## Contents

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

1. **Introducing Progress Dynamics** ............................... 1–1
   - Object repository .............................................. 1–4
   - Integrated development tools ................................. 1–4
   - Standard, reusable application components ................. 1–5
   - Built-in environment managers .............................. 1–6

2. **Setting Up a Tutorial Environment** ......................... 2–1
   - Creating tutorial working directories ...................... 2–2
   - Creating configuration files ................................. 2–2
   - Creating startup scripts and shortcuts .................... 2–3
     - Copying the shortcuts to a Desktop folder .............. 2–3
     - Copying the database scripts .............................. 2–4
     - Modifying the shortcuts ................................. 2–4
     - Modifying the database scripts .......................... 2–5
   - Creating a tutorial Repository ............................. 2–6
   - Creating the application database ......................... 2–10
     - Creating an empty database .............................. 2–11
     - Loading the database schema .............................. 2–12
     - Loading the data ......................................... 2–13
   - Adding an entry to the Windows Services file ............ 2–14
   - Setting your PROPATH ......................................... 2–15
     - Logging in to the framework tools ........................ 2–15
     - Using the Propath Editor ................................. 2–17

3. **Generating Initial Objects** ...................................... 3–1
   - Logging in to the framework tools .......................... 3–3
   - Using the AppBuilder with Progress Dynamics ............ 3–3
   - Connecting to the application database .................... 3–4
   - Creating a new product and modules ....................... 3–6
   - Importing tables as entities into the Repository ......... 3–8
   - Customizing the entity data .................................. 3–11
   - Generating application objects .............................. 3–16
   - Viewing your application objects ........................... 3–20
Figures

Figure 1–1: Progress Dynamics run-time architecture ......................... 1–2
Figure 4–1: Completed sample application windows ......................... 4–2
Tables

Table 2–1: Tutorial working directories ........................................ 2–2
Table 4–1: Order Selection Window properties ................................ 4–30
Table 5–1: Where to go from here ..................................................... 5–23
This Preface contains the following sections:

- Purpose
- Audience
- Organization
- Using this manual
- Typographical conventions
- Example procedures
- Third party acknowledgements
Purpose

This manual provides an overview of Progress Dynamics®, the repository-based OpenEdge application framework. It also provides an introduction to the process of building an application with Progress Dynamics tools.

Audience

This manual is designed for developers who are new to Progress Dynamics. The exercises in this manual do not require knowledge of the ABL (Advanced Business Language), OpenEdge development tools, or the Application Development Model (ADM).

Organization

Chapter 1, “Introducing Progress Dynamics”

Provides an overview of the purpose and features of Progress Dynamics.

Chapter 2, “Setting Up a Tutorial Environment”

Provides instructions for setting up a separate working environment and databases for the tutorial.

Chapter 3, “Generating Initial Objects”

Takes you through using the framework’s tools for automatically generating basic application objects for your application database.

Chapter 4, “Building the Sample Application”

Takes you through assembling the pregenerated objects into application windows.

Chapter 5, “Customizing the Application”

Introduces you to the possibilities of quickly customizing applications using the Progress Dynamics Managers.

Using this manual

Before using this manual, you must install the OpenEdge Studio. Progress Dynamics is included in a complete install of the OpenEdge Studio.

OpenEdge provides a special purpose programming language for building business applications. In the documentation, the formal name for this language is ABL (Advanced Business Language). With few exceptions, all keywords of the language appear in all UPPERCASE, using a font that is appropriate to the context. All other alphabetic language content appears in mixed case.

For the latest documentation updates see the OpenEdge Product Documentation category on PSDN http://www.psdn.com/library/kbcategory.jspa?categoryID=129.
References to ABL compiler and run-time features

ABL is both a compiled and interpreted language that executes in a run-time engine that the documentation refers to as the ABL Virtual Machine (AVM). When documentation refers to ABL source code compilation, it specifies ABL or the compiler as the actor that manages compile-time features of the language. When documentation refers to run-time behavior in an executing ABL program, it specifies the AVM as the actor that manages the specified run-time behavior in the program.

For example, these sentences refer to the ABL compiler’s allowance for parameter passing and the AVM’s possible response to that parameter passing at run time: “ABL allows you to pass a dynamic temp-table handle as a static temp-table parameter of a method. However, if at run time the passed dynamic temp-table schema does not match the schema of the static temp-table parameter, the AVM raises an error.” The following sentence refers to run-time actions that the AVM can perform using a particular ABL feature: “The ABL socket object handle allows the AVM to connect with other ABL and non-ABL sessions using TCP/IP sockets.”

References to ABL data types

ABL provides built-in data types, pre-defined class data types, and user-defined class data types. References to built-in data types follow these rules:

- Like most other keywords, references to specific built-in data types appear in all UPPERCASE, using a font that is appropriate to the context. No uppercase reference ever includes or implies any data type other than itself.

- Wherever integer appears, this is a reference to the INTEGER or INT64 data type.

- Wherever decimal appears, this is a reference to the DECIMAL data type.

- Wherever numeric appears, this is a reference to the INTEGER, INT64, or DECIMAL data type.

References to pre-defined class data types appear in mixed case with initial caps, for example, Progress.Lang.Object. References to user-defined class data types appear in mixed case, as specified for a given application example.

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>SMALL, BOLD CAPITAL LETTERS</td>
<td>Small, bold capital letters indicate OpenEdge® key functions and generic keyboard keys; for example, GET and CTRL.</td>
</tr>
</tbody>
</table>
Example procedures

This manual provides numerous example procedures that illustrate syntax and concepts. You can access the example files and details for installing the examples from the following locations:

- The Documentation and Samples CD that you received with your product.
- The OpenEdge Documentation page on PSDN:


OpenEdge messages

OpenEdge displays several types of messages to inform you of routine and unusual occurrences:

- **Execution messages** inform you of errors encountered while OpenEdge is running a procedure; for example, if OpenEdge cannot find a record with a specified index field value.

- **Compile messages** inform you of errors found while OpenEdge is reading and analyzing a procedure before running it; for example, if a procedure references a table name that is not defined in the database.
• **Startup messages** inform you of unusual conditions detected while OpenEdge is getting ready to execute; for example, if you entered an invalid startup parameter.

After displaying a message, OpenEdge proceeds in one of several ways:

• Continues execution, subject to the error-processing actions that you specify or that are assumed as part of the procedure. This is the most common action taken after execution messages.

• Returns to the Progress Procedure Editor, so you can correct an error in a procedure. This is the usual action taken after compiler messages.

• Halts processing of a procedure and returns immediately to the Procedure Editor. This does not happen often.

• Terminates the current session.

OpenEdge messages end with a message number in parentheses. In this example, the message number is 200:

```
** Unknown table name table. (200)
```

If you encounter an error that terminates OpenEdge, note the message number before restarting.

**Obtaining more information about OpenEdge messages**

On Windows platforms, use OpenEdge online help to obtain more information about OpenEdge messages. Many OpenEdge tools include the following Help menu options to provide information about messages:

• Choose **Help → Recent Messages** to display detailed descriptions of the most recent OpenEdge message and all other messages returned in the current session.

• Choose **Help → Messages** and then enter the message number to display a description of a specific OpenEdge message.

• In the Procedure Editor, press the **HELP** key or **F1**.

**Third party acknowledgements**


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 software developed by Vermont Creative Software. Copyright © 1988-1991 by Vermont Creative Software.

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 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 the RSA Data Security, Inc. MD5 Message-Digest Algorithm. Copyright ©1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.

OpenEdge includes 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 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 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,
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 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 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 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 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 Sonic software which includes a version of the Saxon XSLT and XQuery Processor from Saxonica Limited that has been modified by Progress Software Corporation. The contents of the Saxon source code and the modified source code file (Configuration.java) 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 http://www.mozilla.org/MPL/ and a copy of the license (MPL-1.0.html) can also be found in the installation directory, in the Docs7.5/third_party_licenses folder, along with a copy of the modified code (Configuration.java); and a description of the modifications can be found in the Progress SonicMQ and Progress Sonic ESB v7.5 README files. 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 The Saxon XSLT and XQuery Processor from Saxonica Limited. The Initial Developer of the Original Code is Michael Kay (http://www.saxonica.com/products.html). Portions created by Michael Kay are Copyright © 2001-2005. All rights reserved. Portions created by Progress Software Corporation are Copyright © 2007. All rights reserved.

OpenEdge 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 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 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 the University Corporation for Advanced Internet Development http://www.ucaid.edu Internet2 Project. Copyright © 2002 University Corporation for Advanced Internet Development, Inc. All rights reserved. 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 DataDirect products for the Microsoft SQL Server database which contains a licensed implementation of the Microsoft TDS Protocol.

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.
Introducing Progress Dynamics

The Progress Dynamics® framework is a comprehensive, repository-based environment for OpenEdge® Studio developers who are building new applications, or who need to migrate existing applications to take advantage of new OpenEdge technologies. A Progress Dynamics application is built up of many components of different types. Some of these provide support for the user interface, such as windows, browsers, viewers, toolbars, and tab folders. Others are procedural objects that define the business logic of an application. Many application components are dynamic or data-driven objects, created at run time using data in the Dynamics Repository. Progress Dynamics is to some extent based on the OpenEdge Application Development Model (ADM) and OpenEdge SmartObjects™, which provide a basis for defining and combining standard components. Progress Dynamics extends the ADM and adds even more capabilities to the OpenEdge development environment.

The Progress Dynamics Repository stores data for a wide variety of purposes. Most application objects are represented as data stored in the Repository, so you do not need to create, compile, deploy, or maintain the source code for them. The framework includes programs that render these objects at run time from the data in the Repository.

Progress Dynamics is designed to run in a distributed, n-tier environment, with the visual portion of the application running in either a ABL (Advanced Business Language) client session or some other client type, without a local database connection. Business logic runs on one or many AppServers™ for OpenEdge applications where you can locate the repository database and the application database, maximizing efficiency of access to the database.

Progress Dynamics access to the AppServer is always stateless, so that a small pool of AppServer sessions can support a large number of clients.
To support this stateless AppServer access, Dynamics includes a number of Environment Managers to handle various aspects of the application and its environment. Each of these procedures runs as both a client-side process and a server-side process. The server-side manager maintains one or more repository database tables on the server and provides data to client sessions as needed. Data is cached on the client for maximum efficiency and returned to the server when necessary to allow the repository database to provide persistent storage for all data related to the running of the application.

Figure 1–1 shows a diagram of the Progress Dynamics run-time architecture and how the different components are related.

**Figure 1–1: Progress Dynamics run-time architecture**

The components that make up Progress Dynamics are described in the following sections:

- **Object repository**
- **Integrated development tools**
- **Standard, reusable application components**
- **Built-in environment managers**
Progress Dynamics combines these elements with a best practices approach that allows you to take full advantage of the framework’s capabilities. The best practice recommendations lead you without being unnecessarily restrictive. This approach provides guidelines for:

- Designing the application databases, business logic, and UI.
- Separating application business logic from your UI logic.
- Using the Progress Dynamics tools to build application components.
- Accessing Progress Dynamics features, such as message handling and UI interaction, in your code.

For an existing application that you migrate to the Progress Dynamics framework, these best practices provide the design and implementation guidelines against which you can assess how to prepare your application for migration. The more compliant an existing application is with Progress Dynamics guidelines, the easier it is to migrate. In keeping with the flexible scope of Progress Dynamics, you can still migrate an existing application using a subset of these guidelines.

In particular, the separation of business logic from the UI logic is essential to provide your application with the maximum deployment flexibility. Some OpenEdge deployment models (for example, OpenEdge WebClient™) do not work at all without this separation of your application logic. Progress Dynamics provides additional support to help you distribute your application logic appropriately.
Object repository

The heart of Progress Dynamics is its object repository. The Progress Dynamics Repository consists of an OpenEdge database named `icfdb` whose schema is defined to store information about an OpenEdge application. This includes development and run-time information about the application that is completely separate from the data stored in your application database.

Progress Dynamics relies on the data you provide in the Repository to specify the user interface (UI), design- and run-time configuration, and other features of an OpenEdge application. It is the Repository, and the toolset that accesses it, that allows you to automatically generate many application objects based on the schema of an application database. Such objects are known as dynamic objects because their implementation is determined at run time using data stored for the object instance in the Repository.

While the main purpose of the Repository is to store the run-time properties for dynamic objects, it can also store references to static objects that you want to include in your application. Static objects are objects defined at compile time. These static objects can include Progress Dynamics versions of ADM objects or any other procedures that you include in your application.

In addition to application objects of various types, the Repository stores a variety of additional information about an application, including:

- Configuration management data for both development and deployment.
- User profile data for configuring the application to users’ preferences.
- Automatic database auditing and comment information.
- Customizeable sequence and reference number generation.
- Security data.
- Localization data.
- Online help integration data.
- Multi-media data.

Integrated development tools

The Progress Dynamics toolset includes:

- An Application Development Environment (ADE).
- Additional tools and utilities to facilitate application design, migration, and deployment.

The heart of the Progress Dynamics ADE is the AppBuilder, which supports the development of both dynamic objects and traditional static ABL objects. Support for Progress Dynamics objects includes the ability to automatically generate a standard set of data and UI objects directly from the schema of your application database. Progress Dynamics also allows you to edit the generated Repository object data to customize Progress Dynamics objects for the specific requirements of your application.
Additional enhancements to the ADE allow you to manage:

- Administrative features such as application configuration and security.
- Application deployment options available with or without the use of Roundtable® TSMS™, a third-party source code management system designed for OpenEdge developers.

In addition, Progress Dynamics provides:

- A utility to help migrate ADM static objects to Progress Dynamics objects.
- A template that you can use with ERwin®, a third-party database design program, to design and generate OpenEdge database schemas with Progress Dynamics-compliant standards and built-in referential integrity validation.

**Standard, reusable application components**

Progress Dynamics supports a variety of application components with standard features that require no programming to implement. For example, you can use the predefined Progress Dynamics menu structure and toolbars to provide table navigation and maintenance. These menus and toolbars then become available for use by any appropriate object of your application.

The user interface (UI) components all provide a consistent and predictable look and feel for any application completely built using Progress Dynamics objects. However, you can also create customized objects and layouts that you can reuse throughout your application.

Progress Dynamics provides a rich variety of static and dynamic objects to support most functions required for an application. Dynamics stores the run-time properties for dynamic objects in the Repository.

Progress Dynamics objects fall into two large categories:

- **Data objects** — Handle transactions on data and provide a structure for including business logic.
- **UI objects** — Visualize data and control application functionality.

**Data objects**

Data objects include:

- **OpenEdge SmartDataObject™ (SDO)** — Handles transactions on data in a single database table. SDOs can be either static or dynamic objects.
- **OpenEdge SmartBusinessObject™ (SBO)** — Handles transactions on data in multiple database tables and serves as a container for multiple SDOs. SBOs can be either static or dynamic objects.
- **Data Logic procedure** — Provides a structure to specify business logic for the data managed by an associated SDO or SBO.
UI objects

Progress Dynamics UI objects include the following basic types:

- **Window Container Object** — Includes static and dynamic windows and folder objects. It can contain most other objects.
- **Field Object** — Includes OpenEdge SmartDataFields™, DataFields, and calculated fields.
- **Dynamic Lookup** — Presents a list of valid choices for a field value in the form of a browse window.
- **Dynamic Combo** — Presents a list of valid choices for a field value in the form of a combo box.
- **OpenEdge SmartDataViewer™ (SDV)** — Visualizes and accepts user changes of data in a single record or view. SDVs can be either static or dynamic objects.
- **OpenEdge SmartDataBrowser™ (SDB)** — Visualizes and accepts user changes of data in multiple records or views. SDBs can be either static or dynamic objects.

Built-in environment managers

While the Progress Dynamics Repository provides the storage for dynamic objects and the data for all other application features, the Progress Dynamics managers provide the engine for managing these objects and features in your application. These managers support virtually all Progress Dynamics functions, from application development and deployment to live execution of the application itself. Each manager is implemented as a persistent procedure, and the framework instantiates them in a prescribed order, depending on the specified configuration.

The common characteristic of all Progress Dynamics managers is that they run on both the client and the AppServer in a distributed environment, with the client-side version serving as a proxy to communicate with its counterpart running on the AppServer. In a stand-alone or client/server (database server) configuration, these managers run completely on the client.

Progress Dynamics ships with the following standard managers:

- Configuration File Manager.
- Connection Manager.
- Service Type Managers for the AppServer, database, JMS, and Web services.
- Customization Manager.
- Profile Manager.
- General Manager.
- Localization Manager.
- Referential Integrity Manager.
- Repository Manager.
• Repository Design Manager.
• Security Manager.
• Session Manager.
• User Interface Manager.
• Web Request Manager.
Setting Up a Tutorial Environment

This chapter provides instructions for setting up a separate Progress Dynamics environment for
the tutorial. Setting up a complete development environment includes creating databases,
loading them with data, copying and modifying configuration files, and creating startup scripts
and shortcuts. Working through this process helps you understand how the framework
components fit together and might give you insights about setting up development
environments for your own work.

Before you set up your tutorial development environment, you must install and configure the
OpenEdge™ Studio, which includes Progress Dynamics. See OpenEdge Getting Started: 
Installation and Configuration for complete instructions.

Note: Sample files are included in the Documentation and Samples CD that you received
with your product. For more information, see the “Example procedures” section on

This chapter covers the following topics:

- Creating tutorial working directories
- Creating configuration files
- Creating startup scripts and shortcuts
- Creating a tutorial Repository
- Creating the application database
- Adding an entry to the Windows Services file
- Setting your PROPATH
Creating tutorial working directories

The tutorial assumes that files are stored in certain directories. Table 2–1 lists the directories that you should create under your Progress Dynamics working directory.

Table 2–1:  Tutorial working directories

<table>
<thead>
<tr>
<th>Directory</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>\Tutorial</td>
<td>Main directory for tutorial files.</td>
</tr>
<tr>
<td>\Tutorial\modules</td>
<td>Tutorial product module objects.</td>
</tr>
<tr>
<td>\Tutorial\databases</td>
<td>Tutorial databases.</td>
</tr>
<tr>
<td>\Tutorial\databases\icfdb</td>
<td>Tutorial Repository database.</td>
</tr>
<tr>
<td>\Tutorial\databases\dynsports</td>
<td>Tutorial application database.</td>
</tr>
</tbody>
</table>

Creating configuration files

Progress Dynamics uses a number of configuration files that you can customize for different sessions. Before starting the tutorial, you should create copies of these files in your \<wrk>\Tutorial directory, where <wrk> is your Progress Dynamics working directory. You can then modify these files to customize the framework for your tutorial session, without affecting your installed version of Progress Dynamics.

icfconfig.xml

The icfconfig.xml configuration file tells the framework how to start and run individual sessions, including which managers to start, which startup procedures or objects to use, and other session characteristics. Since the configuration file uses logical and relative paths, the copy that you use to start the tutorial session should be in the same directory structure where your tutorial Repository and application database are stored.

To create a tutorial configuration file, copy <install-dir>\gui\dynamics\icfconfig.xml to \<wrk>\Tutorial, where <install-dir> is your OpenEdge installation directory and <wrk> is your Progress Dynamics working directory.

Later in the tutorial, you customize the icfconfig.xml file by creating a session type for running the sample application. For now, the standard configuration file contains the necessary information to start a development session for the tutorial.
Creating startup scripts and shortcuts

**icf.ini**

The `icf.ini` file contains initialization data such as color, fonts, and window sizes. It also sets the initial PROPATH, the list of directories where the session searches for files and databases, for the session.

To create a tutorial initialization file, copy `<install-dir>\bin\icf.ini` to `<wrk>Tutorial`.

**icf.pf**

The `icf.pf` file contains startup parameter information.

To create a tutorial startup file, copy `<install-dir>\icf.pf` to `<wrk>Tutorial`.

---

**Creating startup scripts and shortcuts**

Next, you must copy and modify the startup scripts and shortcuts for your working Progress Dynamics environment to launch the tutorial environment.

**Copying the shortcuts to a Desktop folder**

To avoid confusion with your working Progress Dynamics environment, create a Desktop folder for the tutorial and place your modified shortcuts there.

To create a Desktop folder:

1. Right-click on your Windows desktop and choose **New** → **Folder**.
2. Type **Progress Dynamics Tutorial** as the folder’s name.
3. Open the folder.
4. Choose **Start** → **Programs** → **OpenEdge** → **Progress Dynamics**.

**Note:** This procedure assumes you chose to install the OpenEdge shortcuts in the default location.

5. Copy the following shortcuts in turn and paste them into the Desktop folder:
   - **Dynamics Development**.
   - **Start Dynamics DB Servers**.
   - **Stop Dynamics DB Servers**.
Copying the database scripts

Your Progress Dynamics installation includes scripts that start and stop the Progress Dynamics Repository. Copies of these scripts can be modified to start and stop your Repository for this tutorial.

To create tutorial database scripts, copy startdbs.bat and stopdbs.bat from your <install-dir>\bin directory to your <wrk>Tutorial directory, where <install-dir> is your OpenEdge install directory and <wrk> is your Progress Dynamics working directory.

Modifying the shortcuts

Next, you modify the original scripts and shortcuts to start your tutorial environment, rather than your standard work environment.

To modify the development startup shortcut:

1. Right-click the Dynamics Development shortcut and choose Properties on the pop-up menu.
2. On the General tab, type Dynamics Tutorial Development as its name.
3. On the Shortcut tab, type the following in the Target field:

   <install-dir>\bin\prowin32.exe -p icfstart.p -pf icf.pf -ini icf.ini -icfparam ICFSESSTYPE=ICFDEV,ICFCONFIG=icfconfig.xml

   Where <install-dir> is your OpenEdge installation directory.

   **Note:** The ICFDEV session type contains the necessary configuration to start up Progress Dynamics in development mode locally. You will learn about session types and controlling your application environment with them later in the tutorial.

4. Type “<wrk>Tutorial” in the Start in field, where <wrk> is your Progress Dynamics working directory.

   **Note:** The quotation marks prevent problems if there are spaces in the directory names.

5. Click OK.

To modify the development server startup shortcut:

1. Select the Start Dynamics DB Servers shortcut and choose Properties on the pop-up menu.
2. On the General tab, type Start Tutorial DB Servers as its name.
3. On the Shortcut tab, type startdbs.bat in the Target field.
Creating startup scripts and shortcuts

**Note:** Because you copied the files to your working directory, you do not need the full path to the file.

4. Type “<wrk>Tutorial” in the **Start in** field.
5. Click **OK**.

**To modify the database server shutdown shortcut:**

1. Select the **Stop Dynamics DB Servers** shortcut and choose **Properties** on the pop-up menu.
2. On the **General** tab, type **Stop Tutorial DB Servers** as its name.
3. On the **Shortcut** tab, type **stopdbs.bat** in the **Target** field.
4. Type “<wrk>Tutorial” in the **Start in** field.
5. Click **OK**.

**Modifying the database scripts**

The database scripts that you copied must be modified for the tutorial. They point to the Repository for your working version of Progress Dynamics, but you want to work in a separate Repository for the tutorial.

You can also use these scripts to manage your application database. By adding another line to the script, you can start a server for the tutorial’s application database. Later in the tutorial, you modify the framework to automatically connect to this server when you start your session.

**To modify the database scripts:**

1. Open **startdbs.bat** in Notepad.
2. Find the following line:

   ```
call proserve "<wrk>databases\icfdb\icfdb" -S icfdb
   ```

   and replace it with these lines:

   ```
call proserve "<wrk\Tutorial\databases\icfdb\icfdb" -S icfdb
call proserve "<wrk\Tutorial\databases\dysports\dysports"
   -S dysports
   ```
3. Save your changes and close the file.

4. Open stopdbs.bat in Notepad.

5. Find the following line:

   ```
   call proshut -by "<wrk>\databases\icfdb\icfdb"
   ```

   and replace it with these lines:

   ```
   call proshut -by "<wrk>\Tutorial\databases\icfdb\icfdb"
   call proshut -by "<wrk>\Tutorial\databases\dynsports\dynsports"
   ```

6. Save your changes and close the file.

   **Note:** If the scripts do not work properly when you run them, check that your DLC environment variable is set properly.

---

### Creating a tutorial Repository

Progress Dynamics recommends that you use a separate Repository to develop each of your applications. This practice makes it easier to manage the application’s components and reduces the application’s footprint. Progress Dynamics also recommends that each Repository have a unique site number. The site number relates objects to a certain Repository as part of a unique object identifier. The unique object identifier reduces conflicts when objects from multiple Repositories are combined.

Setting up a functional Repository requires that many steps be performed in the correct order. Because of this complexity, you should only create a Repository using the Progress Dynamics Configuration Utility (DCU).
Creating a tutorial Repository

To create a Repository to use with the tutorial:

1. Choose **Start**→**Programs**→**OpenEdge**→**Progress Dynamics**→**Dynamics Configuration Utility** to launch the DCU, as shown:

2. Click **Next**. The **Installation Paths** page appears:

The DCU attempts to derive your Progress Dynamics installation path automatically. If it does not derive the correct path, you can use the **Browse** button to set the correct path.

3. Click **Next**. The **Working Paths** page appears. The DCU attempts to derive the working paths automatically. If it does not derive the correct paths, you can use the **Browse** button to set the correct paths.
4. Set the **Working Path** field to `<wrk>\Tutorial`, where `<wrk>` is your Progress Dynamics working directory, using the **Browse** button, as shown:

5. Click **Next**. The **ICFDB Parameters** page appears.

6. Set the **ICFDB Path** field to `<wrk>\Tutorial\databases\icfdb\icfdb.db` using the **Browse** button.

7. Set the **Data path** field to `<install-dir>\src\dynamics\db\icf\dump`, if necessary. This is the location of the files that contain the basic data that has to be loaded into each new Repository.

8. Select the **Create new ICFDB Database** toggle box, as shown:
9. Click Next. The ICFDB Site Number page appears.

Every Progress Dynamics Repository database should have a unique site number assigned to it. The site number is combined with a database sequence value to form the key value for every record in the Repository database. The unique site number of each Repository database enables dynamic object definitions from several Repositories to be combined. The unique site numbers prevent conflicts in the key values that organize the records that make up the dynamic object definitions.

The site number 1040 has been reserved for *OpenEdge Development: Progress Dynamics Getting Started* tutorial users.

10. Type **1040** for the Site Number. You can accept the default values for the other fields, as shown:

11. Click Next. The Processing Status page appears:
12. Click **Process** to start the creation of the tutorial Repository. The following message appears when the process is complete:

![Information dialog box](image)

13. Click **OK** to clear the message dialog box.

14. Click **Finish** on the **Complete** page to close the DCU, as shown:

![Progress Dynamics Configuration Utility - Complete](image)

Your tutorial Repository is now ready to use.

**Creating the application database**

The next step in setting up your development environment is to create the application database. During the tutorial, you define application screens and menus for the DynSports database. This database was designed using the ERwin data modeling tool. Progress Dynamics comes with an ERwin template for creating databases with tables and fields that follow the Progress Dynamics naming conventions. ERwin has generated all the database triggers for the DynSports database.

**Note:** Because your Progress Dynamics tutorial session is not fully configured, the following procedure is done through your main OpenEdge installation.
Creating an empty database

To create a DynSports database to use with the tutorial:

1. Choose Start → Programs → OpenEdge → Data Administration to launch the Data Administration tool, as shown:

2. Select Database → Create. The Create Database window appears.

3. Type `<wrk>\Tutorial\databases\dynsports\dynsports` in the New Physical Database Name field, where `<wrk>` is your Progress Dynamics working directory, and start with An EMPTY Database, as shown:
4. Click OK. The Connect Database dialog box appears when the database is ready.

Note that the Physical Name is set to your new database and the Logical Name has defaulted to dynosports. The logical name is used to resolve ambiguous database references. When a procedure is compiled against a database, the logical database name is stored in the procedure’s object code. When the procedure executes, its database name references must match the logical database name of a connected database, as shown:

5. Click OK to connect the database.

Loading the database schema

Next, you must load the data definitions for the application database.

To load the DynSports database schema:

1. Choose Admin → Load Data and Definitions → Data Definitions (.df file). The Load Data Definitions window appears.

2. Select <install-dir>Tutorial\dynamics\dynsports\db\dfd\dynsportsdbfull.df, where <install-dir> is your OpenEdge install directory, as the Input File, as shown:

Note: This file contains the complete schema definitions for the dynosports database.

3. Leave the other settings as they are and click OK.

4. Click OK to clear the load completion message.
Loading the data

Next, you must load the table contents for the application database.

To load the DynSports table contents:

1. Choose Admin → Load Data and Definitions → Table Contents (.d file). The Select Tables window appears. Click Select Some, as shown:

2. Select all the tables using the wild card (*), and click OK:

3. The Load Data Contents for all Tables dialog box appears. Type \<install-dir>\Tutorial\dynamics\dynsports\db\dump as the Input Directory, where \<install-dir> is your OpenEdge install directory, as shown:

   Note: This directory contains the .d files that contain the table data.
4. Deselect the **Include LOB** toggle box. This is the recommended practice when you are not specifying an LOB directory.

5. Leave the other settings as they are and click **OK**. The load process can take some time.

6. Click **OK** to clear the load completion message.

7. Choose **Database** → **Disconnect** to disconnect the database. Then exit the Data Administration tool.

**Copying the DynSports triggers**

In order for the trigger code to run correctly, it must be located in a subdirectory under the DynSports database. Your next task is to copy the triggers to the appropriate directory.

---

**To copy the DynSports triggers:**

1. In the Windows Explorer, browse to `<install-dir>Tutorial\dynamics\dynsports\trg`.
2. Copy the directory.
3. Paste a copy of the directory in `<wrk>Tutorial\databases\dynsports`.

**Adding an entry to the Windows Services file**

When you installed Progress Dynamics, you had to add entries for the NameServer (NS1) and Repository’s database server (icfdb) in the Windows Services file. You must add another line for your application database.

---

**Note:** Do not run another Progress Dynamics session at the same time that you are running the tutorial session. To run multiple sessions with different Repositories simultaneously, you would need to set up multiple entries in the Services file to give each Repository its own port.

---

**To add the entry to the Windows Services file:**

1. In a text editor, open your services file, located by default in your `C:\Windows\System32\drivers\etc\` directory.
2. Find the lines you created after installing the OpenEdge Studio for your NameServer and Repository database. For example:

   ```
   NS1 5162/upd
   icfdb 8000/tcp
   ```

3. Add a line for the DynSports database on a separate line. For example:

   ```
   dynsports 3500/tcp
   ```
Note: Make sure you choose a port number that is not already assigned to another service. If the startup script ever returns an error because the port is already in use, you must edit the file again to choose another port.

4. Save and close the file.

Your copy of the DynSports application database is now ready for use in the tutorial.

Setting your PROPATH

OpenEdge Studio sessions use a special path to search for files. The path can be stored in the environment variable called the PROPATH or it can be stored in the icf.ini file. To finish setting up your tutorial session, you must edit its PROPATH to point at several new directories.

The easiest way to edit a PROPATH is to use one of the PRO*Tools. The PRO*Tools are a set of utility applications that aid in developing and running applications in the OpenEdge Studio. You must start your tutorial session to access the PRO*Tools.

Logging in to the framework tools

Progress Dynamics has integrated user and security definitions that let you control access to the development tools and to your completed application. The initial installation of Progress Dynamics creates a standard administrator user. The administrator user has access to all functionality in the framework. The administrator has a user name of “admin” and has no password.

You can login to the framework and an application using a particular language. You can translate both the development tools and your finished application into any number of languages. You add some translations to the sample application and create a new user who uses the translations in Chapter 5, “Customizing the Application.”

You can also define login companies to represent any organizational entity or role. Using different login company definitions, you can customize the look of your application and security restrictions by organization as well as by user.

Progress Dynamics runs in different modes depending on the application configuration and context. In this tutorial, you use the framework in development mode. Development mode provides the correct tools and options to develop the sample application.
To start your tutorial environment:

1. Open the **Progress Dynamics Tutorial** folder in your Windows Desktop.

2. Double-click the **Start Tutorial DB Servers** shortcut.

3. Double-click the **Dynamics Tutorial Development** shortcut. The **Application Login** window appears:
4. Type **admin** as your **Login Name**, leave the **Password** empty, and click **OK**. The AppBuilder tools appear:

![AppBuilder screenshot](image)

**Using the Propath Editor**

The PROPATH for your Progress Dynamics session is defined in your *icf.ini* file. Because you copied the file from your installed version, almost all the directories you need for the tutorial are already listed in the PROPATH. You only need to add the locations of the DynSports database triggers and your working directories.
To add the directories to the tutorial PROPATH:

1. Click the PROPATH icon on the PRO*Tools window. The Propath Editor appears:

2. Click Add. The Add Directory from Propath dialog box appears.

3. Type `<wrk>Tutorial\databases` in the Directory field, where `<wrk>` is your Progress Dynamics working directory, as shown:

4. Click OK. The new directory is added to the top of the list in the Propath Editor.

5. Use the Move Down button to place it below the [current directory] entry.

6. Repeat Step 2 through Step 5 to add `<wrk>Tutorial` to the PROPATH, where `<wrk>` is your Progress Dynamics working directory.

7. Click OK to close the Propath Editor. The following dialog box appears asking if you want to save the changes in your startup file:

8. Click YES to save the new PROPATH to your tutorial’s icf.ini file.

9. Exit the session and shut down the database servers.

You have completed setting up the environment for the tutorial. In the next chapters, you import information from the application database, generate objects from it, and build your application windows from those objects.
Generating Initial Objects

After your Progress Dynamics development environment is properly set up, you can begin building an application. In the first stage of building an application with the framework, you import entity information from your application database and generate objects from that data. Afterwards, you can build an application by editing and combining these building blocks. The rapid generation of these starting building blocks is one of the advantages of developing in the framework.

This chapter provides instructions for importing information about the sample application database, DynSports, into your tutorial Repository. Then, it guides you through using the tools to generate objects from that data.

**Note:** Sample files are included in the Documentation and Samples CD that you received with your product. For more information, see the “Example procedures” section on page Preface–4.

To help you understand more of what the framework does, additional explanatory information is interspersed within the tutorial steps. For more detailed information on these general types of tasks, see *OpenEdge Development: Progress Dynamics Basic Development* and *OpenEdge Development: Progress Dynamics Advanced Development*.

This chapter covers the following topics:

- Logging in to the framework tools
- Using the AppBuilder with Progress Dynamics
- Connecting to the application database
Generating Initial Objects

- Creating a new product and modules
- Importing tables as entities into the Repository
- Customizing the entity data
- Generating application objects
- Viewing your application objects
Logging in to the framework tools

Progress Dynamics has integrated user and security definitions that let you control access to the development tools and to your completed application. The initial installation of Progress Dynamics creates a standard administrator user. The administrator user has access to all functionality in the framework. The administrator has a user name of “admin” and has no password. Later in the tutorial, you will create a new user and assign a password.

Progress Dynamics runs in different modes depending on the application configuration and context. In this tutorial, you use the framework in development mode. Development mode provides the correct tools and options to develop the sample application.

To start your tutorial environment:

1. Open the Progress Dynamics Tutorial folder in your Windows Desktop.
2. Double-click the Start Tutorial DB Servers shortcut.
3. Double-click the Dynamics Tutorial Development shortcut. The Application Login window appears:

![Application Login Window]

4. Type admin as your Login Name, leave the Password empty, and click OK. The AppBuilder windows appear.

Using the AppBuilder with Progress Dynamics

The OpenEdge AppBuilder is extended in Progress Dynamics sessions with additional menus and tools. The AppBuilder is the primary visual programming tool in the OpenEdge Studio. In Progress Dynamics, you create and assemble application components in the AppBuilder. The AppBuilder’s Section Editor makes it easier to write ABL (Advanced Business Language) business logic code to complement the user interface logic.

In case you are not familiar with them, this tutorial introduces you to some of the standard features of the AppBuilder as well as the new features of Progress Dynamics. If you want more information about a particular tool or dialog box, check the online help.

If you have worked with the AppBuilder before, notice the additions to the main window that support the Progress Dynamics framework. Most of the additions are on the new Build menu. You will use many of these tools during the tutorial.

If you are not familiar with the OpenEdge AppBuilder, it consists of three windows: the main window, the Object Palette, and the PRO*Tools. The Object Palette shows the various objects that you can add to your application windows. Clicking on an icon and then clicking inside a window automatically adds an instance of that object type to the window with all its basic...
controlling code. The PRO*Tools are a set of utility programs that you might need while developing your applications, as shown in the following figure:

Connecting to the application database

While building the sample application, you define application screens and menus for the DynSports database. This database was designed using the ERwin data modeling tool. Progress Dynamics comes with an ERwin template for creating databases with tables and fields that follow the Progress Dynamics naming conventions. ERwin has generated all the database triggers for the DynSports database.

You must connect to the DynSports database before starting to work on the tutorial. In Chapter 5, “Customizing the Application,” you build a session type into the Configuration Manager to automatically make the connection when the finished application starts. The standard ICFDEV session type already contains an entry for the Repository. When it reads that session type from the configuration file at startup, the Configuration Manager automatically orders the Repository to be connected.
To connect to the DynSports database:

1. From the AppBuilder main menu, select Tools → Database Connections. The Database Connections dialog box appears:

2. Click Connect.

3. Browse for the tutorial DynSports database, as shown:

4. Click Options>> and select the Multiple Users toggle box, as shown:

5. Click OK and then close the Database Connections dialog box.
Creating a new product and modules

To help organize the objects in your application, particularly for deployment, Progress Dynamics supports a hierarchy of Products and Product Modules. Using this hierarchy helps you organize and more easily locate your components. The products and product modules can describe any useful type of organization for your application. You must assign all dynamic application objects to a product module. For more information on using products and product modules, see the chapter on preparing to build application objects in *OpenEdge Development: Progress Dynamics Basic Development*.

To define a DynSports Product and its Product Modules:

1. From the AppBuilder main window, choose **Tools** → **Administration**. A separate **Administration** window appears:

   ![Administration Window]

   The Administration window provides access to tools for configuring and maintaining your application development environment. The tools include functions such as application structure, deployment, security, localization, and entity control. You will use some of these later in the tutorial.

2. Choose **Application** → **Product Control**. The **Product Control** window appears:

   ![Product Control Window]

3. Click **Add**. The **Product Maintenance** dialog box appears:

   ![Product Maintenance Dialog]

   For more information on generating initial objects, see the chapter on preparing to build application objects in *OpenEdge Development: Progress Dynamics Basic Development*. 
4. Set the following values in the **Product Maintenance** dialog box, and leave the default settings in the other fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product code</td>
<td>DynSports</td>
</tr>
<tr>
<td>Production description</td>
<td>Progress Dynamics sample app</td>
</tr>
<tr>
<td>Product installed</td>
<td>Selected</td>
</tr>
</tbody>
</table>

5. Click **Save**. The dialog box should look like the following:

6. Choose the **Product Module** tab, and click **Add**.

7. Set the following values on the tab:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product module code</td>
<td>ds-OE</td>
</tr>
<tr>
<td>Production module description</td>
<td>Order Entry module</td>
</tr>
<tr>
<td>Product module installed</td>
<td>Selected</td>
</tr>
<tr>
<td>Number of users</td>
<td>0</td>
</tr>
<tr>
<td>Relative path</td>
<td>modules/oe</td>
</tr>
</tbody>
</table>

The Relative Path directory is the subdirectory under your working directory where the framework saves physical objects such as static viewers, browses, and SDOs.
8. Click **Save**. The dialog box should look like the following:

9. Follow the procedure in **Step 6** through **Step 8** to add the following Product Modules to the DynSports product. Remember to select the **Product module installed** toggle box for all the modules. Set the following values on the tab:

<table>
<thead>
<tr>
<th>Product module code</th>
<th>Description</th>
<th>Relative path</th>
</tr>
</thead>
<tbody>
<tr>
<td>ds-EMP</td>
<td>Employee and Department information</td>
<td>modules/emp</td>
</tr>
<tr>
<td>ds-Entity</td>
<td>DynSports entity and datafield info</td>
<td>modules/entity</td>
</tr>
<tr>
<td>ds-General</td>
<td>Objects used across the sample app</td>
<td>modules/general</td>
</tr>
</tbody>
</table>

10. Choose the **Product Module List** tab to see your modules. Then, exit the **Product Maintenance** and **Product Control** windows.

**Importing tables as entities into the Repository**

The Progress Dynamics Repository stores several levels of data to describe application components. The first level holds descriptions of the tables in your application database, the fields in those tables, and a description of which fields should be used to generate default application components. In this section, you will import that application table information into the Repository.

In this section, you import *entities*, abstractions of the Repository tables, from the application database into the Repository. From this entity data, framework tools can use the structure of your data, your naming conventions, and other information to automatically generate objects more successfully.
To import and assign data to the ds-Entity Module:

1. Choose **System** → **Entity Import** in the **Administration** window. The **Entity Import** dialog box appears.

2. Select **dynsports** in the **Database** combo box, if necessary, as shown:

![Entity Import dialog box](image)

Note that the tables in the DynSports database start with a three-letter acronym (TLA) followed by an underscore. The first two letters identify the organizational structure, such as “AP” for the accounts payable module and “HR” for the human resources module. The third letter indicates the relative volatility of the data, for example, “C” for constant, “M” for master, “T” for transactional, “V” for raw, and “X” for cross-reference/many-to-many relationships. The framework uses its understanding of this naming convention in various ways. For more information on the naming conventions and their uses, see *OpenEdge Development: Progress Dynamics Basic Development*.

3. Choose **Select All** to select all the table names from the **Table Name** browse.

**Note:** For this application, you are only using a few tables in the DynSports database that relate to customer maintenance and order entry. The other tables are not used in the tutorial, but you might want them later to experiment or follow examples in the programming guides.
4. Set the **Prefix Length** to **4**. The prefix length tells the framework what, if any, standard prefix you use for table names. Setting this field to **4** follows the Progress Dynamics naming convention of a TLA followed by an underscore.

5. Set the **Field Name Separator** to an underscore ( _ ). If you specify an underscore or hyphen as the standard separator in field names, the framework automatically replaces the separator with a space while generating labels for the fields or creating descriptions for the table to use in messages.

6. Select **ds-Entity** in both **Product Module** combo boxes.

7. Leave all the other settings at their defaults. The dialog box should look like the following:

8. Click **Import**. The **Continue import** dialog box appears:
9. Click **Yes**. The import process takes a while. After it completes, a message dialog box appears confirming successful completion:

![Entity Memonic Import](image)

Dialog boxes like this come up during application development. They are examples of how all Progress Dynamics messages are handled. You can define (and translate) application messages in the Repository. When a message is displayed on the client, it comes up in this dialog box with the following tabs:

- Message summary.
- Message detail.
- System information.
- AppServer information (if connected to an AppServer).

The message dialog box also contains buttons that let you view a stack trace of the ABL procedure that generated the message, and send an e-mail message if your PC is correctly configured. The final button alternates the dialog box between full screen and normal modes.

10. Click **OK**, and exit the **Entity Import** window.

## Customizing the entity data

Once you have imported default data into your entity tables from the application database schema, you can customize that data as needed. You can change the fields, field formats, or the order in which the fields appear.

Part of the best practices recommendation for database design in Progress Dynamics is to include a unique key field on each table that is used only as a target for relationships. These are the object (_obj) fields. Generally, you do not want the object field that uniquely identifies a table to appear in visual objects for that table. You can eliminate them through the Entity Control tool.

By default, the Object Generator creates field labels by appending the table name to the field name. You might not want to see the table name appearing in every label. You can use the Entity Control to change the default label for a field.
To customize the entity data:

1. Choose **System**→**Entity Control**. The **Entity Control** dialog box appears:

![Entity Control dialog box]

2. Select **dynsports** in the **Database** combo box.

3. Double-click the **ARMCU** record. The **Entity Maintenance** dialog box appears.

4. Click **Modify record**.

5. Select the **customer_obj** record on the **Datafields** tab:

![Entity Maintenance dialog box]

6. Deselect the **Include in browsers** and **Include in viewers** toggle boxes.

The _obj fields are meant only for internal use. Their principal use on GUI objects is as a target for SDFs. Browsers do not support SDFs, so you should uncheck that box for all _obj fields. You are unlikely to want an SDF based off the same table for which you are generating a viewer, so you should always uncheck that box for the _obj field of the table with which you are working.
7. Click Save. The dialog box should look like this:

![Image of dialog box]

8. Select the login_company_obj record.

9. Deselect the Include in browsers toggle box. You want this field as an SDF target on viewers.

10. Set the Label for the field to Login company, and save your changes.

11. Select the salesrep_obj record.

12. Deselect the Include in browsers toggle box. You want this field as an SDF target on viewers.

13. Set the Label for the field to Sales rep, and save your changes.

14. Set the default labels for the other fields as follows:

<table>
<thead>
<tr>
<th>Field instance</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer_code</td>
<td>Customer code</td>
</tr>
<tr>
<td>customer_name</td>
<td>Customer name</td>
</tr>
<tr>
<td>customer_comments</td>
<td>Comments</td>
</tr>
<tr>
<td>customer_credit_limit</td>
<td>Credit limit</td>
</tr>
<tr>
<td>customer_balance</td>
<td>Balance</td>
</tr>
<tr>
<td>customer_discount</td>
<td>Discount</td>
</tr>
<tr>
<td>customer_terms</td>
<td>Terms</td>
</tr>
<tr>
<td>customer_fax</td>
<td>Fax</td>
</tr>
<tr>
<td>customer_email_address</td>
<td>Email</td>
</tr>
</tbody>
</table>
Generating Initial Objects

Notice that you also have access to the Dynamic properties sheet for each field in the Edit master section of this tab. If there are properties that you always set, setting them here might be easier than doing it after you generate the objects.

**Note:** You must save your changes after altering each individual record.

Now, you must make similar changes for the other entities needed in the tutorial.

To customize the remaining entries:

1. Select the ARMSR record in the Entity Control dialog box. The Entity Maintenance dialog box repopulates with the records for the new entity.
2. Select salesrep_obj on the Data Fields tab.
3. Deselect the Include in browsers and Include in viewers toggle boxes, and save your changes.
4. Select the ARTOL record in the Entity Control dialog box. The Entity Maintenance dialog box repopulates with the records for the new entity.
5. Select orderline_obj on the Data Fields tab.
6. Deselect the Include in browsers and Include in viewers toggle boxes, and save your changes.
7. Select the status_obj record.
8. Deselect the Include in browsers toggle box. You want this field as an SDF target on viewers.
9. Set the Label for the field to Status, and save your changes.
10. Select the item_obj record.
11. Deselect the Include in browsers toggle box. You want this field as an SDF target on viewers.
12. Set the Label for the field to Item, and save your changes.
13. Select the order_obj record.
14. Deselect the Include in browsers toggle box. You want this field as an SDF target on viewers.
15. Set the Label for the field to Order, and save your changes.
16. Set the default labels for the other fields as follows:

<table>
<thead>
<tr>
<th>Field instance</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>orderline_line_number</td>
<td>Line number</td>
</tr>
<tr>
<td>orderline_price</td>
<td>Price</td>
</tr>
<tr>
<td>orderline_qty</td>
<td>Quantity</td>
</tr>
<tr>
<td>orderline_discount</td>
<td>Discount</td>
</tr>
<tr>
<td>orderline_extended_price</td>
<td>Extended price</td>
</tr>
</tbody>
</table>

17. Select the ARTOR record in the Entity Control dialog box. The Entity Maintenance dialog box repopulates with the records for the new entity.


19. Deselect the Include in browsers and Include in viewers toggle boxes, and save your changes.

20. Select the status_obj record.

21. Deselect the Include in browsers toggle box. You want this field as an SDF target on viewers.

22. Set the Label for the field to Status, and save your changes.

23. Select the customer_obj record.

24. Deselect the Include in browsers toggle box. You want this field as an SDF target on viewers.

25. Set the Label for the field to Customer, and save your changes.

26. Select the salesrep_obj record.

27. Deselect the Include in browsers toggle box. You want this field as an SDF target on viewers.

28. Set the Label for the field to Sales rep, and save your changes.

29. Select the warehouse_obj record.

30. Deselect the Include in browsers toggle box. You want this field as an SDF target on viewers.

31. Set the Label for the field to Warehouse, and save your changes.
32. Set the default labels for the other fields as follows:

<table>
<thead>
<tr>
<th>Field instance</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>order_code</td>
<td>Order code</td>
</tr>
<tr>
<td>order_date</td>
<td>Order date</td>
</tr>
<tr>
<td>order_promise_date</td>
<td>Promise date</td>
</tr>
<tr>
<td>order_ship_date</td>
<td>Ship date</td>
</tr>
<tr>
<td>order_carrier</td>
<td>Carrier</td>
</tr>
<tr>
<td>order_instructions</td>
<td>Instructions</td>
</tr>
<tr>
<td>order_po</td>
<td>PO</td>
</tr>
<tr>
<td>order_terms</td>
<td>Terms</td>
</tr>
<tr>
<td>order_creditcard</td>
<td>Credit card</td>
</tr>
<tr>
<td>order_incoming</td>
<td>Incoming</td>
</tr>
</tbody>
</table>

33. Exit the Entity Maintenance and Entity Control dialog boxes, then restart your Progress Dynamics session.

Before generating your application objects, you must clear the framework’s cached data. This ensures that the Object Generator has access to the changes you made in the entity data.

Remember to reconnect the application database after restarting your session.

Generating application objects

The next level of data maintained by the Repository describes application grouping components, objects that define groups of fields that are used together. This level includes the following types of objects:

- **SmartDataObject (SDO)** — Moves data from the database on the server to the user session on the client where it can be displayed and updated.

- **Browse** — Lets you scroll through, select, and update records from an SDO’s database query.

- **Viewer** — Lets you display and update details for a record in a frame.
Generating application objects

The Object Generator creates all these objects from the entities you import from your application database. You can use the objects as generated or modify them to your needs. The Object Generator names the objects with their table name (or table dump name if this is shorter) and a suffix _fullo for SDOs, _fullb for Browses, and _viewv for Viewers. You can modify this naming convention and other defaults for your own applications if you wish.

To generate objects from the imported data:

1. From the AppBuilder main menu, choose Build→Object Generator. The Object Generator window appears. By default, the last database you used, DynSports, is selected in the Database combo box, as shown:

2. Select the DataObjects, Browses, and Viewers toggle boxes in the Generate Objects section. Since you have already created datafields in the Entity Import phase, deselect the Datafields toggle box. The corresponding tabs in the window are enabled.

3. Select the arm_customer, arm_salesrep, art_order, art_orderline, and ivm_item table names in the browse.

4. Choose the DataObjects tab. The information on this tab is used to create dynamic SDOs and data logic procedures (DLPs). The generated DLP has standard validation code for the entity. For your own applications, you should review this code and edit it to fit your needs. Generally, you put your business logic in the DLPs.
5. Set the **Root folder** to `<wrk>/Tutorial`, where `<wrk>` is your Progress Dynamics working directory. The root folder is the directory where you want any generated physical objects saved.

**Note:** Your root folder must be in your PROPATH. Some of the Progress Dynamics tools might not run if the directory is not in your PROPATH.

6. Select **ds-OE** from the **Product Module** combo box in the **DataObject** section. The **Relative Path** is automatically set to the directory you chose when you defined the Product Module. The fields in the **DataLogic Procedure** section also update automatically.

7. Select **Astra** for the **AppServer partition**.

8. Deselect the **Follow Joins** option. If your application database has internal joins (like the DynSports database), you should deselect this option. If you leave it selected, Progress Dynamics generates a complex query to follow the internal joins that might result in filtering that returns no records. You can leave the other options in this window at their default settings, as shown:

![Object Generator](image)

9. Choose the **Viewers** tab.
10. Change the **Number of Fields** to 24 and the **Maximum Fields per Column** to 8, as shown:

![Object Generator Interface]

**Note:** The default values are larger than you need in the tutorial. Viewers with a maximum of three columns of eight fields work better.

11. Click **Start** to begin generating your SDOs, browses, and viewers. A message dialog box appears warning you that datafields will not be generated, as shown:

![Message Dialogue]

This is not a problem, since you generated data fields during the Entity Import phase.

12. Click **Yes**. The object generation starts. As this proceeds, the tool displays information in the **Logging** tab. This process can take several minutes. When the object generation process completes, a message dialog box appears:

![Object Generation Complete]

13. Click **OK** to clear the message dialog box. You can export the object generation results to a log file by specifying an **Export filename** and choosing the **Export Log** button.

14. Check that the process ran successfully by expanding the nodes in the TreeView. If there is a **Completed** node for each step, the process ran successfully, as shown:

![Screen capture showing the TreeView of the Object Generator with nodes expanded, including a Completed node for each step.]

15. Exit the **Object Generator**.

For extra practice, you can complete this procedure for the tables in the **ds-EMP** product module: **hrm_benefits**, **hrm_department**, **hrm_employee**, and **hrm_family**. While these tables are not used for this tutorial, you might need them to follow examples in the programming guides. There are no objects to generate for the General module, because it is not associated with a set of tables in the DynSports database.

### Viewing your application objects

The Object Generator created several kinds of static files and dynamic objects for you. The SDOs, browses, and viewers are fully dynamic objects defined as records in the Repository. The AppBuilder lets you view and edit the dynamic objects using different tools, each appropriate to the type of object.

The static files, an include and two DLPs, associated with each SDO are stored in the `\dynsports\oe` and `\dynsports\emp` directories. The SDO uses the include file to construct the internal temp-tables that pass data between the server and client. You should put any code for the business logic associated with the SDO’s data in the DLPs.
Viewing an SDO

One of the SDOs that you use repeatedly in this tutorial is the Customer SDO.

To view the Customer SDO:

1. Click Open Object on the AppBuilder toolbar.

2. Select the ds-OE module from the Module combo box. This filters out all Repository objects except those in the Order Entry module.

3. Select DynSDO (Dynamic SmartDataObject) from the Type combo box.

4. Select armcufullo in the browse. Note that the object filenames are made by combining the tables’ entity names with the suffixes specified in the Object Generator, as shown:

![Object Browse](image)

**Note:** If you type the first few letters of a filename in the Object Filename field, the browse filters on objects that begin with the letters you typed.

5. Double-click on its row in the browse or click Open. A design window for the SDO appears:
6. Double-click inside the design window to look at the query the Object Generator created for you, as shown:

With the **Query** button, you can edit the query. For example, if there is a description field that you always want to have available when using the Customer table, such as the RepName field in the SalesRep table, you can edit the query to add a join to the table that contains it.

With the **Fields** button, you can edit the list of fields defined in the SDO’s temp-table. By default for generated SDOs, all fields are included in the list. Also by default, all fields, except the Progress Dynamics Object ID fields, are marked as updateable. To reduce the amount of data sent across the AppServer connection, you might remove fields that are never directly viewed or updated in the client part of your application.

The **DLP** field shows the location of the data logic procedure for the SDO. This file is where business logic like a validation hook procedure goes.

**Note:** Do not make any changes to the query at this time.

7. Click **Cancel** and then close the SDO’s design window.
Viewing a dynamic Browse

You can also open a design window for a dynamic Browse through the Open Object tool.

To access the design window:

1. Click Open Object, and select DynBrow (Dynamic SmartDataBrowser) in the Type combo box, as shown:

   ![Design window for DynBrow](image)

   **Note:** The AppBuilder retained the Module filter on the ds-OE module. The filtering remains until you change it or exit the session.

2. Double-click armcufullb, the generated browse for the Customer table. The design window for the dynamic Browse appears. Unlike the design window for a nonvisual object like an SDO, the design window for the dynamic Browse allows you to change its appearance, as shown:
3. Double-click inside the window to bring up its property sheet, as shown:

You can choose the **Fields** button to edit the field list or specify instance attributes like labels, width, and format. The dynamic objects of each class use common code to create an object from Repository data. If you change that common code, you would change the behavior of all objects of that class. To customize the behavior of a specific object, you can define a **Custom Super Procedure** and associate the procedure with the object.

In the **Folder Window to Launch** field, you can enter the name of a default logical container to launch when the user double-clicks on a row in the browse. This is normally a maintenance window for the table. Later, you will specify a launch container when you put the browse into a window.

4. Close the property sheet and then the design window.

You have now created the building blocks for the sample application. In the next chapter, you use these objects to build the application windows.
This chapter covers building a sample application from the objects you just generated. The application conforms to the Progress Dynamics best practices, allowing you to create all the elements of the application without having to write any ABL (Advanced Business Language) code to customize its behavior. However, you can customize or extend every aspect of the framework and its components as needed to satisfy the requirements of your applications.

To help you understand more of what the framework does, explanatory information is interspersed within the tutorial steps. For detailed information on these general tasks, see OpenEdge Development: Progress Dynamics Basic Development and OpenEdge Development: Progress Dynamics Advanced Development.

As you work through this tutorial, you will be performing the following:

- Editing your application objects
- Creating browse windows
- Creating folder windows
- Creating an order maintenance window
- Creating a main menu window for your application
- Using the Toolbar and Menu Designer
- Running the completed application
Figure 4–1 shows a few screen shots of the completed application to give you an idea of what you are building.

**Figure 4–1: Completed sample application windows**

**Note:** Remember that you must reconnect to the DynSports database each time you restart your tutorial session, as shown in the “Connecting to the application database” section on page 3–4.
Editing your application objects

You can customize the dynamic objects created by the Object Generator. You can change the look and feel of the objects and add additional functionality. This section describes how to open and edit dynamic objects in the AppBuilder.

To open and edit the dynamic objects in the Properties Window:

1. Choose File → Open Object from the AppBuilder main window.

2. Select the dynamic Viewer, armcuviewv, then click Open. The SmartDataViewer design window appears, looking something like the following:

The layout of the viewer might differ slightly depending on your user preferences. The individual field labels are specified through the schema of the DynSports database. The schema also specified that the Comments field be displayed as an editor instead of a fill-in.

3. Move the Comments field to the bottom of the viewer.

The Dynamic Properties window lets you edit Repository-based attributes and events for master objects or object instances registered in the Repository, as shown:

![Dynamic Properties Window](image)

The fields at the top of the property sheet describe the object: the container in which the object exists, the object’s full name, the object’s class, and a list of any possible result codes defined in the system. Result codes are used to customize your application for different UI types, different user types, and similar categories. This tutorial does not cover result codes. The Attributes tab displays all the attributes for the selected object, although only some are updateable. The Events tab displays all allowable events for the current object. See OpenEdge Development: Progress Dynamics Basic Development for more information on using result codes, specific attributes, and events.

Note: User-defined attributes stored in the Repository are not supported in static object property sheets. You can modify user-defined attributes only through this property sheet and the Repository Object Maintenance window.

5. Choose the Attributes tab, if necessary.

6. Change the value of the Scrollbar-Horizontal row to Yes. A mark in the first column indicates that the attributes value has been changed from its default value.
7. Change the value of the **Scrollbar-Vertical** row to **Yes**, if necessary, as shown:

![Screenshot of property sheet](image1)

8. Close the properties sheet. Your design window should now look something like this:

![Screenshot of design window](image2)

**Note:** The scroll bars might not appear immediately after you apply the new attribute values. This is caused by stale information in the cache. You can refresh the cache by saving your changes, closing the design window, and then reopening it.

Another automatic aspect of the framework is refiguring the minimum size needed for a dynamic viewer each time you save one. The framework trims excess space based on the position of the widgets in the viewer.

9. Delete the **Login company** field, by selecting it and pressing **DELETE**.
Building the Sample Application

10. Resize the design window and arrange the fields as you like. If you wish, add rectangles from the Object Palette to organize the fields into groups. Leave space to add a dynamic Combo for the Sales Rep field. When you have finished arranging the layout of the Viewer, it should look something like this one:

11. Click Save on the AppBuilder toolbar.

Adding a dynamic Combo to your viewer

The most common addition to a viewer is a selection list for a foreign key field. A foreign key field is a field in one table that matches a unique key field in another table. An example in the DynSports database is the SalesRep field in the Customer table, which is a foreign key for the SalesRep key field in the SalesRep table.

Progress Dynamics provides two different visualizations for choice lists: a dynamic Combo and a dynamic Lookup. Both are based on the SmartDataField (SDF), which provides a specialized representation of a single field in a viewer. The SDF retrieves the possible values for a field and visualizes them as either a combo box (dynCombo) or a button (dynLookup), which launches a maintenance window.

Part of the best practices recommendation for database design in Progress Dynamics is to include a unique key field on each table that is used only as a relationships target. These are the object (_obj) fields. Generally, you do not want the object field that uniquely identifies a table to appear in visual objects for that table. However, they are the recommended targets for SDFs.

To add a dynamic Combo for Sales Reps to the viewer:

1. Right-click the SmartDataField icon on the Object Palette. A pop-up menu of specific types of SmartDataFields appears.

2. Select DynamicCombo from the list. The cursor changes to represent a field object.
3. Select the **Sales Rep** field in the viewer to replace it with the dynamic Combo object. After the AppBuilder automatically sets up the proper connections, the **Choose Existing SmartDataField** dialog box appears:

![Choose Existing SmartDataField](image)

4. Click **Create New SDF**. The **SmartDataField Maintenance** window appears:

![SmartDataField Maintenance](image)

5. Type **SalesrepCombo** in the **SmartDataField** field and **Sales rep dynamic combo** in the **Description** field.

6. Select **ds-Entity** for the **Product module**.
7. Select **Database query** in the **Data source** radio set and select the **Use cache** toggle box.

These options control if and how the data in dynamic combos and lookups is cached. Caching this data can improve the performance of your application. However, you should not use caching without due consideration. For more information on this feature, see *OpenEdge Development: Progress Dynamics Basic Development*.

8. Double-click in the **Specify base query string** editor. The **Query Builder** appears:

![Query Builder](image)

This SDF uses a simple query that you could have typed directly into the editor. However, the **Query Builder** is useful for creating more complex queries. The query should always include the **NO-LOCK** keyword. Because a combo box cannot update the table, it only needs to read values and should not lock the table. Locking tables unnecessarily can have a major impact on performance.

9. Select **dynsports** in the **Database** combo box, if necessary.

10. Select **arm_salesrep** in the **Available Tables** list.

11. Click **Add>>**.

12. Click **OK**. The **SmartDataField Maintenance** window refreshes with the details of the table.

13. Type **1** in the **Display seq** field for **arm_salesrep.salesrep_code** in the browse. By setting the values in this browse, you can choose which fields to display in the SDF and in what order to display them. You should choose fields that let a user easily select the correct value. For this combo, the Sales Rep code and name are good choices.

14. Type **2** in the **Display seq** field for **arm_salesrep.salesrep_name** in the browse.
15. Select `arm_salesrep.salesrep_obj` for the **Key field**.

16. Type **Sales rep** for the **Field label** and **Select Sales rep from list** for the **Tooltip** on the **Details** tab. You can leave other fields in the property sheet at their default values, as shown:

17. Click **Save** to save this definition as a named, reusable dynamic Combo, and exit the **SmartDataField Maintenance** window.

18. Make any final adjustments in the design window to the viewer layout, then click **Save** in the AppBuilder toolbar. Your finished viewer should look something like the following:

19. Close the viewer design window.
Adding a dynamic Lookup to a viewer

A dynamic Lookup can represent a larger set of records than you can display in a dynamic Combo. In this exercise, you add a dynamic Lookup object to the dynamic viewer for the Order table. The dynamic Lookup launches a separate dynamic browse of Customer records. Using the dynamic Lookup, you can assign a customer to the selected Order.

To add the dynamic Combo to the Order viewer:

1. Choose File→Open Object in the AppBuilder main window.
2. Select the dynamic Viewer, artorviewv, then click Open.
3. Rearrange the fields in the design window. Lay out the fields any way you like, but leave some room to the right of the object fields, as shown:

4. Right-click on the SmartDataField icon on the Object Palette and select DynamicCombo.
5. Select the Sales Rep field to replace it with the dynamic Combo object. The Choose Existing SmartDataField dialog box appears.
6. Double-click SalesrepCombo on the Details tab. The SmartDataField Maintenance dialog box displays the properties for this new instance of the Sales Rep combo.
7. Click Save and exit the dialog box.

You have just reused the dynamic Combo that you created in the “Adding a dynamic Combo to your viewer” section on page 4–6.
To add a dynamic Lookup for customers to the viewer:

1. Right-click on the SmartDataField icon on the Object Palette and select DynamicLookup.

2. Select the Customer field in the viewer to replace it with the dynamicLookup object. After the AppBuilder automatically sets up the proper connections, the Choose Existing SmartDataField dialog box appears:

3. Click Create New SDF. The SmartDataField Maintenance dialog box appears:
4. Type **CustomerCodeLookup** in the **SmartDataField** field and **Customer code dynamic lookup** in the **Description** field.

5. Select **ds-Entity** for the **Product module**.

6. Select the **Use cache** toggle box.

7. Type the following query in the **Specify base query string** editor:

   ```
   FOR EACH arm_customer NO-LOCK INDEXED-REPOSITION
   ```

   Since this Lookup will not update the table, the query should include the **NO-LOCK** keyword. The use of the **INDEXED-REPOSITION** keyword is recommended for data access in a stateless environment.

8. Click **Refresh**. The remaining area of the property sheet is filled in with default choices for the arm_customer table, as shown:

9. On the **Details** tab, select **arm_customer.customer_obj** in the **Key field** combo.
10. Select `arm_customer.customer_name` in the **Displayed field** combo. This is the field whose value shows in the dynamic Lookup.

11. Type **Customer** for the **Field label**.

12. Select the following fields in the browse and set their **Display Sequences** as listed. These fields will display in the Customer browse launched by the dynamic Lookup:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Display sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>arm_customer.customer_code</td>
<td>1</td>
</tr>
<tr>
<td>arm_customer.customer_name</td>
<td>2</td>
</tr>
</tbody>
</table>

You can sort on a browse column by clicking on the **column header**. This screen shot shows the browser sorted by the **Browse Sequence** column after assigning the sequence values:
13. Select `arm_customer.customer_balance` in the browse.

In addition to showing fields in the Lookup browse window, you can display one or more fields of the related table on the viewer. You will add a field that displays the customer’s balance.

14. Type `YES` in the `Link field` column.

Normally, you also map each `Link Field` to a local variable in the viewer where the field’s value displays.

15. Type `fiCustomerBalance`, the field you will define to display the customer’s balance, in the `Linked widget` column, as shown:

![Image of the viewer setup with `fiCustomerBalance` field]

**Note:** The `fi` prefix is a naming convention for a fill-in.

16. Choose the `Other` tab.
17. Type `armcufullo` for the Maintenance SDO and `oeCustFoldWin` for the Maintenance object. You will build a browse window called `oeCustFoldWin` in the “Creating folder windows” section on page 4–33. Since it does not exist in the Repository yet, you cannot use the lookup and must be careful to spell the name correctly, as shown:

![Image of SmartDataField Maintenance (Instance Properties) window]

**Note:** There is a useful shortcut for filling in many Lookup fields. If you type a few letters in the field and press the TAB key, the framework automatically searches for matching records. If only one match exists, the framework populates the field with that value. If there are more matches, the framework launches a dialog box filtered to display the matches.

18. Click Save and exit the maintenance dialog box.

19. Click Fill In from the Object Palette. This is the local field for the Linked Widget your defined for your Lookup.
20. Drop the fill-in into the design window next to the **binoculars** icon representing the dynamic Lookup, as shown:

![Design Window](image1.png)

21. Type **fiCustomerBalance** for the **Object** and **Balance** for the **Label** in the AppBuilder main window, as shown:

![AppBuilder Window](image2.png)

The dynamic Lookup makes the correct association with the **Linked Widget** name you specified when you defined your lookup and displays the *customer_balance* field in this position.

22. Double-click the field to bring up its property sheet. Users should not edit the contents of this field, so you must disable it.
23. Deselect the **Enable** toggle box, as shown:

24. Click **OK**. Your **Order** viewer should now look something like the following:

25. Click **Save**.
Adding more dynamic Combos

There are two more object fields on the Order viewer. The validation triggers for the DynSports database require these values when performing certain operations from the Order viewer. You can now practice what you did in the “Adding a dynamic Combo to your viewer” section on page 4–6 by creating dynamic Combos for the **Status** and **Warehouse** fields. You might want to re-read that section to refresh your memory before beginning.

To create the Status Combo:

1. Select **DynamicCombo** from the **Object Palette**.
2. Select the **Status** field in the viewer.
3. Click **Create New SDF**.
4. Set the following values on the **SmartDataField Maintenance** dialog box:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SmartDataField</td>
<td>StatusCombo</td>
</tr>
<tr>
<td>Description</td>
<td>Status dynamic combo</td>
</tr>
<tr>
<td>Product module</td>
<td>ds-Entity</td>
</tr>
<tr>
<td>Specify base query string</td>
<td>FOR EACH gem_status NO-LOCK INDEXED-REPOSITION</td>
</tr>
<tr>
<td>Display seq. of gem_status.status_code</td>
<td>1</td>
</tr>
<tr>
<td>Display seq. of gem_status.status_name</td>
<td>2</td>
</tr>
<tr>
<td>Key field</td>
<td>gem_status.status_obj</td>
</tr>
<tr>
<td>Field label</td>
<td>Status</td>
</tr>
<tr>
<td>Tooltip</td>
<td>Select status from list</td>
</tr>
</tbody>
</table>

5. Click **Save** and exit the **SmartDataField Maintenance** dialog box.

To create the Warehouse Combo:

1. Select **DynamicCombo** from the **Object Palette**.
2. Select the **Warehouse** field in the viewer.
3. Click **Create New SDF**.
4. Set the following values on the **SmartDataField Maintenance** dialog box:
5. Click Save and exit the **SmartDataField Maintenance** dialog box.

6. Make any final adjustments in the design window to the viewer layout, then click Save in the AppBuilder toolbar. Your finished viewer should look something like the following:

7. Close the viewer design window.
Modifying the Orderline viewer

The Orderline viewer also has several object fields on it. You must replace them with dynamic Combos or Lookups.

To modify the Orderline viewer:

1. Choose File → Open Object in the AppBuilder main window.
2. Select the dynamic Viewer, artolviewv, then click Open.
3. Rearrange the fields in the design window. Lay out the fields any way you like, but leave some room to the right of the object fields, as shown:

   ![Modified Orderline Viewer]

To create the Order Lookup:

1. Select DynamicLookup from the Object Palette.
2. Select the Order field in the viewer.
3. Click Create New SDF.
4. Set the following values on the SmartDataField Maintenance dialog box:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SmartDataField</td>
<td>OrderLookup</td>
</tr>
<tr>
<td>Description</td>
<td>Order dynamic lookup</td>
</tr>
<tr>
<td>Product module</td>
<td>ds-Entity</td>
</tr>
<tr>
<td>Specify base query string</td>
<td>FOR EACH art_order NO-LOCK INDEXED-REPOSITION</td>
</tr>
<tr>
<td>Display seq. of art_order.order_code</td>
<td>1</td>
</tr>
<tr>
<td>Key field</td>
<td>art_order.order_obj</td>
</tr>
<tr>
<td>Displayed field</td>
<td>art_order.order_code</td>
</tr>
<tr>
<td>Field label</td>
<td>Order</td>
</tr>
<tr>
<td>Maintenance SDO</td>
<td>artorfullo</td>
</tr>
<tr>
<td>Maintenance object</td>
<td>oeOrderFoldWin</td>
</tr>
</tbody>
</table>

1 You build this window in the "Creating an order maintenance window" section on page 4–48.
5. Click **Save** and exit the **SmartDataField Maintenance** dialog box.

### To add the Status Combo:

1. Select **DynamicCombo** from the **Object Palette**.
2. Select the **Status** field in the viewer.
3. Double-click **StatusCombo** on the **Details** tab. The **SmartDataField Maintenance** dialog box displays the properties for this new instance of the **Status** combo.
4. Click **Save** and exit the dialog box.

### To create the Item Combo:

1. Select **DynamicCombo** from the **Object Palette**.
2. Select the **Item** field in the viewer.
3. Click **Create New SDF**.
4. Set the following values on the **SmartDataField Maintenance** dialog box:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SmartDataField</td>
<td>ItemCombo</td>
</tr>
<tr>
<td>Description</td>
<td>Item dynamic combo</td>
</tr>
<tr>
<td>Product module</td>
<td>ds-Entity</td>
</tr>
<tr>
<td>Specify base query string</td>
<td>FOR EACH ivm_item NO-LOCK INDEXED-REPOSITION</td>
</tr>
<tr>
<td>Display seq. of ivm_item.item_code</td>
<td>1</td>
</tr>
<tr>
<td>Display seq. of ivm_item.item_name</td>
<td>2</td>
</tr>
<tr>
<td>Key field</td>
<td>ivm_item.item_obj</td>
</tr>
<tr>
<td>Field label</td>
<td>Item</td>
</tr>
<tr>
<td>Tooltip</td>
<td>Select item from list</td>
</tr>
</tbody>
</table>
5. Click Save and exit the SmartDataField Maintenance dialog box. Your Orderline viewer should now look something like the following:

![SmartDataField Viewer](image)

6. Save your changes and close the design window.

Creating browse windows

You used the Entity Import tool to add to the Repository the first part of the data that defines your application, entity data. You used the Object Generator to generate the second part of data, objects like SDOs, browses, and viewers. Then, you customized that data using the AppBuilder.

The next stage is defining the windows where you can display those objects, along with tab folders, toolbars, and other objects, to complete your application’s user interface. You will use the Container Builder to create your dynamic windows.

The Container Builder is the preferred tool for building complex containers in Progress Dynamics. The Container Builder manages the layout of objects and the links between them. When you save the container, these relationships are stored as records in the Repository.

You can also mark a container as a template. Then, you can use it as the basis to quickly build several similar containers by customizing instances of the template. See OpenEdge Development: Progress Dynamics Basic Development for in-depth information on the Container Builder.
Defining the basic window

To create a browse window:

1. Choose Build → Container Builder from the AppBuilder main window. The Container Builder appears:

![Container Builder toolbar](image)

**Note:** This is where you build the dynamic windows that make up the application.

The framework makes extensive use of data maintenance functions where you first select a record in a browse window and then maintain it in a separate window. The browse window is an independent window, a window that is invoked directly from a button or menu item without requiring any values to be passed in to it. The framework usually calls such windows “Object Controllers,” therefore their Object Type is “DynObjc.” You can use this object type to build any window that can be run directly.

1. Click New on the Container Builder toolbar to create a new container.
2. Click **Lookup** in the **Create from existing container** group. The **Container Lookup** dialog box appears:

![Container Lookup dialog box]

The **Container Lookup** dialog box lists predefined templates for building different styles of windows. You can add more templates to this list that you create for your application. The templates are existing containers that can have both visual and nonvisual objects, links between the objects, and multiple tab pages. When you create a container from a template, you replace the existing template objects with your application objects.

3. Double-click the **rywinObjCont** template to select it. You return to the **Container Builder**.

**Note:** You can find the template quickly using the **Filter** tab. Type **rywin** in the **From Value** field for the **Object filename**, then choose **Apply**.

4. Click **Create**. This creates an instance of the template object for you to modify. Note that the **Type** is set as **DynObjc / Independent Window**.

5. Type **oeCustBrowseWin** in the **Container** field.

6. Type **Customer selection browse** as the **Description** for the new window.

7. Select **ds-OE** for the **Product Module**.

8. Type **Customer Browse** for the **Window Name**.
Replacing template objects on the folder page

The right-hand side of the Container tab folder shows a grid. The visual grid acts as placeholder for the relative layout of objects like toolbars, browses, and viewers. The nonvisual grid acts as a placeholder for data sources like SDOs and SBOs. As shown in the grid, the window has three visual objects, two toolbars and a template object for a dynamic browse. Template objects are nonfunctional place holders that must be replaced by actual objects. It also has one nonvisual object, an SDO template object. You replace these templates with actual objects to retrieve and display data from the database.

To replace the template objects with your application objects:

1. Click Show all visual objects in the Container folder.

2. Choose rytemfullb, the browse template object, in the grid. The Instance information displays the data for the object.

3. Click Lookup beside the Object field. The Object Lookup dialog box appears, filtered to show only browses:

4. Double-click armcufullb, the customer browse, to select it. The Container Builder automatically substitutes the customer browse for the browse template object. (This is a temporary change until you save your work to the Repository.) You automatically return to the Container Builder window.

5. Type CustomerBrowse as the Name for this instance of the armcufullb browse.
6. Click **Dynamic Properties** on the folder’s object instance toolbar. The *object instance toolbar* is the one inside the folder. The toolbar at the top of the window is the *container toolbar*. The **Dynamic Properties** sheet for the browse appears:

![Dynamic Properties Sheet](image)

7. Type *oeCustFoldWin* for the **FolderWindowToLaunch** attribute. This is the Customer Maintenance window you will build in the “Creating folder windows” section on page 4–33. It will launch whenever a user chooses a record to edit from the Customer browse window.

8. Close the property sheet. The property value is saved temporarily. The change is not saved permanently until you save the container.
9. Click **Show all non-visual objects** in the **Container** folder. The grid now shows the template’s SDO:

![Image of Container Builder showing Show all non-visual objects]

10. Click **Lookup** next to the **Object** field. The **Object Lookup** dialog box appears, filtered to show only SDOs:

![Image of Object Lookup dialog box]

*Note: The screenshots and images are placeholders for the actual content.*
11. Double-click **armcufullo**, the customer SDO, to select it. You automatically return to the **Container Builder**, and the following message dialog box appears:

![Message dialog box](image)

The SDO template object is a static object. The customer SDO is a dynamic object. The framework recognizes that these are different object types. This message reminds you that different objects might not have all the same attributes and behavior changes might result. In this case, it does not matter because the template was designed so either a dynamic or static SDO could replace it.

12. Click **OK** to continue.

13. Type **CustomerSDO** as the **Name** for this instance of the armcufullo SDO.

14. Click **Links Maintenance** in the container toolbar. The **Links Maintenance** dialog box appears:
SmartObjects communicate through named connections called OpenEdge SmartLinks™. A SmartLink is a two-way association of two SmartObjects. The link type establishes how the SmartObjects relate to each other and what actions to expect from the other. Simply by naming the source and target objects and the link type, you enable the AppBuilder to set up the channels the objects need to communicate. See *OpenEdge Development: Progress Dynamics Basic Development* for more information on SmartLinks.

The links you see are defined in the template container. When you replaced the template objects with your application objects, the **Container Builder** automatically updated the links to use your objects.

15. Verify that the following links are defined for the objects on the **oeCustBrowseWin** window. Select the **Source** column header to sort the links:

<table>
<thead>
<tr>
<th>Source</th>
<th>Link name</th>
<th>Target</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BrowseToolbar</td>
<td>Toolbar</td>
<td>CustomerBrowse</td>
<td>Enables toolbar to execute browse functions.</td>
</tr>
<tr>
<td>CustomerSDO</td>
<td>Data</td>
<td>CustomerBrowse</td>
<td>Passes data values to browse.</td>
</tr>
<tr>
<td>ObjcTop</td>
<td>ContainerToolbar</td>
<td>THIS-OBJECT</td>
<td>Enables toolbar to execute container functions.</td>
</tr>
<tr>
<td>ObjcTop</td>
<td>Navigation</td>
<td>CustomerSDO</td>
<td>Notifies SDO to navigate through data records.</td>
</tr>
<tr>
<td>THIS-OBJECT</td>
<td>PrimarySDO</td>
<td>CustomerSDO</td>
<td>Indicates to container that CustomerSDO is primary SDO.</td>
</tr>
</tbody>
</table>

**Note:** THIS-OBJECT refers to the container, oeCustBrowseWin.

16. Exit the **Links Maintenance** dialog box.

17. Click **Save**, and exit the **Container Builder**.
Creating an Order Selection Window

To complete the tutorial, you also need an Order Selection Window, similar to the Customer Browse Window you just created. See if you can repeat the procedure from the “Creating browse windows” section on page 4–22 to build your Order Selection Window. Table 4–1 shows the substitutions to make while building your Order Selection Window. All the other properties values are the same for both windows.

Table 4–1: Order Selection Window properties

<table>
<thead>
<tr>
<th>For . . .</th>
<th>Substitute . . .</th>
<th>With . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container name</td>
<td>oeCustBrowseWin</td>
<td>oeOrderBrowseWin</td>
</tr>
<tr>
<td>Container description</td>
<td>Customer selection browse</td>
<td>Order selection browse</td>
</tr>
<tr>
<td>Window name</td>
<td>Customer Browse</td>
<td>Order Selection</td>
</tr>
<tr>
<td>Template browse</td>
<td>armcufullb</td>
<td>artorfullb</td>
</tr>
<tr>
<td>Browse instance name</td>
<td>CustomerBrowse</td>
<td>OrderBrowse</td>
</tr>
<tr>
<td>FolderWindowToLaunch attribute</td>
<td>oeCustFoldWin</td>
<td>oeOrderFoldWin</td>
</tr>
<tr>
<td>Template SDO</td>
<td>armcufullo</td>
<td>artorfullo</td>
</tr>
<tr>
<td>SDO instance name</td>
<td>CustomerSDO</td>
<td>OrderSDO</td>
</tr>
</tbody>
</table>

When you complete building your Order Selection Window, you are ready to see the results of your work. You can now run the windows and see what they look like.

Running your dynamic browse window

Now you can run your window. Because the window is a dynamic object, defined only by records in the Progress Dynamics Repository, there is nothing to compile. However, you cannot simply run a dynamic window as a procedure. There is a tool that lets you run it as a logical container. This is referred to as launching the container.
To launch your window:

1. Choose Compile→Dynamic Launcher from the AppBuilder main window.

2. Type oeCustBrowseWin in the Name of Container to Launch field, as shown:

![Dynamic Launcher](image)

The Dynamic Launcher has the following options:

- The Run Persistent toggle box tells the launcher to run the dynamic window as a persistent procedure. This is usually the case for Progress Dynamics objects.

- The Always run new instance toggle box tells the launcher to create a new instance of the container each time you choose Run. If you want to only have a single instance of any container running at once, deselect this toggle box.

- The Clear repository cache toggle box refreshes all object definition data cached on the client, enabling you to see the effects of your latest work. Leave this toggle box selected when launching a dynamic object that you just created or modified.

- The Clear data cache toggle box refreshes the application data cache. This cache contains data from your application database, rather than from the Repository.

- The Destroy ADM Super Procedures toggle box clears another level of the framework’s persistent data. Choosing this option shuts down all currently running super procedures, forcing them to be restarted using the latest versions of the files.

- The Clear the MRU history button clears the Dynamic Launcher’s list of most recently used files.

**Note:** Occasionally, you can clear the cache, restart the super procedures, and still not get the results you expect. If that happens, restart your session. That clears any remaining cached information.

3. Click Run. The Container Browser window appears:

![Customer Browse](image)
4. Try out a few of the features that all browser windows built with this template inherit automatically. For instance, when you resize it, the toolbars and the browser resize appropriately.

5. Click **Filter Records** and a dynamic filter dialog box for this table’s query appears. Index information is generated automatically to let the user know which fields to use for efficient filtering. Enter a value in the **From** and **To** columns to filter all records where the field matches that range of values.

Entering a value in just the **From** column filters all values greater than or equal to that value. Scroll the browser to the right, or resize it, and you can see that fields with a word index can be filtered by contained words, and all character fields can be filtered on a character string match, as shown:

![Filter Records Dialog Box]

**Note:** The Customer Code field is actually a CHARACTER field. This affects the way the filter operates.

If you select the **Permanent** radio button, the filters you have set up are saved as a permanent user preference in the Repository. Otherwise, the filters last for the current session. This is an example of customizations and preferences saved for each different user. (Because the framework tools themselves are largely implemented in the framework, this is equally true for developers and for end users.)
The framework knows the effective and ineffective ways to filter the data based on the available indexes. If you choose an ineffective filter criteria, you might see the following message:

![Warning message](image)

6. Click **OK** on the **Filter** dialog box to return to the **Customer Browse** window. Note that the **Filter Records** button now has a check mark on it. This shows that your data has been filtered.

7. You can also click **Find** to reposition to the first record matching the criteria you enter.

The other standard buttons transfer values from the browse to an Excel™ spreadsheet or XML document, generate Audit trail records if auditing is enabled for the table, and associate Comments or a Status record with a selected record.

8. Exit the **Customer Browse** window and the **Dynamic Launcher**.

---

### Creating folder windows

You now have browse windows for the Customer and Order tables. In this section, you create a maintenance window for the Customer table. This tutorial shows the style of building application windows that is used extensively in the framework. You can organize your application screens any way you like. Creating multi-page tab folders gives you a sense of what you can do.

> **To create a Customer maintenance browse:**

1. Choose **Build → Container Builder** from the AppBuilder main window. The **Container Builder** appears.

2. Click **New** on the **Container Builder** toolbar.

3. Type **rywinFolder** in the **Container** field in the **Create from existing container** section, and press the **TAB** key.

   The rywinFolder object is a template for a dependent window. This rywinFolder template has a Page 0 with a Navigation and Update toolbar, the tab folder itself, and a folder Page 1 where a Viewer goes to display and update records.

   You will pass in the Customer key from the Customer Browse Window so your Customer Maintenance browse displays the same record.

4. Click **Create** to create the folder window.
Note that this container’s type is **DynFold / Dependent Window**. A dependent window links to another window that supplies key values for record retrieval. Typically in the framework, a dependent window is a tab folder that might display multiple pages and update related data for the selected record. The object type of DynFold is named to reflect this fact. However, you can build any type of window, with or without a tab folder, as a dependent window.

5. Select **ds-OE** in the **Product module** combo box.

6. Type **oeCustFoldWin** in the **Container** field.

7. Type **Customer maintenance folder window** in the **Description** field.

8. Type **Customer Orders Browse** in the **Window name** field, as shown:
9. Click **Page Maintenance** in the **Container Builder** toolbar. The **Page Maintenance** window appears:

![Page Maintenance Window](image)

**Note:** This tool allows you to manage data about pages in your folder windows.

10. Select the **Details** record, and click **Modify**.

11. Type **Customer** in the **Page label** and **Security token** fields, as shown:

![Page Label and Security Token](image)
12. Click **Save**, and exit the **Page Maintenance** window.

13. Choose the **Customer (Pg 1)** tab, as shown:

![Image](image.png)

14. Select the template viewer icon in the grid.

15. Type `rmcuviewv`, your Customer Viewer, in the **Object** field and press the **TAB** key. The **Container Builder** substitutes the dynamic Viewer for the template object.

16. Click **OK** to clear the **Replacing objects of different type** message box.

17. Type **CustomerViewer** for the **Name**.

18. Click **Save** in the container toolbar.

Now you have filled in the predefined elements of the Folder template. You can also expand upon what the template gave you by adding more folder pages and objects.

**Adding another page to the folder window**

To make your Customer Maintenance browse more useful, you can add another page to the tab folder to show Orders and OrderLines for the current Customer. You can quickly build the additional page from another template.
To add a new page to your folder window:


2. Click New to add a second page. Note that the Page Sequence is automatically set to 2.

3. Type Orders/Lines as the Page label.

4. Click Save. Note that the Security token is automatically set to the Page label, as shown:

5. Type rywinparentchild in the Container field and press the TAB key.

6. Click Insert/Replace, and exit the Page Maintenance window.
To modify the new folder page:

1. Choose the Orders/Lines (Pg 2) tab in the Container Builder.

   The Parent-Child layout that you just added contains template objects for two SDOs (nonvisual) and two Browses (visual), as shown:

   ![Container Builder screenshot](image)

   ![Table](image)

2. Select the icon in Cell A-2 on the grid. You can change the layout of the window by rearranging these icons.
3. Left-click the icon, and drag and drop it on **Cell B-1**, as shown:

![Container Builder window](image)

4. Click **Show all non-visual objects**, as shown:

![Container Builder window](image)

5. Select the first SDO (**Cell C-1** in the Container Builder’s grid).

6. Type **artorfullo**, your Order SDO, in the **Object** field and press the **TAB** key. The **Container Builder** substitutes the dynamic SDO for the template object.

7. Click **OK** to clear the **Replacing objects of different type** message box.
8. Type OrderSDO for the Name.

9. Select the second SDO (Cell D-1 in the Container Builder’s grid).

10. Type artorfullo, your Order SDO, in the Object field and press the TAB key. The Container Builder substitutes the dynamic SDO for the template object.

11. Click OK to clear the Replacing objects of different type message box.

12. Type OrderlineSDO for the Name.

Setting the SDO foreign fields

Now that your new page has its SDOs, you must define the relationship between the Order SDO and the external Customer SDO from which the Order SDO gets its key value. Remember that you want to display Orders for the currently selected Customer. You must identify the key value that gets passed from the Customer SDO in the Customer Browse Window to this Order SDO. This list of key values is called the Foreign Fields list. As with most common AppBuilder tasks, you can use several methods to set up these relationships.

In the tutorial application, the Order SDO’s query is filtered by the customer_obj field, and the parent Customer SDO’s customer_obj field is passed in and inserted into the Order SDO’s query.

To set the foreign fields:

1. Select the artorfullo SDO in the folder’s grid.

2. Click Dynamic Properties from the instance toolbar inside the folder page.

3. Type art_order.customer_obj,customer_obj for the value of the ForeignFields attribute, as shown:
4. Select the artoldullo SDO in the folder’s grid.

5. Click Foreign-Field Mapping beside the Foreign fields editor. The Foreign-Field Mapping dialog box appears:

![Foreign-Field Mapping dialog box]

6. Select order_obj in the Source column and art_orderline.order_obj in the Target column.

7. Click Map, and exit the Foreign-Field Mapping dialog box.

Adding browsers on the new page

Now that you have set the proper relationships between the page’s SDOs, you must replace the template browsers with Order and Order Lines browsers.

To add the browsers:

1. Click Show all visual objects in the Container folder.


3. Type artorfullb, your Order browse, in the Object field and press the TAB key. The Container Builder substitutes the dynamic Browse for the template object.

4. Type OrderBrowse for the Name.

5. Select the DynBrow object in Cell B-1.

6. Type artolfullb, your Orderline browse, in the Object field and press the TAB key. The Container Builder substitutes the dynamic Browse for the template object.
7. Type OrderlineBrowse for the Name.

8. Click Save to save your container.

Setting links for the new page

The links that define how data and event messages are passed between objects are normally defined in the template. When you replace the template objects with real application objects, those links are established between the real objects in your application window.

But in this case, you added a template for another page to the basic one-page folder template. As you saw, you had to identify the key values to be passed in to the Order SDO on Page 2. You also must define a link between your container window and that SDO, so the framework can route the data.

To set the links:

1. Click Links Maintenance in the Container Builder toolbar. The Links Maintenance window displays all the existing links for your window, as shown:

   ![Links Maintenance Window]

   Note: These are the links for both template containers.
2. Click New.

3. Select **THIS-OBJECT** as the Source, **Data** as the Link, and **OrderSDO** as the Target. Then save the new link.

**Note:** If you do not see the appropriate choice in the lists, choose the Filter from source button under the list.

4. Click New.

5. Select **THIS-OBJECT** as the Source, **Data** as the Link, and **CustomerViewer** as the Target. Then save the new link.

6. Verify that all of the links in the following table are listed:

<table>
<thead>
<tr>
<th>Source</th>
<th>Link</th>
<th>Target</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>afspfoldrww</td>
<td>Page</td>
<td>THIS-OBJECT</td>
<td>Enables folder to communicate to container for page handling.</td>
</tr>
<tr>
<td>CustomerViewer</td>
<td>Update</td>
<td>THIS-OBJECT</td>
<td>Saves customer changes back to customer SDO.</td>
</tr>
<tr>
<td>OrderlineSDO</td>
<td>Data</td>
<td>OrderlineBrowse</td>
<td>Passes orderline data values to orderline browse.</td>
</tr>
<tr>
<td>OrderSDO</td>
<td>Data</td>
<td>OrderBrowse</td>
<td>Passes order data values to order browse.</td>
</tr>
<tr>
<td>OrderSDO</td>
<td>Data</td>
<td>OrderlineSDO</td>
<td>Passes order data (foreign field key) to order line SDO.</td>
</tr>
<tr>
<td>StandardToolbar</td>
<td>ContainerToolbar</td>
<td>THIS-OBJECT</td>
<td>Enables toolbar to execute container functions.</td>
</tr>
<tr>
<td>StandardToolbar</td>
<td>Navigation</td>
<td>THIS-OBJECT</td>
<td>Notifies SDO to navigate through data records.</td>
</tr>
<tr>
<td>StandardToolbar</td>
<td>TableIO</td>
<td>CustomerViewer</td>
<td>Enables toolbar to request customer viewer to allow changes.</td>
</tr>
<tr>
<td>THIS-OBJECT</td>
<td>Data</td>
<td>OrderSDO</td>
<td>Passes customer data (foreign field key) to order SDO.</td>
</tr>
<tr>
<td>THIS-OBJECT</td>
<td>Data</td>
<td>CustomerViewer</td>
<td>Passes customer data to customer viewer.</td>
</tr>
</tbody>
</table>
THIS-OBJECT represents the container window for the tab folder. The Customer key values pass through the container from the Customer Browse Window that calls it. When constructing the window, you treat the container as if it were the source of the data values.

7. Exit the **Links Maintenance** window.
8. Click **Save** in the **Container Builder** toolbar, then exit the **Container Builder**.

**Running your folder window**

To see the results of your work, you must first launch the Customer Browse. Your new Customer Orders Browse is a dependent window. It expects a foreign key value to be passed in from its parent window.

**Note:** Remember that you must have the Repository and the DynSports databases connected to run your application. The Repository is automatically connected by the framework managers when you start Progress Dynamics. However, you have not modified the configuration file yet to instruct the managers to connect the DynSports database at startup. If you want to check the connections, choose **Tools**→**Database Connections** from the AppBuilder main window.

To view the Customer Orders Browse:

1. Choose **Compile**→**Dynamic Launcher**. The **Dynamic Launcher** appears.
2. Type `oeCustBrowseWin` in the **Name of Container to Launch** field, and click **Run**. Your **Customer Browse** appears:
3. Select a customer and click **Modify** in the bottom toolbar. The **Customer Orders Browse** appears:

Look over the **Customer** tab. Note the **Sales Rep** combo box that you defined for the viewer earlier. If you choose the button next to one of the decimal fields, **Credit Limit** or **Balance**, a calculator appears:

This is a standard feature of dynamic viewers. You get the calculator by default for decimal fields unless you request otherwise. Similarly, a calendar comes up for date fields. These are ABL-based objects, not ActiveX controls.
4. Select the **Orders/Lines** tab, as shown:

Here you see the dynamic Browsers you defined for this page. You can see Orders for the current Customer and Order Lines for the current Order. Select another Order to see how the **Order Lines Browser** is repopulated.

5. Exit the **Customer Orders Browse** and the **Customer Browse** windows, and close the **Dynamic Launcher**.

### Changing the layout of your dynamic window

Most Progress Dynamics windows and pages use *relative* layouts to determine object positions and sizes. When you define a template for a container or page layout, you specify how the objects are laid out relative to one another. You can also specify if objects should be resized horizontally or vertically when the container is resized. By default, browses resize in both dimensions, to show more fields and more rows; viewers do not resize; and toolbars resize horizontally. An object’s dynamic properties allow you to individually configure resizing.

The Parent-Child window apportioned the available space for the two browses, after laying out the toolbar and the Tab Folder. If you resize the window, the browses resize. You can easily change these characteristics to get a different arrangement.

To change the page layout:

1. Choose **Build→Container Builder** from the AppBuilder main window. The **Container Builder** appears.

2. Type **oeCustFoldWin** in the **Container** field and press the **TAB** key.

3. Choose the **Orders/Lines** tab.

4. Choose the **Show visual object** button to see your two dynamic Browses.
5. Select the artolfullb browse in Cell B-1.

Remember that this grid does not reflect absolute column and row position, but the relative ordering of objects from top to bottom and from left to right.

6. Drag the artolfullb browse to Cell A-2. Note that as you drag the object, the grid highlights the possible places to which you can move it, as shown:

7. Click Save, and exit the Container Builder.

You must shut down running objects before making changes. An object's definition is only records in the Repository database. The Dynamic Launcher by default clears the client-side cache of those records each time it is run. If you want to see the effects of your changes, you have to get back out to the Dynamic Launcher before rerunning a window.

Remember also that you must run the Customer Browse, oeCustBrowseWin, from the Dynamic Launcher rather than trying to run the Customer Orders Browse, oeCustFoldWin, directly. The folder window needs a customer_obj key as input when it is launched, and it generates errors if it is run directly.


9. Type oeCustBrowseWin in the Name of Container to Launch field, and choose Run.
10. Double-click on a customer record to launch the **Customer Orders Browse**. Your **Orders/Lines** page with the new layout should look like the following screen:

![Customer Orders Browse](image)

Note that the **Order Browse** is now on top of the **OrderLine Browser**. The dynamic window layout program takes care of figuring out the sizes and positions for you, based on your new relative layout.

11. Close your application windows and the **Dynamic Launcher**.

### Creating an order maintenance window

You have successfully built a customer maintenance window. In this section, you build a maintenance window for the Order and Order Lines tables. You do not use a template to build the basic container for this window.

To build an Order maintenance window:

1. Choose **Build** → **Container Builder** from the AppBuilder main window. The **Container Builder** appears.

2. Click **New** in the **Container Builder**.
3. Set the values shown in the following table for your new window:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container</td>
<td>oeOrderFoldWin</td>
</tr>
<tr>
<td>Type</td>
<td>DynFold / Dependent Window</td>
</tr>
<tr>
<td>Description</td>
<td>Order maintenance folder window</td>
</tr>
<tr>
<td>Product module</td>
<td>ds-OE / Order Entry module</td>
</tr>
<tr>
<td>Window name</td>
<td>Order and Lines Browse</td>
</tr>
</tbody>
</table>

4. Add the objects shown in the following table to Page 0 of your folder, using the New button on the folder’s object instances toolbar:

<table>
<thead>
<tr>
<th>Object filename</th>
<th>Grid location</th>
</tr>
</thead>
<tbody>
<tr>
<td>StandardToolbar</td>
<td>A1</td>
</tr>
<tr>
<td>afspfoldrw.w</td>
<td>A9</td>
</tr>
</tbody>
</table>

The afspfoldrw.w file is an OpenEdge SmartFolder™ object. It handles multi-page windows and supports interactions with them. The Container Builder should look something like this:
To add an Order page to the tab folder:

1. Click Page Maintenance. The Page Maintenance dialog box appears.
2. Click New in the Page Maintenance dialog box.
3. Type Order for the Page label.
4. Click Save, then exit the Page Maintenance dialog box.
5. Choose the new Order tab.
6. Click New on the folder’s object instances toolbar.
7. Type artorviewv in the Object field and press the TAB key.
8. Select Cell A-1 in the grid to place the Order viewer. The Container Builder should look something like this:

   ![Container Builder](image)

9. Type OrderViewer for the Name, and click Save.
To set up the appropriate links for the Order page:

1. Click **Links Maintenance**. The **Links Maintenance** dialog box appears. There should be only one link listed, the Page link for the folder.

2. Add the links shown in the following table:

<table>
<thead>
<tr>
<th>Source</th>
<th>Link</th>
<th>Target</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OrderViewer</td>
<td>Update</td>
<td>THIS-OBJECT</td>
<td>Saves order changes back to customer SDO.</td>
</tr>
<tr>
<td>StandardToolbar</td>
<td>Navigation</td>
<td>THIS-OBJECT</td>
<td>Notifies SDO to navigate through data records.</td>
</tr>
<tr>
<td>StandardToolbar</td>
<td>TableIO</td>
<td>OrderViewer</td>
<td>Enables toolbar to request order viewer to allow changes.</td>
</tr>
<tr>
<td>StandardToolbar</td>
<td>ContainerToolbar</td>
<td>THIS-OBJECT</td>
<td>Enables toolbar to execute container functions.</td>
</tr>
<tr>
<td>THIS-OBJECT</td>
<td>Data</td>
<td>OrderViewer</td>
<td>Passes order data to order viewer.</td>
</tr>
</tbody>
</table>

**Note:** If a choice is not listed in one of the columns, use the filter button below the column to show all the available choices.
3. Exit the **Links Maintenance** dialog box, as shown:

![Links Maintenance dialog box](image)

4. Click **Save** in the **Container Builder**.

You now have a basic folder window with an Order viewer on it.

**Adding an Order Lines page**

Next, you extend the basic window by adding a new page with a browse of the OrderLines SDO records for the current Order. The window should allow updates to the current OrderLine using a dynamic viewer created by the Object Generator.

To add the Order Lines page:

1. Click **Page Maintenance**. The **Page Maintenance** dialog box appears.
2. Click **New** to add a second page. Note that the **Page Sequence** is automatically set to 2.
3. Type **Order lines** for the **Page label**.
4. Click **Save**.
5. Type **rywinbrsdynvw** in the **Container** field and press the **TAB** key.
6. Click **Insert/Replace**, and then exit the **Page Maintenance** dialog box.
Creating an order maintenance window

Adding objects to the Order Lines page

Now, you must replace the template objects with your application objects.

To change the objects:

1. Choose the Order Lines (Pg 2) tab.
2. Replace the template objects with the objects for the OrderLine table, as shown in the following table:

<table>
<thead>
<tr>
<th>Replace . . .</th>
<th>With . . .</th>
<th>Name . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDO template</td>
<td>artofullo</td>
<td>OrderlineSDO</td>
</tr>
<tr>
<td>Dynamic browse template</td>
<td>artofullb</td>
<td>OrderlineBrowse</td>
</tr>
<tr>
<td>Dynamic viewer template</td>
<td>artolviewv</td>
<td>OrderlineViewer</td>
</tr>
</tbody>
</table>

Note: When you replace the template SDO, the message box about replacing an object with an object of a different type appears again. Choose OK and continue.

3. Open the Dynamic Properties sheet for the Orderline SDO.
4. Type `art_orderline.order_obj.order_obj` for the ForeignFields attribute.
5. Click Save.

Linking the Order Lines page

As on the Customer Orders Browse, you must add links to hook the OrderLine SDO with the Order SDO and the Order viewer. The template you used for this page already has the rest of the links to pass data from the OrderLine SDO to its browse and viewer.

To add the links:

1. Click Links Maintenance. The Links Maintenance dialog box appears.
2. Verify that all of the links in the following table exist, and add any links that are missing:

<table>
<thead>
<tr>
<th>Source</th>
<th>Type</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>afspoldrwv</td>
<td>Page</td>
<td>THIS-OBJECT</td>
</tr>
<tr>
<td>OrderlineSDO</td>
<td>Data</td>
<td>OrderlineBrowse</td>
</tr>
<tr>
<td>OrderlineSDO</td>
<td>Data</td>
<td>OrderlineViewer</td>
</tr>
<tr>
<td>OrderlineViewer</td>
<td>Update</td>
<td>OrderlineSDO</td>
</tr>
</tbody>
</table>
3. Exit the Links Maintenance window.

4. Click Save, then exit the Container Builder window.

To test your new application, launch oeOrderBrowseWin from the Dynamic Launcher. This is the second browse window that you created earlier. You can then launch your new window, oeOrderFoldWin, by editing one of the displayed records. Remember that oeOrderFoldWin is a dependent window and needs the order_obj key from oeOrderBrowseWin as input. Your new browse should look something like the following:
Because the template for the Order Lines page of the Order and Lines Browse is designed to have its own update and navigation toolbar, the UI of the window might be confusing: there is one toolbar at the top of the window to navigate and maintain the Order displayed on Page 1, and another toolbar on Page 2 to maintain and navigate OrderLines. A different set of templates could use a single toolbar on Page 0 to handle the currently selected page. You can leave the construction of this variant for an exercise, after you have read the chapter on using the Container Builder in OpenEdge Development: Progress Dynamics Basic Development.

Note: You could have created a window where Customers and Orders were created and updated together, rather than one window that updates Customers and just displays Orders and OrderLines, and a separate window to maintain Orders. The sample application puts things together in a way that shows as much of the framework as possible while building a few objects.

Creating a main menu window for your application

In this section, you pull the pieces together into something that looks like a real application. The starting template for the main menu window includes some standard menus. In the next section, you will create a custom menu that launches your application windows.

To create a main menu window:

1. Choose Build → Container Builder from the AppBuilder main window.
2. Click New in the Container Builder.
3. Type rywinMenuCont for the Container in the Create from existing container area and press the TAB key. This is the framework’s default template for a menu controller window. You can build and use a different template customized for your applications.

Note: A Dynamic Menu Controller is a window with just a MenuBar and Toolbar. It is intended to be the main entry point of an application.

4. Click Create.
5. Set the following values in the Container Builder:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container</td>
<td>oeMenuWin</td>
</tr>
<tr>
<td>Type</td>
<td>DynMenc / Dynamic Menu Controller</td>
</tr>
<tr>
<td>Description</td>
<td>Order entry menu window</td>
</tr>
<tr>
<td>Product module</td>
<td>ds-OE / Order Entry module</td>
</tr>
<tr>
<td>Window name</td>
<td>DynSports Order Entry</td>
</tr>
</tbody>
</table>
The **Container Builder** should look like the following:

![Container Builder](image)

6. Click **Save**, then exit the **Container Builder**.

You cannot run this window yet. You must define the menu items that launch your maintenance windows first.

### Using the Toolbar and Menu Designer

In this section, you use the Toolbar and Menu Designer. You can use this tool to construct a toolbar. You create the toolbar buttons and menu items you need for your application and assign each of them a specific function. For the sample application, the template menu window provides most of what you need. You must add another top-level menu to it to provide access to the Customer and Order browse and maintenance windows.

Having built your windows, you now must pull the pieces together into a single application. In this section, you create a menu with entries for launching the Customer Browse and Order Selection windows that you created earlier.
Creating item categories

Next, you will create an item category by building the toolbar.

To build the toolbar:

1. Choose **Build → Toolbar and Menu Designer** from the AppBuilder main window. The **Toolbar and Menu Designer** appears:

   ![Toolbar and Menu Designer](image)

   For this sample application, you use the **Toolbar and Menu Designer** to define some menu items and insert them into your menu window. You can browse a bit through the toolbar buttons and menu items that are part of the framework tools themselves by expanding the **Item Categories, Bands**, and **SmartToolbar** nodes in the tree.

2. Select **ds-general** in the **Module** combo box. There are many predefined menu elements already in the Repository for the framework’s use. Filtering the TreeView to a single module improves its performance. This also ensures that your menu elements are conveniently grouped when they are stored in the Repository.

   The first subnode under **Toolbar & Menu Designer** is **Item Categories**. **Items** are all the individual elements of a Menu and/or Toolbar. In a menu, an item can be visualized as a menu item with a label. It can also simply be a label or a separator. In a toolbar, an item is represented as a button with either a text label or a bitmap. If an item is not just a label or a separator, it is associated with an action that occurs when it is selected. **Categories** are just a way of organizing items so that you can find them more easily. The framework organizes its items under categories such as Navigation and Commit.
3. Right-click the **Item Categories** node and choose **Add Category** from the pop-up menu that appears. A **Category** update page appears on the tab folder, as shown:

![Category update page](image.png)

4. Set the values in the following table on the update page, and leave the other settings at their default values:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category label</td>
<td>DynSports</td>
</tr>
<tr>
<td>Description</td>
<td>DynSports menu items</td>
</tr>
<tr>
<td>System owned</td>
<td>Unselected</td>
</tr>
</tbody>
</table>
You should leave the **System Owned** option checked only for items that the framework itself depends on. This option prevents anyone from changing the behavior of the tools without special user privileges. Clear this option for all other menu objects, as shown:

The update page is just a tab folder window like those you created. The dynamic windows you have built are realized at run time by one procedure (`rydyncontw.w`). This procedure reads all the related records out of the Repository and builds a dynamic window based on them.

The TreeView layout has its own dynamic window builder procedure (`rydyntreew.w`). The procedure reads the same data out of the Repository and builds a different visualization from it, in this case a frame that appears inside the larger TreeView window. This is an example of the flexibility of a Repository-based application: different programs can render the same data differently without needing any changes to the data itself. Your application can take on a new look just by creating a new procedure to interpret the data.

5. Click **Save**.
Creating menu items

Next, you must create the menu items that you use to build the DynSports menus.

To create menu items in the DynSports category:

1. Right-click the DynSports node that you created.

2. Select Add Item on the pop-up menu. An Item update page appears on the folder:

3. Set the following values in the update page, and leave the other settings at their default values:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item reference</td>
<td>oeCustMenuItem</td>
</tr>
<tr>
<td>Menu label</td>
<td>Customers</td>
</tr>
<tr>
<td>Description</td>
<td>Customer maintenance</td>
</tr>
<tr>
<td>System owned</td>
<td>Unselected</td>
</tr>
<tr>
<td>Tooltip</td>
<td>Customer maintenance</td>
</tr>
<tr>
<td>Style</td>
<td>Text only</td>
</tr>
</tbody>
</table>

The default Item Type is Action, which means an action takes place when this item is selected. This is what you want. An item can also be a Label, a visual Separator (rule), or a placeholder for something to be defined at run time.
The default **Action Type** is **Launch**, which is also what you want. You want to launch the appropriate browse window when this item is selected. You can also set an action to a logical property, publish a named event, run an internal in the linked object, or go to a URL. The **Run** option allows you to bring up an existing procedure window or run some other application function from a dynamic menu.

4. Type `oeCustBrowseWin` as the **Object filename** in the **Action** area and press the **TAB** key, as shown:

5. Click **Save**.

6. Right-click the **DynSports** node, and select **Add Item** on the pop-up menu.

7. Define a menu item for the **Order Maintenance** window, using the values in the following table:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item reference</td>
<td><code>oeOrderMenuITEM</code></td>
</tr>
<tr>
<td>Menu label</td>
<td>Orders</td>
</tr>
<tr>
<td>Description</td>
<td>Order maintenance</td>
</tr>
<tr>
<td>System owned</td>
<td>Unselected</td>
</tr>
<tr>
<td>Tooltip</td>
<td>Order maintenance</td>
</tr>
<tr>
<td>Style</td>
<td>Text only</td>
</tr>
<tr>
<td>Item type</td>
<td>Action</td>
</tr>
</tbody>
</table>
The Item page should look like the following:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action type</td>
<td>Launch</td>
</tr>
<tr>
<td>Object filename</td>
<td>oeOrderBrowseWin</td>
</tr>
</tbody>
</table>

8. **Save** your changes.

**Creating a submenu label**

Next, you will create a submenu label.

To create a Submenu item to group your new items together:

1. Right-click the **SubMenu** node under the **Item Categories** node, and choose **Add item** on the pop-up menu.

2. Set the following values in the update page, and leave the other settings at their default values:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item reference</td>
<td>oeSubMenuLabel</td>
</tr>
<tr>
<td>Item type</td>
<td>Label</td>
</tr>
<tr>
<td>Menu label</td>
<td>&amp;Order Entry</td>
</tr>
</tbody>
</table>
There is no Action Type because this item serves simply as a label to display on the top-level menu. When the submenu item is selected, it displays the other two menu items beneath it.

3. Click Save. The Toolbar and Menu Designer should look like the following:

### Creating a menu band

Next you must group the actions together into a band. A band is a set of related actions. You can visualize a band as a submenu (all the items that pop up when you select a top-level menu item) or a set of buttons that appear together in a toolbar. There are four types of bands:

- **MenuBar bands** — Define top-level menus.
- **Submenu bands** — Define the items under a top-level menu item.
- **Toolbar bands** — Define groups of related buttons.
- **Menu & Toolbar bands** — Have both visualizations.

Some bands can contain child bands in a hierarchical manner. Bands can also be reused in multiple OpenEdge SmartToolbars™.
To create a band:

1. Expand the Bands node to see the four band types.

2. Right-click on the Submenu Bands node, and choose Add Band from the pop-up menu. Note that two update pages are created: Band and Band Object, as shown:

3. Set the values on the Band update page as shown in the following table:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Band code</td>
<td>oeSubMenuBand</td>
</tr>
<tr>
<td>Description</td>
<td>Order Entry band</td>
</tr>
<tr>
<td>Band type</td>
<td>Submenu</td>
</tr>
<tr>
<td>Item reference used for label</td>
<td>oeSubMenuLabel</td>
</tr>
<tr>
<td>System owned</td>
<td>Unselected</td>
</tr>
</tbody>
</table>
4. Click Save. The **Toolbar and Menu Designer** should now look something like the following:

![Toolbar and Menu Designer](image)

**Adding band items**

Next, you will add items to the band.

**To add your items to the band:**

1. Right-click on the `oeSubMenuBand` node, and select **Add Item to Band** from the pop-up menu, as shown:

![Add Item to Band](image)
Note that two update pages are displayed: **Band item** and **Item (add to Band)**.

2. Set the following values on the **Band Item** update page as shown in the following table:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item reference</td>
<td>oeCustMenuItem</td>
</tr>
<tr>
<td>Item sequence</td>
<td>1</td>
</tr>
</tbody>
</table>

3. Click **Save**.

4. Add the **Order** menu item using the following values:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item reference</td>
<td>oeOrderMenuItem</td>
</tr>
<tr>
<td>Item sequence</td>
<td>2</td>
</tr>
</tbody>
</table>

5. Click **Save**. The **Toolbar and Menu Designer** should now look something like the following:
Adding the menu to your application

Next, you will add the menu to your application.

To add the OrderEntry menu dynamically to your menu window’s toolbar:

1. Choose the oeSubMenuBand node, and select the Band Object tab in the folder, as shown:

2. Click Add.

3. Type oeMenuWin for the Object Filename and press the TAB key.

   The menu controller window template you used defines top-level menu items for File, Window, and Help. It also has a placeholder for any number of menu items to be inserted into the MenuBar after the File menu. The name for this generic placeholder is a “DynamicMenu.”

4. Type DynamicMenu in the Item placeholder field.
5. Type 1 for the **Band Sequence**, and make sure that **Insert Submenu** is selected. The update page should look something like the following:

![Toolbar and Menu Designer](image)

6. Click **Save**, then close the **Toolbar and Menu Designer**.

---

**Running the completed application**

You have completed building the application. Now you can run the main menu window from the Dynamic Launcher. Because menu Repository data is cached differently on the client from the data for application objects, the **Clear repository cache** toggle box on the Dynamic Launcher is not sufficient to refresh the data for your new menu window. To see your latest work, you must select the **Clear data cache** and **Destroy ADM Super Procedures** toggle boxes as well. This option clears the running procedure instances on the client where Progress Dynamics stores menu data. Menu data is stored in running instances of certain ADM super procedures. You must destroy these instances to force restarting them with the latest definitions.

To run your sample application from the Dynamic Launcher:

1. Choose **Compile -> Dynamic Launcher** from the AppBuilder main window.
2. Type **oeMenuWin** in the **Dynamic Launcher**.
3. Select the **Clear data cache** and **Destroy ADM Super Procedures** toggle boxes, then click **Run**, as shown:

![Dynamic Launcher](image)

**Note:** A message box might appear that says the ADM Super Procedures are in use and you should close certain procedures or cancel. If you receive this message, close all the Progress Dynamics tools except the AppBuilder main window. The tools are built with the ADM and might tie up a procedure. If the message appears again, you must shut down Progress Dynamics and restart it. If you do have to restart, remember to reconnect to the DynSports database.

Your main menu window appears:

![DynSports Order Entry](image)

4. Choose **OrderEntry** → **Customers** and try out the **Customer Browse** window.

5. Choose **OrderEntry** → **Orders**.

6. Double-click an order to launch the **Order Maintenance** window.

7. Click **Modify** on the toolbar.

8. Click **Lookup** for the **Customer** field.

9. To change the Customer for the current Order, double-click a record in the browse.

You now have a working application. In the next chapter, you will look at some of the tools that Progress Dynamics provides for administering your applications. You will also learn to customize an application by providing translations for the sample application windows.
Customizing the Application

This chapter introduces you to some of the framework administration tools and how to customize your application. Specifically, you create translations for a window in the sample application and add a user who accesses those translations, as shown in the following sections:

- Using the Progress Dynamics Managers
- Creating a configuration XML file for your application
- Creating a shortcut for your application
- Running your application from the Desktop
- Web development
- Summing up
Using the Progress Dynamics Managers

There is more to Progress Dynamics than building application components. The framework includes a set of Managers, which control different aspects of a completed application, including:

- Security.
- User Profiles.
- Localization with translations of application screens and messages.
- Configuration of a distributed application.
- Connection parameters for databases, AppServers, and other components outside the bounds of the application itself.
- Session startup and communication with distributed objects.

Now that you have built your sample application, you can take a brief look at a few of the capabilities of the Managers. You can begin with delivering translations for your application based on a user’s login language.

Using the Localization Manager to translate a window

The Progress Dynamics Localization Manager lets you translate all visual elements of your application screens (including application messages) into any number of languages. You can also use the Language facility to provide alternative texts for any purpose, such as specializing labels, prompts, and messages for different user organizations.

To define a new language:

1. Choose Tools → Administration from the AppBuilder main window. The Administration window appears:

2. Choose Application → Language Control. The Language Control dialog box appears:
3. Click Add \( \text{Add} \) \( \text{Add} \). The Language Maintenance dialog box appears.

4. Type FR for the Language Code and French for the Language Name. For clarity, use the standard, two-letter acronym for the language, followed by an optional two-letter variant acronym, such as EN-US for US English and EN-UK for British English, as shown:

![Language Maintenance - FR](image)

5. Click Save, then exit the Language Control and Language Maintenance dialog boxes.

6. Close the Administration window.

Now that you have a code in the Repository for French, you can store French translations. Then, you can create a French language user. In Progress Dynamics, you can translate all visual elements. However, depending on the type of element, you use different tools:

- **Widgets** are translated screen by screen while they are running.
- **Menus** are translated using the Menu and Toolbar Designer.
- **Messages** are translated in the Message Control tool.

In this tutorial, you will translate widgets and menus. For translation of messages, see OpenEdge Development: Progress Dynamics Basic Development.

**Translating widgets**

You can translate any application screen while it is running.

**To translate a screen into French:**

1. Run oeMenuWin from the Dynamic Launcher. Remember to select all the toggle boxes.

   \( \text{Note:} \) Remember to reconnect the DynSports database if you have restarted your session since last running the application.

2. Choose Order Entry \( \rightarrow \) Orders.

3. Double-click on an order to launch your Order and Lines Browse.
4. Select the **Order Lines** tab, as shown:

![Order Lines tab](image1)

5. Choose **File**→**Translate**. The **Translate Window** dialog box appears:

![Translate Window](image2)

The Localization Manager builds the browse with all the translatable strings in the current window.

6. Select **French** in the **Language** combo box, and leave the **Source Language** on **English**. For a complete description of the options in the **Translate Window**, see *OpenEdge Development: Progress Dynamics Basic Development*.

7. Select the **orderby_price** fill-in widget.

8. Enter **Prix ligne de commande** in the **Translated label** column.
9. Enter the widget label translations from the following table in the browse:

<table>
<thead>
<tr>
<th>Widget type</th>
<th>Original label</th>
<th>Translated label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fill-in</td>
<td>Discount:</td>
<td>Rabais ligne de commande:</td>
</tr>
<tr>
<td>Fill-in</td>
<td>Extended Price:</td>
<td>Prix prolonge ligne de commande:</td>
</tr>
<tr>
<td>Fill-in</td>
<td>Quantity:</td>
<td>Quantite:</td>
</tr>
<tr>
<td>Tab</td>
<td>Order</td>
<td>Ordre</td>
</tr>
<tr>
<td>Tab</td>
<td>Order Lines</td>
<td>Ligne de commandes</td>
</tr>
</tbody>
</table>

10. Click OK, and close your application windows.

Translating menus

To translate a menu into French:
1. Choose Build → Toolbar and Menu Designer in the AppBuilder main window.
2. Select the ds-general module.
3. Choose Item Categories → DynSports → Customers. The Customers details appear in the update frame, as shown:
4. Click **Translate Menu Item**. The **Menu Item Translation** dialog box appears:

5. Select **French** in the **To** combo box.

6. Select **Menu Item Label** in the browser.

7. Type **Clientes** in the **French** column and press **TAB**.

8. Click **OK**.

9. Add the translations as shown in the following table:

<table>
<thead>
<tr>
<th>Menu element</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item Categories→ DynSports→ Orders</td>
<td>Commandes</td>
</tr>
<tr>
<td>Item Categories→ SubMenu→ Order Entry</td>
<td>Saisie de commandes</td>
</tr>
</tbody>
</table>

10. Exit the **Toolbar and Menu Designer**.

Before you can see the effects of your translations, you must log on as a French user. So, the next step is to create a French user.
Creating a new user

You create users in the Security Control TreeView, which uses data maintained by the Profile Manager.

To create a new user with a login language of French:

1. Choose Security → Security Control from the Administration window. The Security Control TreeView appears:

2. Choose Security control → Security maintenance → Users in the TreeView, as shown:

3. Click Add from the User Maintenance toolbar.
4. Use the values from the following table for your new user:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>User login name</td>
<td>jacques</td>
</tr>
<tr>
<td>Full name</td>
<td>Jacques Forgeron</td>
</tr>
<tr>
<td>User category</td>
<td>At guru level (Expert)</td>
</tr>
<tr>
<td>Development User</td>
<td>selected</td>
</tr>
<tr>
<td>Based on profile</td>
<td>admin</td>
</tr>
<tr>
<td>Default language</td>
<td>FR / French</td>
</tr>
</tbody>
</table>

5. Click **Save**, then exit the **Security Control** TreeView, as shown:

Now that you have a French user, you can see the results of your translations.

**To see the translations:**

1. End your session.
2. Restart your Dynamics Tutorial Development session.
3. Type **jacques** as the login name in the **Application login** window.

5. Launch oeMenuWin from the Dynamic Launcher.

**Note:** If you receive the “ADM super procedures in use” message, you must close your session and restart the framework. Remember to reconnect the DynSports database after restarting.

6. Choose Saisie de commandes → Commandes from the DynSports Order Entry window. The Order Selection window appears.

7. Double-click an order to select it. The Order and Lines Browse appears.

8. Select the Lignes de commande tab and see the translations, as shown:

![Order and Lines Browse](image)

9. Close everything and restart your session as the admin user.
Using the Connection and Configuration Managers

To complete your work, follow these steps to define a session type to start the application. The session type includes all the necessary information for launching a run-time, AppServer, development, or Web UI version of Progress Dynamics, making the correct connections to the Repository and the DynSports database, and launching the application itself. It is possible to configure session types to support many different platforms and deployment configurations. This tutorial covers only the basic settings.

To create a session type:

1. Choose **Session** → **Logical Service Control** from the **Administration** window. The **Logical Service Control** window appears:

![Logical Service Control window]

2. Click **Add** to add a new service for the DynSports database. The **Logical Service Maintenance** window appears.

3. Set the values from the following table for the new service:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical service code</td>
<td>dynsports</td>
</tr>
<tr>
<td>Logical service description</td>
<td>DynSports application database</td>
</tr>
<tr>
<td>Service type</td>
<td>Database Connection (Database)</td>
</tr>
<tr>
<td>Can run locally</td>
<td>Selected</td>
</tr>
<tr>
<td>System owned</td>
<td>Not selected</td>
</tr>
<tr>
<td>Write to config</td>
<td>Selected</td>
</tr>
<tr>
<td>Connect at startup</td>
<td>Selected</td>
</tr>
</tbody>
</table>
The Can run locally toggle box enables you to start a session type with just a local database connection. The Write to config toggle box tells the framework to write out the information you are defining to a generated configuration file. This makes the service you define a part of the startup parameters for the session, not just something available to be run on demand after the session is started. The Connect at startup toggle box tells the framework whether or not to immediately attempt a connection when starting. If a service is not needed immediately, it might be better to not add the overhead to the startup of your application.

For example:

4. Click Save, then exit the maintenance and control dialog boxes.

5. Choose Session → Physical Service Control from the Administration window. The Physical Service Control window appears:

6. Click Add to add a new service for the local DynSports database connection. The Physical Service Maintenance window appears.
7. Set the following values on the **Details** tab for the new service:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical service code</td>
<td>dynsports</td>
</tr>
<tr>
<td>Physical service description</td>
<td>DynSports application database</td>
</tr>
<tr>
<td>Service type</td>
<td>Database Connection / Database</td>
</tr>
<tr>
<td>Physical Database Name (-db)</td>
<td>dynsports</td>
</tr>
<tr>
<td>Network Type (-N)</td>
<td>TCP</td>
</tr>
<tr>
<td>Host (-H)</td>
<td>localhost</td>
</tr>
<tr>
<td>Service (-S)</td>
<td>dynsports</td>
</tr>
</tbody>
</table>

The other options can be left blank.

Although the DynSports database is local on your machine, you will still use a network connection to see how it is done, as shown:

![Physical Service Configuration](image)

Remember that you set up an entry for the DynSports database in your Services file in the “Adding an entry to the Windows Services file” section on page 2–14.

8. Click **Save**, then exit the maintenance and control windows.
Using the Session Manager to edit a session type

When you create a new session type, you can start from nothing or edit an existing session type. For the tutorial, you can extend the Default session type to add a DynSports database connection and launch your DynSports Order Entry window, rather than creating a whole new session type.

To edit the Default session type:

1. Choose Session → Session Type Control from the Administration window. The Session Type Maintenance TreeView appears:

2. Choose the Session Type → Basic → Dynamics → DynDBBound → DynCS node.
3. Right-click the Default node and choose Add Session Type from the pop-up menu. A Details update page appears:

4. Type DynSports for the Session type code and DynSports sample application for the Session type description.

5. Select the WIN32 toggle box in the Valid Operating Systems group.

6. Select GUI Client from the Available physical session types list.

7. Click Add to selected fields to move it to the Selected types list.

8. Click Save.

9. Expand the DynSports node.
Using the Progress Dynamics Managers

10. Right-click the **Session Services** node and select **Add Session Service** from the pop-up menu. A **Details** update page appears:

11. Type *dynsports* for the **Physical service code** and press the **TAB** key.

12. Type *dynsports* for the **Logical service code** and press the **TAB** key.
13. Click **Save**. The **Session Type Maintenance** TreeView should now look like the following:
14. Right-click the **Session Properties** node and choose **Add Session Property** from the pop-up menu. A Details update page appears:

This page shows the logical properties that are defined for this session type. However, because your new **dynsports** session type extends the **Default** session type, it inherits properties from the **Default** session type and its parent session types.

The `run_local` property indicates that the session can run stand-alone, as you want it to do. The `startup_procedure` property defines which OpenEdge procedure is run when the session starts up. In the **Default** session type, its value is `ICFOBJ|afallmencw`. The `ICFOBJ` parameter indicates that you are launching a dynamic container. The parameter value, `afallmencw`, is the **Progress Dynamics Administration** menu. But, for your DynSports session type, you want to launch a different dynamic window: the **DynSports Order Entry** menu controller that you created, `oeMenuWin`.

15. Type `startup_procedure` for the **Property name** and press the **TAB** key.
16. Type ICFOBJoeMenuWin as the Property value, and click Save, as shown:

![Image of the interface showing the property value being set to ICFOBJoeMenuWin]

**Note:** The separator character is a pipe (|).

---

**Creating a configuration XML file for your application**

In the “Creating configuration files” section on page 2–2, you copied some configuration files to your working directory. One of those files was the icfconfig.xml file. This file stores all the session types that let you specify which actions Progress Dynamics takes when launched, for example, connecting servers, connecting databases, starting managers, and launching containers.

The DynSports session type that you created currently exists only in the Repository. To use it, you must export this information to the icfconfig.xml file. Once the session type is listed in the configuration file, you can create a Desktop shortcut that uses the information to launch your application.
To generate a configuration file for the sample application:

1. Choose Option → Generate Configuration File from the Session Type Maintenance TreeView’s menu. The Generate Configuration File window appears:

   ![Generate Configuration File Window]

   - Type `<wrk>Tutorial\dysports.xml` for the File Name, where `<wrk>` is your Progress Dynamics working directory.

   - Select DynSports from the Available Session Types list and choose Add. The DynSports session type moves into the Selected Session Types list. The generation process only adds the selected session types into the configuration file. This enables you to tailor exactly what can be accessed from any deployed version of your application. If the configuration file does not have the session types that support the development environment, the development environment cannot be started successfully using that configuration file.

2. Click Generate Config File to re-create the configuration file. A message dialog box appears when the file is completed:

   ![Save Configuration XML File]

3. Click OK, and exit the Generate Configuration File and Session Type Maintenance windows.
6. Open the dynsports.xml file in a Web browser, and see the results of your changes, as shown:

```xml
- <sessions>
  - <session SessionType="DynSports">
      - <auto_dump_entity_cache>YES</auto_dump_entity_cache>
      - <bound_if Hick>yes</bound_if Hick>
      - <DynamicVersion>2.1A</DynamicVersion>
      - <ICFCM_AppServer>AppServerConnectionManager</ICFCM_AppServer>
      - <ICFCM_Database>DatabaseConnectionManager</ICFCM_Database>
      - <ICFCM_JMS>JM2ConnectionManager</ICFCM_JMS>
      - <ICFCM_WebService>WebServiceConnectionManager</ICFCM_WebService>
      - <login_procedure>Guest</login_procedure>
      - <physical_session_list>QUIT</physical_session_list>
      - <print_preview_preference>
      - <print_preview_stylesheet>default.xmll</print_preview_stylesheet>
      - <root_directory></root_directory>
      - <run_local>YES</run_local>
      - <session_date_format>04/30/1995</session_date_format>
      - <session_year_offset>1995</session_year_offset>
      - <startup_procedure>ICFORDumpMenuWin</startup_procedure>
      - <UseThinRendering>YES</UseThinRendering>
      - <valid_os_list>WIN32</valid_os_list>
      </session>
  - </sessions>

- <services>
  - <service>
      - <ServiceType>AppServer</ServiceType>
      - <ServiceName>Astra</ServiceName>
      - <PhysicalService>ICFDevAS</PhysicalService>
      - <ConnectParams>Default\PUBLIC\ICFDevAS\AppService.xmll</ConnectParams>
      - <DefaultService>YES</DefaultService>
      - <ICConnectAtStartup>YES</ICConnectAtStartup>
      - <ICCanRunLocal>YES</ICCanRunLocal>
  - </service>
  - <service>
      - <ServiceType>Database</ServiceType>
      - <ServiceName>ICFDB</ServiceName>
      - <PhysicalService>ICFDB</PhysicalService>
      - <ConnectParams>Default\PUBLIC\ICFDevAS\AppService.xmll</ConnectParams>
      - <DefaultService>YES</DefaultService>
      - <ICConnectAtStartup>YES</ICConnectAtStartup>
      - <ICCanRunLocal>YES</ICCanRunLocal>
  - </service>
- <service>
```
Creating a shortcut for your application

You can now create a shortcut for the Windows Desktop that launches your application using the new customized Session Type.

To create a shortcut:

1. Open the Progress Dynamics Tutorial folder where you created the shortcuts in the “Creating startup scripts and shortcuts” section on page 2–3.
2. Copy the Dynamics Tutorial Development shortcut.
3. Right-click in the folder and choose Paste Shortcut from the pop-up menu.
4. Right-click the new shortcut, and choose Properties on the pop-up menu.
5. Type DynSports Order Entry Application in the fill-in on the General tab.
6. Select the Shortcut tab.
7. Change ICFSESSTYPE=ICFDev to ICFSESSTYPE=DynSports and ICFCONFIG=icfconfig.xml to ICFCONFIG=dynsports.xml in the Target field.
8. Click OK to save your changes.

Running your application from the Desktop

You now have a startup icon that launches the dynamic menu window of your sample application. Because your DynSports session type connects to your DynSports database in addition to the Repository database, everything you need is taken care of automatically. If you need to make changes to the definition of your startup parameters, only the centrally located configuration information changes.

To start your application from the shortcut:

1. If necessary, close the AppBuilder, but do not stop the database servers. You can run the application without the Progress Dynamics Development environment, but you do still need the database servers.
2. Double-click on your DynSports Order Entry Application shortcut to start your application.
3. Login as either admin or jacques.
Web development

With Progress Dynamics, the same repository definitions that generate a Windows graphic user interface (GUI) can also generate a Web UI. The Web Request and User Interface Managers can translate the abstract repository definitions into Dynamic HTML. The DHTML employs Cascading Style Sheets (CSS) and JavaScript components (JS) to create a Web UI with the features of a Windows GUI. The JavaScript files provide the behavior. The CSS files provide the look and feel of the application.

For more on Web development, OpenEdge WebSpeed®, the new Managers, and using CSS files and JavaScript, see *OpenEdge Development: Progress Dynamics Web Development Guide*. Sample CSS files for use with the DynSports sample application are provided in the `<install-dir>Tutorial` directory, if you want to experiment with running the sample application to the Web.

Summing up

Here is what you have accomplished in your tour of the Progress Dynamics framework:

- Set up a custom development environment for the tutorial.
- Set a unique site number for your database, allowing you to share dynamic object definitions with any other Repository database in the world.
- Created Product and Product Module definitions to organize your application objects.
- Imported entity definitions from your application database into the Repository.
- Generated SDOs, dynamic browsers, and dynamic viewers for your application.
- Added dynamic Combos and a dynamic Lookup to generated objects providing users with lists of valid key values from related tables.
- Created dynamic browse windows where users can select records to update.
- Explored how to filter that data.
- Created dynamic tab folder windows displaying and maintaining several levels of related data and customized the layout of one of those windows.
- Created a dynamic menu window to provide top-level access to the parts of your application.
- Created menu items and a menu band to launch your browse windows and added them to the menu window.
- Defined a new language for your application and translated the labels in a folder.
• Defined a new user and a login profile for that user.
• Created a new Service to define how to start the DynSports database.
• Modified a Session Type to start your database Service.
• Created a shortcut to start your application using your customized Session Type.

You can learn a great deal more about building and managing applications with Progress Dynamics from *OpenEdge Development: Progress Dynamics Basic Development*, and from the rest of the product documentation distributed with the product. Table 5–1 provides a list of books in the Progress Dynamics documentation set you should go to for information about specific topics.

**Table 5–1: Where to go from here**

<table>
<thead>
<tr>
<th>For information about . . .</th>
<th>Go to . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installing and viewing the Progress Dynamics online documentation.</td>
<td><em>OpenEdge Getting Started: Installation and Configuration</em></td>
</tr>
<tr>
<td>Advanced programming topics (including examples).</td>
<td><em>OpenEdge Development: Progress Dynamics Advanced Development</em></td>
</tr>
<tr>
<td>SmartObject procedures, functions, and properties.</td>
<td><em>OpenEdge Development: ADM Reference</em></td>
</tr>
<tr>
<td>Developing applications using the Progress Dynamics framework.</td>
<td><em>OpenEdge Development: Progress Dynamics Basic Development</em></td>
</tr>
<tr>
<td>Developing and deploying Web browser-based Progress Dynamics applications.</td>
<td><em>OpenEdge Development: Progress Dynamics Web Development Guide</em></td>
</tr>
<tr>
<td>Configuring and managing Progress Dynamics.</td>
<td><em>OpenEdge Development: Progress Dynamics Administration</em></td>
</tr>
<tr>
<td>Common administrative procedures that you might need to perform to support all of the applications that you develop using Progress Dynamics.</td>
<td><em>OpenEdge Development: Progress Dynamics Administration</em></td>
</tr>
<tr>
<td>Integration with optional third-party tools, such as ERwin.</td>
<td><em>OpenEdge Development: Progress Dynamics Administration</em></td>
</tr>
<tr>
<td>Progress Dynamics managers.</td>
<td><em>OpenEdge Development: Progress Dynamics Managers API Reference</em></td>
</tr>
<tr>
<td>The structure of the Progress Dynamics Repository database.</td>
<td><em>OpenEdge Development: Progress Dynamics Repository Reference</em></td>
</tr>
</tbody>
</table>
Index

A
Action Type option 4–61
ActiveX 4–56
ADE. See also Application Development Environment (ADE) 1–4
Administration window 3–6
AppBuilder 1–4
Progress Dynamics 1–4
Application running 5–21
standard and reusable components 1–5
Application Development Environment (ADE) 1–4
Application objects generating 3–16
Architecture 1–2

B
Band defined 4–63
Base Query String 4–12
Bottom of grid 4–46
Browse sequence setting 4–13
Browse windows creating 4–22
Browses 3–16
Built-in framework managers 1–6
Cache clearing 4–31
Calculator 4–45
Can Run Locally option 5–10, 5–12
Categories 4–56
Clear cache option 4–31
Company login 2–15
Completed application 4–2
Configuration Manager 5–10
Configuration XML file 5–18
Connection Manager 5–10
Container Builder 4–10
Containers 4–23
Control objects 1–6
csutfoldwin 4–33
Custom super procedure 3–24
customerfullb 4–25

D
Data logic procedure
defined 1–5
Data objects 1–5
defined 1–5
Databases
application 3–4
connecting 3–4
DynSports 2–10
repository 1–4
Deleting objects 4–35
Dependent windows 4–33
Design window
Browse 3–23
SDO 3–21
Desktop shortcut
creating 5–21
Destroy ADM Super Procedures option
4–68
Dynamic combo 1–5
adding to a Viewer 4–6
defined 1–6
Dynamic Launcher 4–30
Dynamic Lookup
adding 4–10
Dynamic lookup 1–5
defined 1–6
Dynamic menu controller 4–55
Dynamic objects
defined 1–4
Dynamic Properties window 4–4
Dynamic windows
creating 4–22
Dynamics ADE. See also Application Development Environment (ADE) 1–4
Dynamics AppBuilder 1–4
Dynamics Repository 1–4
defined 1–4
Dynamics. See Progress Dynamics
DynField object type 4–33
DynObj type 4–23
DynSports
application database 2–10
connecting 3–4

E
Editing objects 4–3
EMP Product Module
creating 3–8
Entities
importing 3–8
Entity Import dialog box 3–8
Environment Managers
list of 1–6
using 5–2
ERwin 1–4, 1–5
Excel 4–33

F
Field name separators 3–10
Field objects 1–6
defined 1–6
Files
configuration 5–18
icfconfig.xml 5–18
Filter Records button 4–32
Finished application 4–2
Folder Window to Launch option 3–24
Follow joins option 3–17
Foreign Fields 4–40
Framework managers 1–6
fullb suffix 3–17
fullo suffix 3–17
G
General Product Module
creating 3–8
Generating objects 3–16
Grid
bottom section 4–46
I
icfconfig.xml 5–18
icfdb database 1–4
Importing entities 3–8
Independent Windows 4–23
Index information 4–32
Instance Properties window 4–12
Internal joins
following 3–17
Item Type option 4–61
Items 4–56
J
Joins 3–17
K
Key values 4–40
L
Launching containers 4–30
Layout
changing 4–46
Link field option 4–13
Linked widget column 4–13
Links
Customer browse window 4–28
Customer orders browse 4–43
Order and lines browse 4–51, 4–53
Logical Services Control window 5–10
Login company 2–15
Login language 5–7
Login window 2–15, 3–3
Lookup
adding 4–10
M
Main menu window
creating 4–55
Maintain System Data option 5–7
Maintenance window
creating 4–48
Order 4–48
Managers 1–6
list of 1–6
using 5–2
Manuals
where to go from here 5–23
Menu controllers 4–55
Menu designer 4–56
Migrating
static objects 1–4
Modules
creating 3–6
N
Naming conventions
object suffixes 3–17
New user
creating 5–7
Nonvisual objects
showing in grid 4–25
O
Object Controllers 4–23
Object Generator tool 3–16
Query 3–21
Object repository 1–4
Object types
DynField 4–33
Objects
deleting from grid 4–35
dynamic 1–4
dynamic physical and logical 1–4
editing 4–3
### Index

<table>
<thead>
<tr>
<th>Page Labels</th>
<th>4–35</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pages</td>
<td>4–37</td>
</tr>
<tr>
<td>Physical Service Control window</td>
<td>5–11</td>
</tr>
<tr>
<td>Prefix length</td>
<td>tables 3–10</td>
</tr>
<tr>
<td>Product Control window</td>
<td>3–6</td>
</tr>
<tr>
<td>Product Maintenance window</td>
<td>3–6</td>
</tr>
<tr>
<td>Product Modules</td>
<td>defining 3–6</td>
</tr>
<tr>
<td>Products</td>
<td>defining 3–6</td>
</tr>
<tr>
<td>Profile Manager</td>
<td>5–7</td>
</tr>
<tr>
<td>Progress Dynamics</td>
<td>application components 1–5</td>
</tr>
<tr>
<td></td>
<td>data objects 1–5</td>
</tr>
<tr>
<td></td>
<td>defined 1–1</td>
</tr>
<tr>
<td></td>
<td>managers 1–6</td>
</tr>
<tr>
<td></td>
<td>repository function</td>
</tr>
<tr>
<td></td>
<td>tool set 1–4</td>
</tr>
<tr>
<td></td>
<td>UI objects 1–6</td>
</tr>
<tr>
<td>Properties</td>
<td>Session Type 5–17</td>
</tr>
<tr>
<td>Properties window</td>
<td>4–3</td>
</tr>
<tr>
<td>Property sheet</td>
<td>Browse 3–24</td>
</tr>
</tbody>
</table>

### R

<table>
<thead>
<tr>
<th>Records</th>
<th>filtering 4–32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative layout</td>
<td>4–46</td>
</tr>
<tr>
<td>Relative path directory</td>
<td>3–7</td>
</tr>
<tr>
<td>Repository. See Dynamics Repository</td>
<td></td>
</tr>
<tr>
<td>Run Locally option</td>
<td>5–10, 5–12</td>
</tr>
<tr>
<td>Run Persistent option</td>
<td>4–31</td>
</tr>
<tr>
<td>Running containers</td>
<td>4–30</td>
</tr>
<tr>
<td>Running the application</td>
<td>5–21</td>
</tr>
</tbody>
</table>

### S

| SBO. See SmartBusinessObject (SBO) |
| SDO. See SmartDataObject (SDO) |
| SDV. See SmartDataViewer (SDV) |
| Separators | field names 3–10 |
| Sequence | browse 4–13 |
| Session Manager | 5–13 |
| Session types | editing 5–13 |
| Shortcut | creating 5–21 |
| Show nonvisual objects button | 4–25 |
| Show visual objects button | 4–25 |
| SmartBusinessObject (SBO) | 1–5 |
| defined 1–5 |
| SmartDataField icon | 4–6 |
| SmartDataObject (SDO) | 1–5, 3–16 |
| defined 1–5, 1–6 |
| SmartDataViewer (SDV) | 4–3 |
| defined 1–6 |
| Source code management (SCM) | RVDB 1–6 |
| Static objects | defined 1–4 |
| Suffixes | 3–17 |
Super procedures
   custom 3–24
   destroy option 4–68
System Owned option 4–61

T
 Templates 4–23
 Toolbar and Menu Designer 4–56
 Toolset 1–4
 TreeView window layout 4–56
 Type property 4–3

U
 UI objects 1–6
    defined 1–5
 User interface objects 1–5
 Users
    adding 5–7

V
 Viewers 3–16
 Viewing objects 3–20
 viewv suffix 3–17
 Visual objects
   showing in grid 4–25

W
 Window container objects
   defined 1–6
 Window Name attribute 4–28
 Window object. See Window container objects
 Write to Config option 5–10, 5–12

X
 XML configuration file 5–18