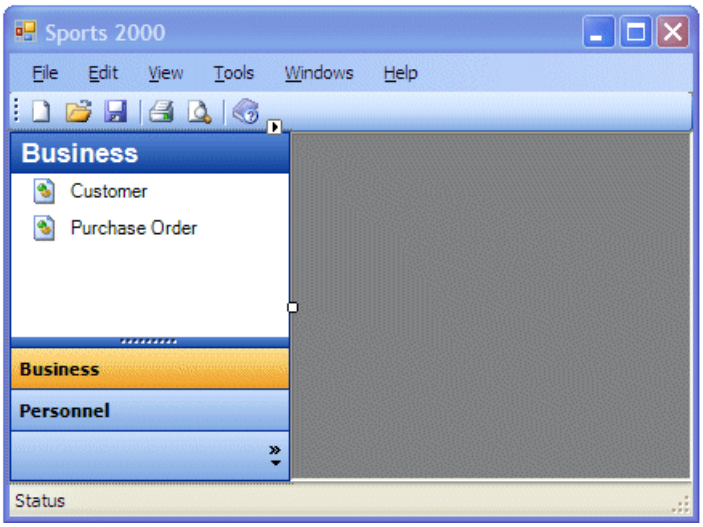
   a) Change the **Text** property to `Customer`.

   b) Change the **Tag** property to `CustMaint`.

   The Text property sets the label of the item. The Tag property is a character string that identifies the item to the event handler logic when you click the item.

9. After clicking **Add Item**:
   a) Change the **Text** property to `Purchase Order`.
   b) Change the **Tag** property to `PurchOrder`.
10. Click **Add Group**, and change the **Text** property to `Personnel`.
11. After clicking **Add Item**:
   a) Change the **Text** property to `Department`.
   b) Change the **Tag** property to `Department`.
12. Click **Close**.

   On the Visual Designer canvas, the Sports 2000 form should look similar to the following:



   If you do not see both the Business and Personnel groups, try increasing the height of the Sports 2000 form.

Adding event handlers

The next step is to add the event methods. This is the code that starts the Customer, Department, and Purchase Order forms as children of the Sports 2000 form. At run time they appear embedded in the right pane of the Sports 2000 form.

Open sportsForm.cls in the ABL Editor and copy and paste the following event methods and event handler before the `DESTRUCTOR` statement:

```
METHOD PRIVATE VOID setAsChild (INPUT oChildForm AS Progress.Windows.Form, INPUT pcTitle
 AS character):
      oChildForm:MdiParent = THIS-OBJECT.
      oChildForm:Text = pcTitle. /*+ " [" + string(childFormNumber) + "]".*/
      oChildForm:FormClosed:SUBSCRIBE(Form_Closed).
      oChildForm:Show( ).
END METHOD.

METHOD PROTECTED VOID showCustomerForm (sender AS System.Object,
e AS System.EventArgs):
     DEF VAR oCBF AS openedge.tutorial.ui.customerForm.
     oCBF = NEW openedge.tutorial.ui.customerForm().
     childFormNumber = childFormNumber + 1.
     setAsChild(oCBF, 'Customer').
DELETE OBJECT e.
END METHOD.     /*showCustomerForm*/

METHOD PROTECTED VOID showDepartmentForm (sender AS System.Object,
e AS System.EventArgs):
     DEF VAR oDT AS openedge.tutorial.ui.departmentForm.
     oDT = NEW openedge.tutorial.ui.departmentForm().
     childFormNumber = childFormNumber + 1.
     setAsChild(oDT, 'Department').
DELETE OBJECT e.
END METHOD.    /* showDepartmentForm */

METHOD PROTECTED VOID showPurchOrderForm (sender AS System.Object,
e AS System.EventArgs):
     DEF VAR oPO AS openedge.tutorial.ui.purchaseOrderForm.
     oPO = NEW openedge.tutorial.ui.purchaseOrderForm().
     childFormNumber = childFormNumber + 1.
     setAsChild(oPO, 'Purchase Order').
DELETE OBJECT e.
END METHOD.

METHOD PRIVATE  VOID ultraExplorerBar1_ItemClick (  sender AS System.Object,  e AS
Infragistics.Win.UltraWinExplorerBar.ItemEventArgs ):
     DEF VAR cTag AS character NO-UNDO.
     cTag = STRING(e:Item:Tag).
     SESSION:set-wait-state('general').
     CASE cTag:
       WHEN 'CustMaint' THEN showCustomerForm(sender, e).
       WHEN 'Department' THEN showDepartmentForm(sender, e).
       WHEN 'PurchOrder' THEN showPurchOrderForm(sender, e).
     END CASE.
SESSION:set-wait-state('').
END METHOD. /* event handler */
```

Subscribe to the event that occurs when the user clicks an item on the Explorer Bar by selecting the UltraExplorerBar1 object in the Visual Designer. Find the **ItemClick** event on the **Events** tab in the **Properties** view, and add `ultraExplorerBar1_ItemClick` as the value.

Modifying the functionality of menu and toolbar items

In an ABL MDI Form, some functionality of the menu and toolbar items is already implemented and requires no further coding on your part. For example, you can use **File** > **Exit** to quit the sample application.

Other functionality is implemented to perform tasks that do not make sense in the context of the sample application. For example, the sample application does not create new files or manipulate existing files, so the default **File** > **New** and **File** > **Open** are unnecessary. In fact, they can confuse an end user if you leave them as they are.

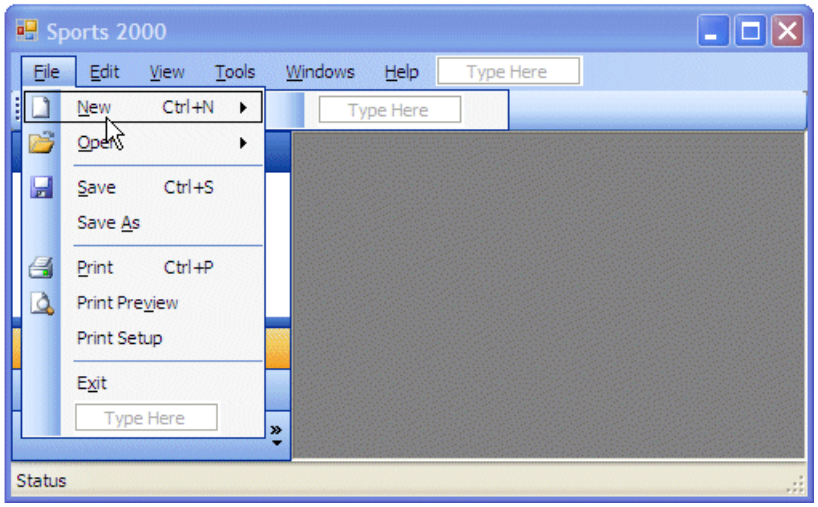
Some menu or toolbar functionality is not implemented at all. For example, in the case of **File** > **Print**, you must either implement print functionality or remove the item from the **File** menu.

In this section, you remove the **File** > **New** menu and change the functionality of the **File** > **Open** menu. You can experiment with the procedures described in this section to modify or delete other items in the menus and toolbar.

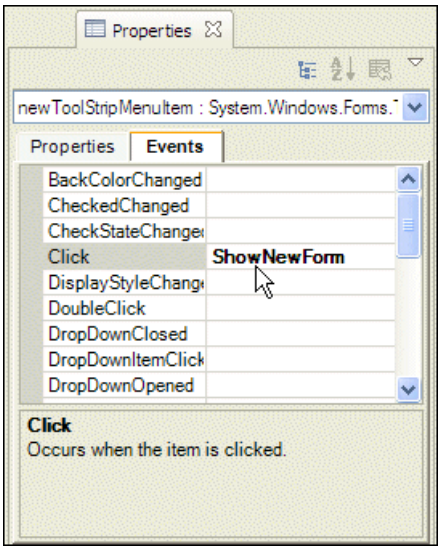
Removing the New menu item

To remove the **File** > **New** menu item:

1. With `sportsForm.cls` open in the Visual Designer, select **File** > **New** from the Sports 2000 menu bar:



Notice that the **Properties** view updates to show the properties and event subscriptions for the selected menu item:



The **Properties** view shows that the **New** menu item (`newToolStripMenuItem`) subscribes to the **Click** event, and the **ShowNewForm** method runs when the menu item is clicked.

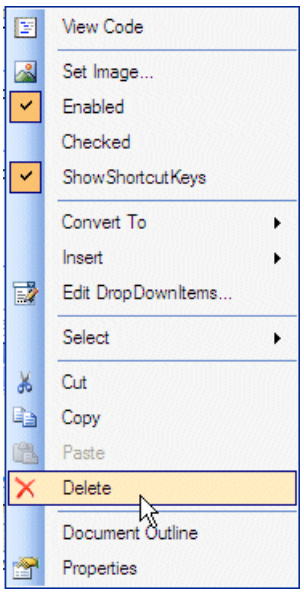
Note: At this point you could disable the **New** menu item simply by deleting **ShowNewForm** from the list, thereby unsubscribing from the **Click** event. However, that would leave an unresponsive menu item, which would look like a defect in a real application.

Another alternative is to modify the `ShowNewForm()` method to do something relevant to the sample application. If you double-click on **ShowNewForm**, the method will appear in the ABL Editor.

---

2. Right-click on the **New** menu item.

   The drop-down menu gives you a number of options, including disabling or enabling, showing or hiding shortcut keys, and deleting as shown in the following illustration:



3. Choose **Delete** to get rid of the **New** menu item.

Modifying the Open menu item

For the Open menu item, you must remove the default functionality and add new features. With the following procedure, you add the ability to open the Customer, Department, and Purchase Order forms from sub-menus of the **Open** menu.

To modify the **File** > **Open** menu item:

1. With `sportsForm.cls` open in the Visual Designer, right-click on the **Open** item in the **File** menu in the Sports 2000 menu.
2. Deselect **Show Shortcut Keys**, which removes **Ctrl+O** from the menu item label.

Note: Another method for removing shortcut keys from menu item labels is to select the menu item and then delete the value of the **ShortcutKeys** property in the Properties view.

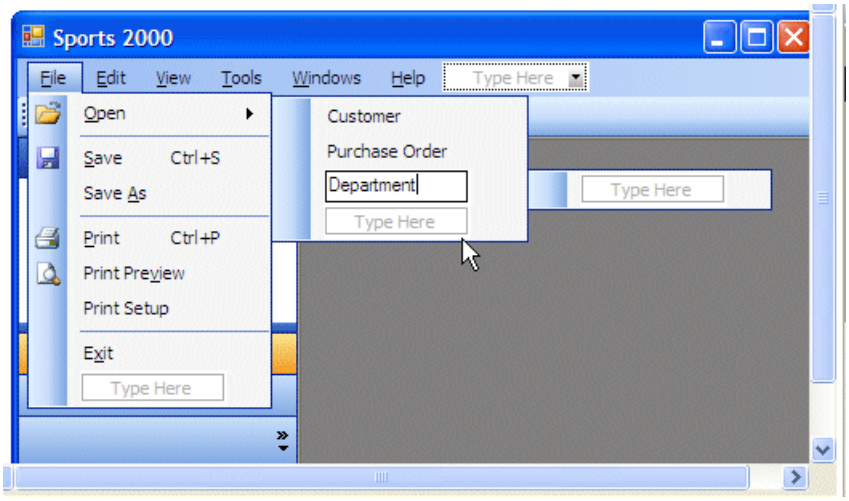
---

3. With the **Open** item selected, find the **Click** event in the **Properties** view.
4. Remove the value `OpenFile` from the **Click** event.

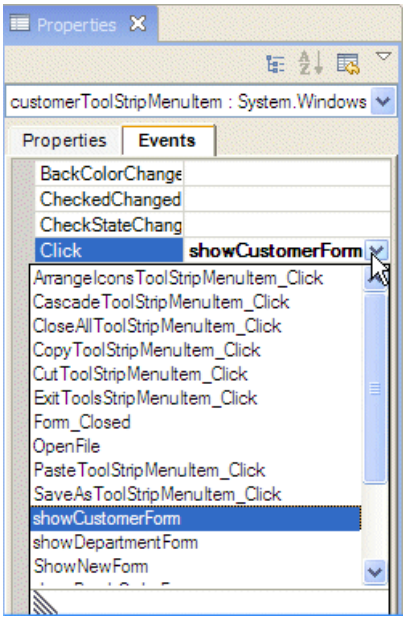
Note: The `OpenFile()` method still exists in the `sportsForm.cls` file, although the **Click** event no longer calls it. You can remove or comment out `OpenFile()`, but there are no adverse consequences if you leave it as is.

5. Click on the **Open** menu item and add `Customer`, `Purchase Order`, and `Department` to the **Type Here** fields of the sub-menu.

   The Sports 2000 **Open** menu looks similar to the following illustration:



6. Select the **Customer** item in the submenu.
7. In the **Properties** view, add `showCustomerForm` as the **Click** event handler. The **Properties** view looks similar to the following illustration:



8. Select the **Purchase Order** item in the submenu.
9. In the **Properties** view, add `showPurchOrderForm` as the **Click** event handler.
10. Select the **Department** item in the submenu.

11. In the **Properties** view, add `showDepartmentForm` as the **Click** event handler.
12. Save `sportsForm.cls`.

Running sportsForm.cls

To run sportsForm.cls:

1. Save `sportsForm.cls`, if necessary.
2. Select `sportsForm.cls` in the **Project Explorer** view.
3. Select **Run** > **Run As** > **Progress OpenEdge Application** on the main menu bar.

When Sports form runs, you can check its functionality, which includes:

- Running the Customer, Purchase Order, and Department forms in the right pane of the Sports form.

  You can run these forms by choosing them on the navigation pane, or by clicking on the **Open** menu item and choosing a form from the submenu.

- Closing the sample application by selecting **File** > **Exit**.

Index

.NET
    control help *20*

A

ABL Dialog *19*
ABL Editor *17*
ABL Form *19*
ABL Inherited Control *20*
ABL MDI Form *19*
ABL User Control *19*
ABL Virtual Machine *13*
AVM *13*

C

Class Browser *17*
CLR *13*
control
    inherited *20*
    user *19*
Control Toolbox *15*
controls
    groups *15*
    Microsoft *15*
    non-visual *14*
    OpenEdge *15*
    OpenEdge Ultra *15*

D

DB Navigator *18*
Debugger *18*
Design Canvas *14*

F

form *19*

H

help *20*

I

inherited control *20*

M

MDI Form *19*
Meta Catalog *18*
Microsoft .NET UI Controls *15*
Microsoft .NET runtime *13*

N

non-visual controls *14*

O

online help *20*

P

perspective *13*

T

Toolbox *15*
Tools for Business Logic *18*

U

user control *19*

V

Visual Designer
    components *14*
    Design Canvas *14*
    perspective *13*