# Contents

Preface ................................................................. Preface–1

1. **Features of the Visual Designer** .......................... 1–1
   What is the Visual Designer? .................................. 1–2
   Components of the Visual Designer ......................... 1–3
      The Design Canvas ........................................... 1–3
      The Control Toolbox ....................................... 1–4
      The Properties view ....................................... 1–5
   Other OpenEdge Architect perspectives, editors, and views ................................. 1–6
      The ABL Editor ............................................... 1–6
      The Class Browser ......................................... 1–6
      Debugger ...................................................... 1–7
      DB Navigator ................................................. 1–7
      Meta Catalog ................................................. 1–7
      Tools for Business Logic .................................. 1–7
   What you can create with the Visual Designer ................. 1–8
      The ABL Form ............................................... 1–8
      The ABL Dialog ............................................. 1–8
      The ABL MDI Form ......................................... 1–8
      The ABL User Control .................................... 1–9
      The ABL Inherited Control ............................... 1–9
   Getting help ..................................................... 1–10
      Visual Designer online help .............................. 1–10
      Help on .NET controls .................................... 1–10
      OpenEdge .NET ABL documentation ..................... 1–10

2. **Setting Up the Sample Application Project** ............... 2–1
   Introducing the sample application ......................... 2–2
   Prerequisites ................................................... 2–3
   Creating the project ......................................... 2–4
   Creating the database ....................................... 2–6
   Adding the database connection to the project ............... 2–7
   Setting up the SampleAppSolution project ................... 2–9
3. Creating the Customer Window ................................................. 3–1
   Overview ................................................................. 3–2
   Tasks in this chapter ...................................................... 3–2
   Before you begin ........................................................... 3–2
   Adding customerForm.cls to the project ................................. 3–3
      Changing the title bar of the form ..................................... 3–5
   Data binding ...................................................................... 3–6
      Adding the ProBindingSource control to a form ..................... 3–6
      Defining the schema ....................................................... 3–7
      Attaching controls to a ProBindingSource ............................ 3–11
      Attaching the ProBindingSource control to a data source ........ 3–14
   Adding interface methods .................................................... 3–18
   Adding event handlers ....................................................... 3–19
   Running customerForm.cls .................................................. 3–21

4. Creating the Department Window ............................................... 4–1
   Overview ................................................................. 4–2
   Tasks in this chapter ...................................................... 4–2
   Before you begin ........................................................... 4–3
   Adding departmentForm.cls to the project ............................... 4–4
   Adding a ProBindingSource control to the form ....................... 4–5
   Adding an UltraTree control to the form .................................. 4–6
   Adding the code that completes data binding ............................ 4–11
   Running departmentForm.cls ............................................... 4–13

5. Creating the Purchase Order Window ........................................ 5–1
   Overview ................................................................. 5–2
   Tasks in this chapter ...................................................... 5–2
   Before you begin ........................................................... 5–2
   Adding purchaseOrderForm.cls to the project ......................... 5–3
   Adding a ProBindingSource control ....................................... 5–4
   Adding a group box control ................................................. 5–5
   Adding editors .................................................................. 5–7
   Adding a panel and buttons ................................................. 5–12
   Adding event handlers ....................................................... 5–15
   Adding the interface methods .............................................. 5–18
   Adding the data binding code .............................................. 5–19
   Running purchaseOrderForm.cls ......................................... 5–20

6. Creating the Sports Window ..................................................... 6–1
   Overview ................................................................. 6–2
   Tasks in this chapter ...................................................... 6–2
   Before you begin ........................................................... 6–2
   Adding sportsForm.cls to the project .................................... 6–3
   Adding an UltraExplorerBar to the form .................................. 6–4
   Adding event handlers ....................................................... 6–6
   Modifying the functionality of menu and toolbar items ............... 6–7
   Running sportsForm.cls .................................................... 6–11
7. Creating the Login Dialog ................................................. 7–1
   Overview ........................................................................... 7–2
   Tasks in this chapter ......................................................... 7–2
   Before you begin .............................................................. 7–2
   Adding LeftBar.cls to the project ....................................... 7–3
       Adding an image list to LeftBar.cls ................................. 7–5
       Adding an UltraExplorerBar to LeftBar.cls .................... 7–7
       Subscribing to events and adding handlers ..................... 7–12
       Adding LeftBar.cls to the Toolbox .................................. 7–14
   Adding LoginBlock.cls to the Toolbox ............................... 7–16
   Adding HelpButton.cls to the Toolbox ............................... 7–17
   Adding LoginDlg.cls to the project ..................................... 7–18
       Adding interface and login methods to LoginDlg.cls ....... 7–20
   Running LoginDlg.cls ....................................................... 7–23
   Where to go from here ....................................................... 7–24
Tables

Table 5–1: Remaining editors for the Details section ......................... 5–11
Table 5–2: Name and Text properties of Purchase Order buttons ............ 5–13
Figures

Figure 1–1: The Visual Designer perspective ................................................. 1–2
Figure 1–2: The Design Canvas ................................................................. 1–3
Figure 1–3: The Control Toolbox ............................................................... 1–4
Figure 1–4: The Properties view ............................................................... 1–5
Figure 2–1: The sample application ............................................................ 2–2
Figure 3–1: The Customer form ................................................................. 3–2
Figure 3–2: Tools links in the Properties view .............................................. 3–13
Figure 3–3: The UltraGrid control in the Customer form .............................. 3–14
Figure 3–4: The UltraGrid with the data band expanded ............................. 3–14
Figure 4–1: The Department form ............................................................. 4–2
Figure 4–2: The UltraTree bound to a ProBindingSource ........................... 4–7
Figure 5–1: The Purchase Order form ........................................................ 5–2
Figure 5–2: The Details section of the Purchase Order form ....................... 5–11
Figure 5–3: The completed Purchase Order form ....................................... 5–14
Figure 6–1: The Sports form ...................................................................... 6–2
Figure 7–1: The Login dialog ..................................................................... 7–2
Figure 7–2: LeftBar.cls ......................................................................... 7–11
This Preface contains the following sections:

- Purpose
- Audience
- Organization
- Typographical conventions
- Third party acknowledgements
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 OpenEdge Architect that allows you to build user interfaces for OpenEdge applications. You create user interfaces with 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

Chapter 1, “Features of the Visual Designer”

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

Chapter 2, “Setting Up the Sample Application Project”

Shows how to set up an OpenEdge Architect project for building the sample application.

Chapter 3, “Creating the Customer Window”

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

Chapter 4, “Creating the Department Window”

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

Chapter 5, “Creating the Purchase Order Window”

Shows how to create an ABL Form that features UltraButtons and a number of editors.
Chapter 6, “Creating the Sports Window”

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

Chapter 7, “Creating the Login Dialog”

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

Typographical conventions

This manual uses the following typographical conventions:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold</strong></td>
<td>Bold typeface indicates commands or characters the user types, provides emphasis, or the names of user interface elements.</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Italic typeface indicates the title of a document, or signifies new terms.</td>
</tr>
<tr>
<td><strong>SMALL, BOLD CAPITAL LETTERS</strong></td>
<td>Small, bold capital letters indicate OpenEdge key functions and generic keyboard keys; for example, GET and CTRL.</td>
</tr>
<tr>
<td><strong>KEY1+KEY2</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>KEY1 KEY2</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Syntax:</strong></td>
<td></td>
</tr>
<tr>
<td>Fixed width</td>
<td>A fixed-width font is used in syntax statements, code examples, system output, and filenames.</td>
</tr>
<tr>
<td><em>Fixed-width italics</em></td>
<td>Fixed-width italics indicate variables in syntax statements.</td>
</tr>
<tr>
<td><em>Fixed-width bold</em></td>
<td>Fixed-width bold indicates variables with special emphasis.</td>
</tr>
<tr>
<td><strong>UPPERCASE fixed width</strong></td>
<td>Uppercase words are ABL keywords. Although these are always shown in uppercase, you can type them in either uppercase or lowercase in a procedure.</td>
</tr>
</tbody>
</table>

This icon (three arrows) introduces a multi-step procedure.

This icon (one arrow) introduces a single-step procedure.

Ellipses indicate repetition: you can choose one or more of the preceding items.
Third party acknowledgements

OpenEdge includes AdventNet - Agent Toolkit licensed from AdventNet, Inc.
http://www.adventnet.com. All rights to such copyright material rest with AdventNet.

OpenEdge includes ANTLR (Another Tool for Language Recognition) software Copyright © 2003-2006, Terence Parr All rights reserved. Neither the name of the author nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes software developed by the Apache Software Foundation (http://www.apache.org/). Copyright © 1999 The Apache Software Foundation. All rights reserved (Xerces C++ Parser (XML) and Xerces2 Java Parser (XML)); Copyright © 1999-2002 The Apache Software Foundation. All rights reserved (Xerces Parser (XML); and Copyright © 2000-2003 The Apache Software Foundation. All rights reserved (Ant). The names “Apache,” “Xerces,” “ANT,” and “Apache Software Foundation” must not be used to endorse or promote products derived from this software without prior written permission. Products derived from this software may not be called “Apache”, nor may “Apache” appear in their name, without prior written permission of the Apache Software Foundation. For written permission, please contact apache@apache.org. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes Concurrent Java software Copyright 1994-2000 Sun Microsystems, Inc. All Rights Reserved. -Neither the name of or trademarks of Sun may be used to endorse or promote products including or derived from the Java Software technology without specific prior written permission; and Redistributions of source or binary code must contain the above copyright notice, this notice and the following disclaimers: This software is provided "AS IS," without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN MICROSYSTEMS, INC. AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN MICROSYSTEMS, INC. OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN MICROSYSTEMS, INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

OpenEdge includes DataDirect software Copyright © 1991-2007 Progress Software Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect for JDBC Type 4 driver); Copyright © 1993-2009 Progress Software Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect for JDBC); Copyright © 1988-2007 Progress Software Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect for ODBC); and Copyright © 1988-2007 Progress Software
Preface

Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect64 for ODBC).

OpenEdge includes DataDirect Connect for ODBC and DataDirect Connect64 for ODBC software, which include ICU software 1.8 and later - Copyright © 1995-2003 International Business Machines Corporation and others All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

OpenEdge includes DataDirect Connect for ODBC and DataDirect Connect64 for ODBC software, which include software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/). Copyright © 1998-2006 The OpenSSL Project. All rights reserved. And Copyright © 1995-1998 Eric Young (eay@cryptsoft.com). All rights reserved.

OpenEdge includes DataDirect products for the Microsoft SQL Server database which contain a licensed implementation of the Microsoft TDS Protocol.

OpenEdge includes software authored by David M. Gay. Copyright © 1991, 2000, 2001 by Lucent Technologies (dtoa.c); Copyright © 1991, 1996 by Lucent Technologies (g_fmt.c); and Copyright © 1991 by Lucent Technologies (rnd_prod.s). Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software. THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.

OpenEdge includes software authored by David M. Gay. Copyright © 1998-2001 by Lucent Technologies All Rights Reserved (decsrtod.c; strtodg.c); Copyright © 1998, 2000 by Lucent Technologies All Rights Reserved (decsrtodf.c; strtodr.c); Copyright © 1998 by Lucent Technologies All Rights Reserved (dmisc.c; gdtoa.h; gethex.c; gmisc.c; sum.c); Copyright © 1998, 1999 by Lucent Technologies All Rights Reserved (gdtoa.c; misc.c; smisc.c; ulp.c); Copyright © 1998-2000 by Lucent Technologies All Rights Reserved (gdtoaimp.h); Copyright © 2000 by Lucent Technologies All Rights Reserved (hd_init.c). Full copies of these licenses can be found in the installation directory, in the c:/OpenEdge/licenses folder. Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of Lucent or any of its entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
OpenEdge includes http package software developed by the World Wide Web Consortium. Copyright © 1994-2002 World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All rights reserved. This work is distributed under the W3C® Software License [http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231] in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

OpenEdge includes ICU software 1.8 and later - Copyright © 1995-2003 International Business Machines Corporation and others All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.


OpenEdge includes Infragistics NetAdvantage for .NET v2009 Vol 2 Copyright © 1996-2009 Infragistics, Inc. All rights reserved.

OpenEdge includes JSTL software Copyright 1994-2006 Sun Microsystems, Inc. All Rights Reserved. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes OpenSSL software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/). Copyright © 1998-2007 The OpenSSL Project. All rights reserved. This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com). Copyright © 1995-1998 Eric Young (eay@cryptsoft.com) All rights reserved. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project. Software distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes Quartz Enterprise Job Scheduler software Copyright © 2001-2003 James House. All rights reserved. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product. This product uses and includes within its distribution, software developed by the Apache Software Foundation (http://www.apache.org/).

OpenEdge includes code licensed from RSA Security, Inc. Some portions licensed from IBM are available at http://oss.software.ibm.com/icu4j/.

OpenEdge includes the RSA Data Security, Inc. MD5 Message-Digest Algorithm. Copyright ©1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.
OpenEdge includes Sonic software, which includes software developed by Apache Software Foundation (http://www.apache.org/). Copyright © 1999-2000 The Apache Software Foundation. All rights reserved. The names “Ant”, “Axis”, “Xalan”, “FOP,” “The Jakarta Project”, “Tomcat”, “Xerces” and/or “Apache Software Foundation” must not be used to endorse or promote products derived from the Product without prior written permission. Any product derived from the Product may not be called “Apache”, nor may “Apache” appear in their name, without prior written permission. For written permission, please contact apache@apache.org.

OpenEdge includes Sonic software, which includes software Copyright © 1999 CERN - European Organization for Nuclear Research. Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. CERN makes no representations about the suitability of this software for any purpose. It is provided "as is" without expressed or implied warranty.

OpenEdge includes Sonic software, which includes software developed by ExoLab Project (http://www.exolab.org/). Copyright © 2000 Intalio Inc. All rights reserved. The names “Castor” and/or “ExoLab” must not be used to endorse or promote products derived from the Products without prior written permission. For written permission, please contact info@exolab.org. Exolab, Castor and Intalio are trademarks of Intalio Inc.

OpenEdge includes Sonic software, which includes software developed by IBM. Copyright © 1995-2003 International Business Machines Corporation and others. All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation. Software distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product. Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

OpenEdge includes Sonic software, which includes the JMX Technology from Sun Microsystems, Inc. Use and Distribution is subject to the Sun Community Source License available at http://sun.com/software/communitysource.

OpenEdge includes Sonic software, which includes software developed by the ModelObjects Group (http://www.modelobjects.com). Copyright © 2000-2001 ModelObjects Group. All rights reserved. The name “ModelObjects” must not be used to endorse or promote products derived from this software without prior written permission. Products derived from this software may not be called “ModelObjects”, nor may “ModelObjects” appear in their name, without prior written permission. For written permission, please contact djacobs@modelobjects.com.

OpenEdge includes Sonic software, which includes code licensed from Mort Bay Consulting Pty. Ltd. The Jetty Package is Copyright © 1998 Mort Bay Consulting Pty. Ltd. (Australia) and others.
OpenEdge includes Sonic software, which includes files that are subject to the Netscape Public License Version 1.1 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at \url{http://www.mozilla.org/NPL/}. Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License. The Original Code is Mozilla Communicator client code, released March 31, 1998. The Initial Developer of the Original Code is Netscape Communications Corporation. Portions created by Netscape are Copyright 1998-1999 Netscape Communications Corporation. All Rights Reserved.

OpenEdge includes Sonic software, which includes software developed by the University Corporation for Advanced Internet Development \url{http://www.ucaid.edu} Internet2 Project. Copyright © 2002 University Corporation for Advanced Internet Development \url{http://www.ucaid.edu} Internet2 Project. Neither the name of OpenSAML nor the names of its contributors, nor Internet2, nor the University Corporation for Advanced Internet Development, Inc., nor UCAID may be used to endorse or promote products derived from this software and products derived from this software may not be called OpenSAML, Internet2, UCAID, or the University Corporation for Advanced Internet Development, nor may OpenSAML appear in their name without prior written permission of the University Corporation for Advanced Internet Development. For written permission, please contact opensaml@opensaml.org.

OpenEdge includes the UnixWare platform of Perl Runtime authored by Kiem-Phong Vo and David Korn. Copyright © 1991, 1996 by AT&T Labs. Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software. THIS SOFTWARE IS BEING PROVIDED “AS IS”, WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHORS NOR AT&T LABS MAKE ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.

OpenEdge includes Vermont Views Terminal Handling Package software developed by Vermont Creative Software. Copyright © 1988-1991 by Vermont Creative Software.

OpenEdge includes XML Tools, which includes versions 8.9 of the Saxon XSLT and XQuery Processor from Saxonica Limited (\url{http://www.saxonica.com/}) which are available from SourceForge (\url{http://sourceforge.net/projects/saxon/}). The Original Code of Saxon comprises all those components which are not explicitly attributed to other parties. The Initial Developer of the Original Code is Michael Kay. Until February 2001 Michael Kay was an employee of International Computers Limited (now part of Fujitsu Limited), and original code developed during that time was released under this license by permission from International Computers Limited. From February 2001 until February 2004 Michael Kay was an employee of Software AG, and code developed during that time was released under this license by permission from Software AG, acting as a "Contributor". Subsequent code has been developed by Saxonica Limited, of which Michael Kay is a Director, again acting as a "Contributor". A small number of modules, or enhancements to modules, have been developed by other individuals (either written especially for Saxon, or incorporated into Saxon having initially been released as part of another open source product). Such contributions are acknowledged individually in comments attached to the relevant code modules. All Rights Reserved. The contents of the Saxon files are subject to the Mozilla Public License Version 1.0 (the "License"); you may not use these files except in compliance with the License. You may obtain a copy of the License at \url{http://www.mozilla.org/MPL/} and a copy of the license can also be found in the
installation directory, in the c:/OpenEdge/licenses folder. Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

OpenEdge includes XML Tools, which includes Xs3P v1.1.3. The contents of this file are subject to the DSTC Public License (DPL) Version 1.1 (the "License"); you may not use this file except in compliance with the License. A copy of the license can be found in the installation directory, in the c:/OpenEdge/licenses folder. Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License. The Original Code is xs3p. The Initial Developer of the Original Code is DSTC. Portions created by DSTC are Copyright © 2001, 2002 DSTC Pty Ltd. All rights reserved.

OpenEdge includes YAJL software Copyright 2007, Lloyd Hilaiel. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. Neither the name of Lloyd Hilaiel nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Features of the Visual Designer

This chapter provides an overview of the Visual Designer.

Topics in this chapter include:

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

Figure 1–1: The Visual Designer perspective

The Visual Designer is a WYSIWIG editor for building 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 is implemented as a plug-in to the Eclipse-based OpenEdge Architect. This provides an integrated development environment with quick access to other OpenEdge Architect tools such as DB Navigator, the ABL Editor, and Tools for Business Logic.

The Visual Designer perspective, shown in Figure 1–1, 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 the “Components of the Visual Designer” section on page 1–3. In addition to the primary design tools, the Visual Designer Perspective includes a Resources view that shows project resources, and an Outline view that helps you to navigate in code files.
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 Figure 1–2, 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. (See the “What you can create with the Visual Designer” section on page 1–8 for more information.) 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 1–2: The Design Canvas](image)

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 Figure 1–2), 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 OpenEdge Architect
- **OpenEdge** — OpenEdge-specific controls for binding to OpenEdge data sources, and for embedding OpenEdge windows that install with OpenEdge Architect
- **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.

When you expand a group, as shown in Figure 1–3, you can select a control, and then drag and drop it on a form in the Design Canvas.

![Control Toolbox Diagram]

**Figure 1–3: 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 Figure 1–3.
The Properties view

The Visual Designer Properties view, shown in Figure 1–4, 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.

![Properties view diagram](image)

- **Name of selected control**
- **List of properties or events**
- **Command panel**
- **Description of property or event**

**Figure 1–4: 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 OpenEdge Architect perspectives, editors, and views

Since the Visual Designer is a plug-in to OpenEdge Architect, you have easy access to the other features of the OpenEdge Architect 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 run the ABL Editor when you 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

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 OpenEdge Architect Eclipse workbench.
Debugger

The OpenEdge Architect 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 OpenEdge Architect 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 OpenEdge Architect workbench.

Meta Catalog

The OpenEdge Architect 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 OpenEdge Architect Eclipse workbench.

Tools for Business Logic

OpenEdge Architect 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 OpenEdge Architect 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 Chapter 3, “Creating the Customer Window,” Chapter 4, “Creating the Department Window” and Chapter 5, “Creating the Purchase Order Window.”

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 Chapter 7, “Creating the Login Dialog.”

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, Print Preview, 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 Chapter 6, “Creating the Sports Window.”
The ABL User Control

The .NET controls in the OpenEdge Architect 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 the “Adding LeftBar.cls to the project” section on page 7–3.

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 the “Adding HelpButton.cls to the Toolbox” section on page 7–17.
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 OpenEdge Architect tools is the OpenEdge Architect 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 OpenEdge Architect Guide.

Help on .NET controls

OpenEdge Architect 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 OpenEdge Architect help. The Microsoft Document Explorer is automatically installed when you install OpenEdge Architect.

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.
The remainder of this manual guides you through the creation of a simple sample application using the Visual Designer. Before you begin working on the sample application, you need to set up the project that is described in this chapter.

Topics in this chapter include:

- Introducing the sample application
- Prerequisites
- Creating the project
- Creating the database
- Adding the database connection to the project
- Setting up the SampleAppSolution project
Introducing the sample application

The sample application, shown in Figure 2–1, consists of five forms: a container, three application windows, and a login dialog. The chapters that follow describe how to create each form. As you create each form, you will learn:

- How to use the Visual Designer to implement some of the controls commonly used in OpenEdge GUI for .NET applications
- How to connect the controls to data sources
- How to handle events

Figure 2–1: The sample application

Be aware that the sample application is not meant to be an example of a real-world application. It does not represent a solution to any particular business problem. It is not meant to be a model of best practices for coding or for UI design. It is a simplified application that introduces the features of the Visual Designer.
Prerequisites

Before you begin to work with the sample application, you must:

- Install the 10.2x release of OpenEdge Architect.
- Install OpenEdge® Ultra Controls for .NET.
- Unzip `visualdesigner.zip` in an OpenEdge Architect workspace. The SampleApp and SampleAppSolution project folders are created in the workspace.

The `visualdesigner.zip` file is available from the Documentation and Samples directory (`doc_samples`) on the OpenEdge product DVD. After you install OpenEdge 10.2x Documentation and Samples from the DVD by running `setup.exe`, you can find `visualdesigner.zip` in:

```
oedoc-samples-install-dir\src\prodoc\visualdesigner
```

You can also find a link to example procedure files (which includes `visualdesigner.zip`) on the OpenEdge Documentation section of the Progress Communities Web site.
Creating the project

In this section, you will convert the SampleApp folder into a project in OpenEdge Architect. The SampleApp folder contains project resources that are necessary for creating and running the sample application. For example, the SampleApp folder contains:

- Most of the files necessary to implement data access
- The graphics files you will use to add images to the UI
- Interface files
- Other files you do not create but will use as you follow the exercises in this manual

**Note:** You can also follow the procedure in this section to convert the SampleAppSolution folder into an OpenEdge project. The SampleAppSolution folder contains required project resources, as well as completed versions of the sample application’s forms. After you add a database resource to the SampleAppSolution project, you will be able to run the sample application (see the “Creating the database” section on page 2–6 and the “Adding the database connection to the project” section on page 2–7).

---

To create a project for the sample application:

1. Start OpenEdge Architect in the workspace that contains the SampleApp folder.

   This should be the workspace where you unzipped the visualdesigner.zip file, which also contains the SampleAppSolution folder.

   **Note:** If you are starting OpenEdge Architect in a new workspace, you should use the OpenEdge Architect Clean startup option.

2. Select File → New → OpenEdge Project.

   The New OpenEdge Project dialog opens.

3. Enter SampleApp in the Project name field. For example:

   ![Project name dialog](image)

   Be sure that Use default location is selected if the SampleApp folder is in the top level of your workspace folder. If it is not, deselect Use default location and browse to the folder that contains SampleApp.
4. Click **Finish**.

OpenEdge Architect creates the **SampleApp** project, which includes all the files and subfolders contained in the SampleApp folder. You can view the contents of the project in the **Resources** view. For example:

5. Right click on **SampleApp** in the **Resources** view, and choose **OpenEdge→ Compile**.

**Note:** After compilation, you will see an error marker on the **startup.p** file, because the file refers to objects that you have not created yet. You can ignore the error marker for now. When you complete all the exercises in this manual the errors will be resolved.
Creating the database

Data access in the sample application is provided by business entities that access the sports2000 database and communicate to the application through a service interface. The class files for the business entities and service interface are already provided in the SampleApp project files, so it is not necessary to create them. However, you must create a working copy of the sports2000 database.

The following procedure shows one of the many ways to create a copy of the sports2000 database. It illustrates the use of the Data Dictionary, which is one of several OpenEdge tools that you can launch directly from OpenEdge Architect.

To create a working copy of the sports2000 database:

1. Start OpenEdge Architect in the workspace where you created the SampleApp project.
2. Select SampleApp in the Resources view.
3. Select OpenEdge→ Admin→ Data Dictionary from the main menu bar.
5. Click OK.
6. Enter the pathname of the database in the New Physical Database Name field.
7. Choose A copy of the sports2000 database in the Start with section.
8. Click OK.
9. In the Connect Database dialog, click Cancel.
10. Select Database→ Exit from the Data Dictionary main menu bar.

After you create a copy of the sports2000 database, you must create a connection profile and add it to the SampleApp project.
Adding the database connection to the project

A database connection is one of the project resources that are necessary to run the sample application.

To add the database connection to the project:

1. In the OpenEdge Editor perspective, right-click SampleApp in the Resources view.
2. From the menu, choose Properties.
3. In the Properties dialog, expand the OpenEdge node and choose Database Connections.
4. In the Database Connections page, click Configure Database Connections.
5. Click New. The Add Connection Profile wizard opens.
6. In the Add Connection Profile page, type sports2000 as a connection name. Then provide the physical name of the database, the host name, and a service or port number. For example:

   ![Connection settings](example.png)

   - Connection name: sports2000
   - Physical name: C:\OpenEdge\WRA\workspace\db\sports2000\sports2000.db

7. Click Next.
8. On the Specify whether to define a SQL Connection page, accept the defaults and click Next.
9. On the Add SQL Connection Profile page, accept the defaults and click Next.

   The sample application must have a running database server in order to access the sports2000 database. Although you can manually start the server with OpenEdge Explorer, Progress Explorer or the proserve command, the auto-start option ensures that the server is running when you open the project, and provides a message in the Console view confirming that the database is connected.

   **Note:** The OpenEdge AdminServer must be running before you can start a database server.

11. Select Auto-shutdown database server if you want to stop the server when you quit the OpenEdge Architect session or change workspaces.
12. Click Finish to return to the Database Connections page.

13. Click OK to return to the project’s Properties dialog.

14. On the Database Connections page of the Properties dialog, select the check box next to the sports2000 connection name.

Select the Show All button if you do not see the following connection listed:

![Database Connections](image)

15. Click OK.

The project restarts, and the database server starts. Messages in the Console view confirm that the AVM (runtime) has started and is connected to sports2000. For example:

```
Project 'SampleApp' runtime started successfully.
Database server 'C:\OpenEdge\WRK\workspace\db\sports2000\sports2000.db' started. (-S 6000)
Project 'SampleApp' runtime connection to 'sports2000' : OK
```
Setting up the SampleAppSolution project

In addition to SampleApp, you should also have a SampleAppSolution folder in your workspace. The SampleAppSolution contains completed versions of the files that you will develop in the following chapters. You can use the files (customerForm.cls, departmentForm.cls, purchaseOrderForm.cls, sportsForm.cls, and LoginDlg.cls) as a source for the code that you will need to copy and paste into your own source files.

You can also follow the directions in the “Creating the project” section on page 2–4 to make a SampleAppSolution OpenEdge project. After you add the sports2000 connection profile to the SampleAppSolution project, you can run customerForm.cls, departmentForm.cls, purchaseOrderForm.cls, and sportsForm.cls by selecting Run → Run As → OpenEdge Application from the main menu bar. See the “Running LoginDlg.cls” section on page 7–23 for information about running LoginDlg.cls, since it differs from the other forms.
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.

Topics in this chapter include:

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

Overview

The Customer form, shown in Figure 3–1, 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 3–1: 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 Chapter 2, “Setting Up the Sample Application Project.”

☐ Start OpenEdge Architect 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 Architect Editor perspective displays messages about project and database startup.)

☐ Open the 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 Resources view, expand the openedge node in the SampleApp project.
2. Right-click on the ui node in the tutorial folder.
3. Select New → ABL Form. For example:

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 `openedge.tutorial.ui.IUpdatable` from the Matching types list in the Interface Selection dialog.

**Note:** If you do not see `openedge.tutorial.ui.IUpdatable` on the list, compile the project. Right-click on `SampleApp` in the Resources view, and choose 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, which should now look similar to the following:
8. Click Finish.

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

![Design Canvas of Visual Designer](image)

**Notes:** 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(). The “Adding interface methods” section on page 3–18 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
- Defining the schema
- Attaching controls to a ProBindingSource
- Attaching the ProBindingSource control to a data source

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. For example:
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:

![ProBindingSource Designer](image)

### 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 field, you can import the schema from a schema definition (XSD) file, or you can 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 the “Adding a ProBindingSource control to the form” section on page 4–5.)

For more information about setting up the sports2000 database, see the “Creating the database” section on page 2–6 and the “Adding the database connection to the project” section on page 2–7.

**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 selected database** button on the menu bar.

The Schema Selection dialog appears. It allows you to select schema information from the tables and fields of databases that are connected to the project.
3. Expand the **sports2000** node and the **Customer** node in the Schema Selection dialog. For example:

![Schema Selection Dialog](image)

4. **CTRL+CLICK** to select **Name**, **CustNum**, **SalesRep**, and **Balance**.

If you select these field names in the stated order, they will appear in the same order in the ProBindingSource Designer.

5. Click **OK**.

6. Select **Customer** from the Tables list and change its name to **ecustomer**. For example:

![Table Selection](image)

**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 field and change the **Name** and **Label** properties to match the following values:

<table>
<thead>
<tr>
<th>Name</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>CustNum</td>
<td>Customer Number</td>
</tr>
<tr>
<td>SalesRep</td>
<td>Sales Rep</td>
</tr>
<tr>
<td>Balance</td>
<td>Balance</td>
</tr>
</tbody>
</table>

The ProBindingSource Designer should now look similar to the following:

8. With the **ecustomer** table selected in the ProBindingSource Designer, add a child table by clicking the **Add Table** button on the menu bar.

9. Rename the child table from **Table1** to **esalesrep**.

When you add a child table, the ProBindingSource Designer adds a default field, **Field1**, as a place holder. You will remove the default field after you import schema from the database.

10. With **esalesrep** selected, click the **Import schema from selected database** button on the menu bar.

11. Expand the **sports2000** node and the **Salesrep** node in the Schema Selection dialog.

12. **CTRL+CLICK** to select **SalesRep** and **RepName**.
13. Click **OK**. The ProBindingSource Designer should now look similar to the following:

![Diagram of ProBindingSource Designer]

14. Right-click **Field1** and select **Delete**.

15. Select each remaining field and change the **Name**, and **Label** properties to match the following values:

<table>
<thead>
<tr>
<th>Name</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>SalesRep</td>
<td>Sales Rep</td>
</tr>
<tr>
<td>RepName</td>
<td>Name</td>
</tr>
</tbody>
</table>

The ProBindingSource Designer should now look similar to the following:

![Diagram of ProBindingSource Designer with updated fields]

16. Click **OK** to save your changes.
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 opens.

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. For example:
4. Select **DataSource**, and then select **bindingSource1** from the drop-down menu. For example:

![Image of Data Source Selection](image)

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

![Image of Data Source Schema Preview](image)

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.

To position the UltraGrid control within the form:

1. Select **ultragrid1**. You can either select it on the Design Canvas or choose it from the drop-down menu at the top of the Properties view.

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. For example:

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 **Figure 3–2**, there are links to several tools that can help you set the properties for the control.

**Figure 3–2:** 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 Figure 3–3) displays bands of data with representative values, based on data type, in various fields. It also shows the label specified for each field.

Figure 3–3: The UltraGrid control in the Customer form

When you expand the node, bands representing the child table appear, as shown in Figure 3–4. The child table is esalesrep, which you specified when you created the ProBindingSource control, bindingSource1.

Figure 3–4: 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.

**Notes:** 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 add 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 Resources 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.
To update the existing constructor and destructor:

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

```csharp
CONSTRUCTOR PUBLIC customerForm ( ):
    SUPER().
    oServiceAdapter = NEW 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:

```csharp
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 `customerForm.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 `DeleteRecord` and `SaveRecord` stub code 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`. 

---

3–18
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 Properties view, the Visual Designer generates the appropriate `SUBSCRIBE` statement in the source code. It also generates the event-handling method. OpenEdge Architect 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:

   ```abl
   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:

```vbnet
METHOD PRIVATE VOID ultraGrid1_BeforeRowsDeleted
(INPUT sender AS System.Object,
THIS-OBJECT:DeleteRecord().
RETURN.
END METHOD.
```

8. Save `customerForm.cls`. 

## Running customerForm.cls

To run customerForm.cls:

1. Save `customerForm.cls`, if necessary.

2. Select `customerForm.cls` in the Resources view.

3. Select `Run → Run As → OpenEdge Application` on the main menu bar.

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.

- Grouping 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.
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.

Topics in this chapter include:

- 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 Figure 4–1, 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 4–1: 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 Chapter 2, “Setting Up the Sample Application Project.”
- Start OpenEdge Architect 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 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” section on page 3–3. 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.
2. Right-click on the `ui` node in the `tutorial` folder.
3. 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 the “Defining the schema” section on page 3–7). Importing schema information from an XSD file is faster and less error-prone than other methods.

See the “Data binding” section on page 3–6 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 opens, click the Import Schema icon on the toolbar. An Open dialog appears.

4. Navigate to the services folder in the 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 Selection dialog appears

6. Click OK. The schema from department.xsd is added to the ProBindingSource Designer.

7. Click OK in the ProBindingSource Designer.

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.
3. Click the Smart Tag arrow on the UltraGrid control 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. For example:

4. In the Smart Tag panel, set DataSource to bindingSource1 and Dock to Fill. For example:

5. Click outside the Smart Tag panel to dismiss it.
Figure 4–2 shows the UltraTree, after data binding, with all the nodes expanded.

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.

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. For example:
The ColumnSet Layout Designer displays the fields of the root table in the right pane. For example:

3. In the ColumnSet Layout Designer, right-click on DeptCode and choose Hide Column. For example:

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 **employee** from the **UltraTree ColumnSet** field. For example:

![UltraTree ColumnSet Layout Designer](image)

The ColumnSet Layout Designer displays the fields of the child table **employee** 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:

![UltraTree ColumnSet Layout Designer](image)

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

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 leave 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 the “Attaching the ProBindingSource control to a data source” section on page 3–14.

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:

```abl
DEFINE PRIVATE VARIABLE oServiceAdapter AS
dopenedge.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:

```abl
/* 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:

```abl
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:

```csharp
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 Resources view.
3. Select **Run** → **Run As** → **OpenEdge Application** on the main menu bar.

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

- Displaying DeptNames as nodes in a tree view. For example:

- Expanding a department node to display employee information for a department. For example:
Creating the Department Window

- Expanding an employee node to display family information. For example:

- Changing the sort order by clicking a column heading
- Resizing column widths
Creating the Purchase Order Window

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

Topics in this chapter include:

- 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

The Purchase Order form, shown in Figure 5–1, 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.

![The Purchase Order form](image)

**Figure 5–1: 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 Chapter 2, “Setting Up the Sample Application Project.”
- Start OpenEdge Architect 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 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” section on page 3–3. That section provides more detailed information about the steps in this section.

To add purchaseOrderForm.cls to the project:

1. Expand the openedge node in the SampleApp project.
2. Right-click on the ui node.
3. 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 openedge.tutorial.ui.IUpdatable from the Matching types 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 the “Data binding” section on page 3–6. 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 Chapter 4.

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 opens, select the Import Schema icon from the toolbar. An Open dialog appears.

4. Navigate to the services folder in the project and choose the purchorder.xsd file.

5. Click Open. The Schema Selection dialog appears.

6. Click OK. The schema from purchorder.xsd is added to the ProBindingSource Designer.

7. Click OK in the ProBindingSource Designer.
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 menubar. Also, check the layout preferences available from Window → Preferences → OpenEdge Architect → 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:

![Purchase Order Window](image)

After you add some controls to the Details group box (see the “Adding editors” section on page 5–7), 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:

   ![UltraNumericEditor control](image)

   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** (nnnnnnnn):

![Smart Tag Panel](image.png)

**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. For example:

```
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**:

![UltraLabel Tasks](image)

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. For example:

```
\[ \]
```

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, chose **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.
Table 5–1 lists the remaining editors and labels to add to ultraGroupBox1.

**Table 5–1: Remaining editors for the Details section**

<table>
<thead>
<tr>
<th>Editor</th>
<th>(Databinding)/Text property</th>
<th>MaskInput property</th>
<th>ultraLabel Text property</th>
</tr>
</thead>
<tbody>
<tr>
<td>UltraTextEditor1</td>
<td>POStatus</td>
<td>NA</td>
<td>Order Status</td>
</tr>
<tr>
<td>UltraNumericEditor2</td>
<td>SupplierIDNum</td>
<td>Integer 1</td>
<td>Supplier ID</td>
</tr>
<tr>
<td>UltraDateTimeEditor1</td>
<td>DateEntered</td>
<td>Date 1</td>
<td>Date Entered</td>
</tr>
<tr>
<td>UltraDateTimeEditor2</td>
<td>ReceiveDate</td>
<td>Date 1</td>
<td>Date Received</td>
</tr>
</tbody>
</table>

After you add and align all the editors and labels, the **Details** section of the Purchase Order form should look similar to **Figure 5–2**.

![Figure 5–2: The Details section of the Purchase Order form](image-url)
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. For example:

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:

![Purchase Order form image]

5. Set the properties of the buttons as specified in Table 5–2.

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

<table>
<thead>
<tr>
<th>Default Name</th>
<th>Name property</th>
<th>Text property</th>
</tr>
</thead>
<tbody>
<tr>
<td>ultraButton1</td>
<td>btnFirst</td>
<td>First</td>
</tr>
<tr>
<td>ultraButton2</td>
<td>btnPrev</td>
<td>Previous</td>
</tr>
<tr>
<td>ultraButton3</td>
<td>btnNext</td>
<td>Next</td>
</tr>
<tr>
<td>ultraButton4</td>
<td>btnLast</td>
<td>Last</td>
</tr>
<tr>
<td>ultraButton5</td>
<td>btnAdd</td>
<td>Add</td>
</tr>
<tr>
<td>ultraButton6</td>
<td>btnDelete</td>
<td>Delete</td>
</tr>
<tr>
<td>ultraButton7</td>
<td>btnSave</td>
<td>Save</td>
</tr>
<tr>
<td>ultraButton8</td>
<td>btnCancel</td>
<td>Cancel</td>
</tr>
</tbody>
</table>

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 Figure 5–3.

Figure 5–3: 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:

```abl
    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:

```vbnet
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):
END METHOD.

METHOD PRIVATE VOID btnPrev_Click (sender AS System.Object,
   e AS System.EventArgs):
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 the **“Adding the interface methods”** section on page 5–18.
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:

<table>
<thead>
<tr>
<th>Button name</th>
<th>Click value</th>
</tr>
</thead>
<tbody>
<tr>
<td>btnFirst</td>
<td>btnFirst_Click</td>
</tr>
<tr>
<td>btnPrev</td>
<td>btnPrev_Click</td>
</tr>
<tr>
<td>btnNext</td>
<td>btnNext_Click</td>
</tr>
<tr>
<td>btnLast</td>
<td>btnLast_Click</td>
</tr>
<tr>
<td>btnAdd</td>
<td>btnAdd_Click</td>
</tr>
<tr>
<td>btnDelete</td>
<td>btnDelete_Click</td>
</tr>
<tr>
<td>btnSave</td>
<td>btnSave_Click</td>
</tr>
<tr>
<td>btnCancel</td>
<td>btnCancel_Click</td>
</tr>
</tbody>
</table>

**Note:** You can double-click on a control to subscribe to the control’s default event. The ABL Editor opens 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.

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 with the other variable definitions that appear after the opening CLASS statement. The variable definition appears in bold in the following example:

   ```abl
   CLASS openedge.tutorial.ui.purchaseOrderForm INHERITS Form IMPLEMENTS IUpdatable :
   DEFPRIVATETAR fRecordState AS CHARACTER NO-UNDO.
   DEFPRIVATETAR components AS System.ComponentModel.IContainer.
   ENDCLASS.
   ``

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

   ```abl
   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.
   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 IUpdatable.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 variable definition for fServiceAdapter below 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 fServiceAdapter AS openedge.tutorial.services.serviceAdapter NO-UNDO.
   DEF PRIVATE VAR fRecordState AS CHARACTER NO-UNDO.
   ```

2. 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.
   ```

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

   ```
   CONSTRUCTOR PUBLIC  purchaseOrderForm (  ):
   SUPER().
   fServiceAdapter = NEW 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.
   ```

4. Save the file.
Running purchaseOrderForm.cls

To run purchaseOrderForm.cls:

1. Save purchaseOrderForm.cls, if necessary.
2. Select purchaseOrderForm.cls in the Resources view.
3. Select Run → Run As → 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.
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.

Topics in this chapter include:

- 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
- Running sportsForm.cls
Overview

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 Figure 6–1, 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 6–1: The Sports form](image)

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 Chapter 2, “Setting Up the Sample Application Project.”
- Start OpenEdge Architect in the workspace that contains the SampleApp project.
- Verify that the SampleApp AVM has started and is connected to the sports2000 server.
- Open the 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 Form name field.
5. Click Finish.

The ABL MDI form appears in the Visual Designer. For example:

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

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. In the Properties view, set **Dock** to **Left**.
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**. For example:

![UltraExplorerBar Designer](image)

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:

```abl
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.

  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*/

  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 */

  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.

  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.

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:

![Menu Item Removal](image)
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. For example:
3. Choose **Delete** to get rid of the **New** 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 menubar.

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. For example:

6. Select the **Customer** item in the submenu.
7. In the Properties view, add `showCustomerForm` as the Click event handler. For example:

![Properties view screenshot](image)

8. Select the Purchase Order item in the submenu.

9. In the Properties view, add `showPurchaseOrderForm` 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 Resources view.
3. Select Run → Run As → 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.
Creating the Login Dialog

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

Topics in this chapter include:

- 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

The Login dialog, shown in Figure 7–1, 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 7–1: 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 Chapter 2, “Setting Up the Sample Application Project.”

☐ Start OpenEdge Architect in the workspace that contains the SampleApp project.

☐ Verify that the SampleApp AVM has started and is connected to the sports2000 server.

☐ Open the Visual Designer perspective.
Adding LeftBar.cls to the project

One or more of the .NET controls in the OpenEdge Architect 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 the “Adding HelpButton.cls to the Toolbox” section on page 7–17.

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.

**To 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. For example:

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

5. Click **Finish**.

**LeftBar.cls** opens in the Visual Designer with a resizable container displayed in the Design Canvas. For example:
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**. For example:

![Toolbox screenshot]

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. For example:

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**. For example:

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. Set the Dock property to Fill.
4. Click the Smart Tag arrow on ultraExplorerBar1 to display the UltraExplorerBar Tasks dialog. For example:

   ![UltraExplorerBar Tasks dialog]

5. Set the properties shown in the following table:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Style of Control</td>
<td>ExplorerBar</td>
</tr>
<tr>
<td>Style of Groups</td>
<td>LargeImagesWithText</td>
</tr>
<tr>
<td>Style of Items</td>
<td>Button</td>
</tr>
</tbody>
</table>

   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 will 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`. For example:

![ImageList Small and Large Properties](image.png)

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. For example:

```
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 start up. 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) 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:

<table>
<thead>
<tr>
<th>Item</th>
<th>Property</th>
<th>Settings&gt;AppearancesLarge&gt;Appearance&gt;Image</th>
<th>Settings&gt;AppearancesSmall&gt;Appearance&gt;Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>Text</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[0]</td>
<td>tom</td>
<td>Tom Jones</td>
<td>user1.png</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[1]</td>
<td>moll</td>
<td>Moll Flanders</td>
<td>user2.png</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[2]</td>
<td>admin</td>
<td>Administrator</td>
<td>admin.png</td>
</tr>
</tbody>
</table>

The value in the key field is passed as a parameter to an event handler when one of the items is selected. See the “Subscribing to events and adding handlers” section on page 7–12 for more information. The value is used in the User and Password fields that are implemented in the “Adding LoginBlock.cls to the Toolbox” section on page 7–16.

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

<table>
<thead>
<tr>
<th>Item</th>
<th>Property</th>
<th>Settings&gt;AppearancesLarge&gt;Appearance&gt;Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>[0]</td>
<td>styles\FlatNature.isl</td>
<td>Green style_green.png</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[1]</td>
<td>styles\TheBlues.isl</td>
<td>Blue style_blue.png</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[2]</td>
<td>styles\Red_No_pill.isl</td>
<td>Red style_red.png</td>
</tr>
</tbody>
</table>


The value in the key field is passed as a parameter to an event handler when one of the items is selected. See the “Subscribing to events and adding handlers” section on page 7–12 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 the “Adding LoginDlg.cls to the project” section on page 7–18.

At this point, LeftBar.cls should look something like Figure 7–2.
Creating the Login Dialog

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:

   ```ABL
   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:

   ```ABL
   @VisualDesigner.
   METHOD PRIVATE VOID ultraexplorerbar1_ActiveItemChanged
     (INPUT sender AS System.Object,
     IF e:Item:Group:Index = 0 THEN
     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
     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.
   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. For example:

   ![Event Handler List]

   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.

To add Leftbar.cls to the Toolbox:

1. Right click on one of control groups in the Toolbox.

2. Select Add Control Group. For example:

![Toolbox with Add Control Group dialog]

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. For example:

![Add Controls dialog box](image)

**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” section on page 7–16 and the “Adding HelpButton.cls to the Toolbox” section on page 7–17 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:

![LoginBlock CLS](image)

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 the “Running LoginDlg.cls” section on page 7–23.

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. LoginBlock appears under My Controls.
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 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.
4. The remaining fields are optional, with the exception of Package root (required).
5. After setting appropriate values in the Properties view, click OK. OpenEdge Architect creates a new class file with the same name that you assigned to the control.

A design window opens, but it is not a WYSIWYG 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.

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 `openedge.tutorial.ui.IStyling`, `openedge.tutorial.ui.IUser`, `openedge.tutorial.ui.IHelpCallback` 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.
Adding LoginDlg.cls to the project

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:

```plaintext
```

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 should now look something like this:
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 (ValidateLogin 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 Resources view and select Open With→OpenEdge ABL Editor.

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

```abl
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.

   UNDO, THROW NEW Progress.Lang.AppError("METHOD NOT IMPLEMENTED").
END METHOD.

   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 ):  
  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.

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.
```

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.
```

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

```plaintext
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 startup.p, which contains a WAIT-FOR statement that delays the launching of the Sports window.

You can open startup.p in the ABL Editor to see how the running of the sample application is coded. The startup.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 startup.p:

1. Select startup.p in the Resources view. It is located in the top-level SampleApp folder.
2. Select Run → Run as → OpenEdge Application from the main menubar.

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 only modified 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 now have a sense of how to use the tools of the Visual Designer to add, arrange, and format various user interface elements. You should also have an idea of how to use properties and events to modify the behavior of forms and controls. Finally, you should know how to implement data binding.

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