



Getting Started with Progress Dynamics®

© 2005 Progress Software Corporation. All rights reserved.

Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. This manual is also copyrighted and all rights are reserved. This manual may not, in whole or in part, be copied, photocopied, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from Progress Software Corporation.

The information in this manual is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear in this document.

The references in this manual to specific platforms supported are subject to change.

A [Stylized], Allegrix, Allegrix & Design, Business Empowerment, eXcelon, ObjectStore, PeerDirect, Progress, Progress Dynamics, Powered by Progress, Empowerment Center, Progress Empowerment Center, Progress Empowerment Program, Progress Fast Track, Progress OpenEdge, Progress Profiles, Partners in Progress, Partners en Progress, Progress en Partners, Progress in Progress, P.I.P., Progress Results, Progress Software Developers Network, ProVision, ProCare, ProtoSpeed, SmartBeans, SpeedScript, Technical Empowerment, and WebSpeed are registered trademarks of Progress Software Corporation or one of its subsidiaries or affiliates in the U.S. and/or other countries. AccelEvent, A Data Center of Your Very Own, AppsAlive, AppServer, ASPen, ASP-in-a-Box, BusinessEdge, Cache-Forward, Fathom, Future Proof, IntelliStream, ObjectCache, ObjectStore Event Engine, ObjectStore RFID Accelerator, ObjectStore Trading Accelerator, OpenEdge, POSSE, POSSENET, ProDataSet, Progress Business Empowerment, Progress for Partners, PSE Pro, PS Select, SectorAlliance, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, WebClient, and Who Makes Progress are trademarks or service marks of Progress Software Corporation or one of its subsidiaries or affiliates in the U.S. and other countries.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Any other trademarks and service marks contained herein are the property of their respective owners.

February 2005



Product Code: 4490
Item Number: 101925;V2.1B

Contents

Preface	ix
Purpose	ix
Audience	ix
Organization of this manual	ix
Typographical conventions	x
Syntax notation	xi
Progress messages	xv
What Is Progress Dynamics?	1-1
1.1 Object repository	1-4
1.2 Integrated development tools	1-5
1.3 Standard, reusable application components	1-6
1.4 Built-in environment managers	1-7
Setting Up a Tutorial Environment	2-1
2.1 Creating tutorial working directories	2-2
2.2 Creating configuration files	2-2
2.3 Creating startup scripts and shortcuts	2-3
2.3.1 Copying the shortcuts to a Desktop folder	2-3
2.3.2 Copying the database scripts	2-4
2.3.3 Modifying the shortcuts	2-4
2.3.4 Modifying the database scripts	2-5
2.4 Creating a tutorial Repository	2-7
2.5 Creating the application database	2-12
2.5.1 Creating an empty database	2-13
2.5.2 Loading the database schema	2-14
2.5.3 Loading the data	2-15

2.6	Adding an entry to the Windows Services file	2-16
2.7	Setting your PROPATH	2-17
2.7.1	Logging in to the framework tools	2-17
2.7.2	Using the Propath Editor	2-19
Generating Initial Objects		3-1
3.1	Logging in to the framework tools	3-3
3.2	Using the AppBuilder with Progress Dynamics	3-3
3.3	Connecting to the application database	3-4
3.4	Creating a new product and modules	3-6
3.5	Importing tables as entities into the Repository	3-9
3.6	Customizing the entity data	3-13
3.7	Generating application objects	3-20
3.8	Viewing your application objects	3-25
3.8.1	Viewing an SDO	3-25
3.8.2	Viewing a dynamic Browse	3-27
Building the Sample Application		4-1
4.1	Editing your application objects	4-3
4.1.1	Adding a dynamic Combo to your viewer	4-6
4.1.2	Adding a dynamic Lookup to a viewer	4-11
4.1.3	Adding more dynamic Combos	4-21
4.1.4	Modifying the Orderline viewer	4-24
4.1.5	Hiding object fields in browses	4-27
4.2	Creating browse windows	4-30
4.2.1	Defining the basic window	4-31
4.2.2	Replacing template objects on the folder page	4-33
4.2.3	Creating an Order Browse	4-39
4.2.4	Running your dynamic browse window	4-39
4.3	Creating folder windows	4-43
4.3.1	Adding another page to the folder window	4-48
4.3.2	Setting the SDO foreign fields	4-52
4.3.3	Adding browsers on the new page	4-53
4.3.4	Setting links for the new page	4-54
4.3.5	Running your folder window	4-57
4.3.6	Changing the layout of your dynamic window	4-60
4.4	Creating an order maintenance window	4-63
4.4.1	Adding an Order page	4-64
4.4.2	Linking the Order page	4-66
4.4.3	Adding an Order lines page	4-67
4.4.4	Adding objects to the Order lines page	4-68
4.4.5	Linking the Order Lines page	4-68
4.5	Creating a main menu window for your application	4-71

4.6	Using the Toolbar and Menu Designer	4-72
4.6.1	Creating item categories	4-73
4.6.2	Creating menu items	4-76
4.6.3	Creating a submenu label	4-79
4.6.4	Creating a menu band	4-80
4.6.5	Adding band items	4-83
4.6.6	Adding the menu to your application	4-85
4.7	Running the completed application	4-86
Customizing the Application		5-1
5.1	Using the Progress Dynamics Managers	5-2
5.1.1	Using the Localization Manager to translate a window	5-2
5.1.2	Creating a new user	5-9
5.1.3	Using the Connection and Configuration Managers	5-12
5.1.4	Using the Session Manager to edit a session type	5-16
5.2	Creating a configuration XML file for your application	5-21
5.3	Creating a shortcut for your application	5-24
5.4	Running your application from the Desktop	5-24
5.5	Web development	5-25
5.6	Summing up	5-25
Index		Index-1

Tables

Table 2–1:	Tutorial working directories	2–2
Table 4–1:	Order Browse window properties	4–39
Table 5–1:	Where to go from here	5–26

Figures

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

Preface

Purpose

This manual provides an overview of Progress Dynamics®, the repository-based Progress® 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 Progress 4GL, Progress development tools, and the Application Development Model (ADM2).

Organization of this manual

This guide is organized as follows:

[Chapter 1, “What Is 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.

Typographical conventions

This manual uses the following typographical conventions:

- **Bold typeface** indicates:
 - Commands or characters that the user types
 - That a word carries particular weight or emphasis
 - Names of user interface elements
- *Italic typeface* indicates:
 - Progress variable information that the user supplies
 - New terms
 - Titles of complete publications
- Monospaced typeface indicates:
 - Code examples
 - System output
 - Operating system filenames and pathnames

The following typographical conventions are used to represent keystrokes:

- Small capitals are used for Progress key functions and generic keyboard keys.

END-ERROR, GET, GO
ALT, CTRL, SPACEBAR, TAB

- When you have to press a combination of keys, they are joined by a hyphen. You press and hold down the first key, then press the second key.

CTRL-X

- When you have to press and release one key, then press another key, the key names are separated with a space.

ESCAPE H
ESCAPE CURSOR-LEFT

Syntax notation

The syntax for each component follows a set of conventions:

- Uppercase words are keywords. Although they are always shown in uppercase, you can use either uppercase or lowercase when using them in a procedure.

In this example, **ACCUM** is a keyword:

Syntax

```
ACCUM aggregate expression
```

- Italics identify options or arguments that you must supply. These options can be defined as part of the syntax or in a separate syntax identified by the name in italics. In the **ACCUM** function above, the *aggregate* and *expression* options are defined with the syntax for the **ACCUM** function in the [Progress Language Reference](#).
- You must end all statements (except for **DO**, **FOR**, **FUNCTION**, **PROCEDURE**, and **REPEAT**) with a period. **DO**, **FOR**, **FUNCTION**, **PROCEDURE**, and **REPEAT** statements can end with either a period or a colon, as in this example:

```
FOR EACH Customer:  
  DISPLAY Name.  
END.
```

- Square brackets (`[]`) around an item indicate that the item, or a choice of one of the enclosed items, is optional.

In this example, `STREAM stream`, `UNLESS-HIDDEN`, and `NO-ERROR` are optional:

Syntax

```
DISPLAY [ STREAM stream ] [ UNLESS-HIDDEN ] [ NO-ERROR ]
```

In some instances, square brackets are not a syntax notation, but part of the language.

For example, this syntax for the `INITIAL` option uses brackets to bound an initial value list for an array variable definition. In these cases, normal text brackets (`[]`) are used:

Syntax

```
INITIAL [ constant [ , constant ] . . . ]
```

NOTE: The ellipsis (`. . .`) indicates repetition, as shown in a following description.

- Braces (`{ }`) around an item indicate that the item, or a choice of one of the enclosed items, is required.

In this example, you must specify the items `BY` and *expression* and can optionally specify the item `DESCENDING`, in that order:

Syntax

```
{ BY expression [ DESCENDING ] }
```

In some cases, braces are not a syntax notation, but part of the language.

For example, a called external procedure must use braces when referencing arguments passed by a calling procedure. In these cases, normal text braces (`{ }`) are used:

Syntax

```
{ &argument-name }
```

- A vertical bar (|) indicates a choice.

In this example, EACH, FIRST, and LAST are optional, but you can only choose one:

Syntax

```
PRESELECT [ EACH | FIRST | LAST ] record-phrase
```

In this example, you must select one of *logical-name* or *alias*:

Syntax

```
CONNECTED ( { logical-name | alias } )
```

- Ellipses (. . .) indicate that you can choose one or more of the preceding items. If a group of items is enclosed in braces and followed by ellipses, you must choose one or more of those items. If a group of items is enclosed in brackets and followed by ellipses, you can optionally choose one or more of those items.

In this example, you must include two expressions, but you can optionally include more. Note that each subsequent expression must be preceded by a comma:

Syntax

```
MAXIMUM ( expression , expression [ , expression ] . . . )
```

In this example, you must specify MESSAGE, then at least one of *expression* or SKIP, but any additional number of *expression* or SKIP is allowed:

Syntax

```
MESSAGE { expression | SKIP [ ( n ) ] } . . .
```

In this example, you must specify `{include-file`, then optionally any number of *argument* or `&argument-name = "argument-value"`, and then terminate with `}`:

Syntax

```
{ include-file  
  [ argument | &argument-name = "argument-value" ] ... }
```

- In some examples, the syntax is too long to place in one horizontal row. In such cases, **optional** items appear individually bracketed in multiple rows in order, left-to-right and top-to-bottom. This order generally applies, unless otherwise specified. **Required** items also appear on multiple rows in the required order, left-to-right and top-to-bottom. In cases where grouping and order might otherwise be ambiguous, braced (required) or bracketed (optional) groups clarify the groupings.

In this example, `WITH` is followed by several optional items:

Syntax

```
WITH [ ACCUM max-length ] [ expression DOWN ]  
    [ CENTERED ] [ n COLUMNS ] [ SIDE-LABELS ]  
    [ STREAM-IO ]
```

In this example, `ASSIGN` requires one of two choices: either one or more of *field*, or one of *record*. Other options available with either *field* or *record* are grouped with braces and brackets. The open and close braces indicate the required order of options:

Syntax

```
ASSIGN { { [ FRAME frame ]  
          { field [ = expression ] }  
          [ WHEN expression ]  
        } ...  
      | { record [ EXCEPT field ... ] }  
    }
```

Progress messages

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

- Execution messages inform you of errors encountered while Progress is running a procedure (for example, if Progress cannot find a record with a specified index field value).
- Compile messages inform you of errors found while Progress is reading and analyzing a procedure prior to 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 Progress is getting ready to execute (for example, if you entered an invalid startup parameter).

After displaying a message, Progress 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 following execution messages.
- Returns to the Progress Procedure Editor so that you can correct an error in a procedure. This is the usual action taken following compiler messages.
- Halts processing of a procedure and returns immediately to the Procedure Editor. This does not happen often.
- Terminates the current session.

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

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

Use Progress online help to get more information about Progress messages. Many Progress tools include the following Help menu options to provide information about messages:

- Choose **Help–Recent Messages** to display detailed descriptions of the most recent Progress message and all other messages returned in the current session.
- Choose **Help–Messages**, then enter the message number to display a description of any Progress message. (If you encounter an error that terminates Progress, make a note of the message number before restarting.)
- In the Procedure Editor, press the **HELP** key (**F2** or **CTRL–W**).

What Is Progress Dynamics?

The Progress Dynamics® framework is a comprehensive, repository-based environment for Progress® 4GL developers who are building new applications, or who need to migrate existing applications to take advantage of new Progress 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 Progress® Application Development Model (ADM) and Progress SmartObjects™, which provide a basis for defining and combining standard components. Progress Dynamics extends the ADM and adds even more capabilities to the Progress 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 4GL client session or some other client type, without a local database connection. Business logic runs on one or many Progress AppServers™ 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.

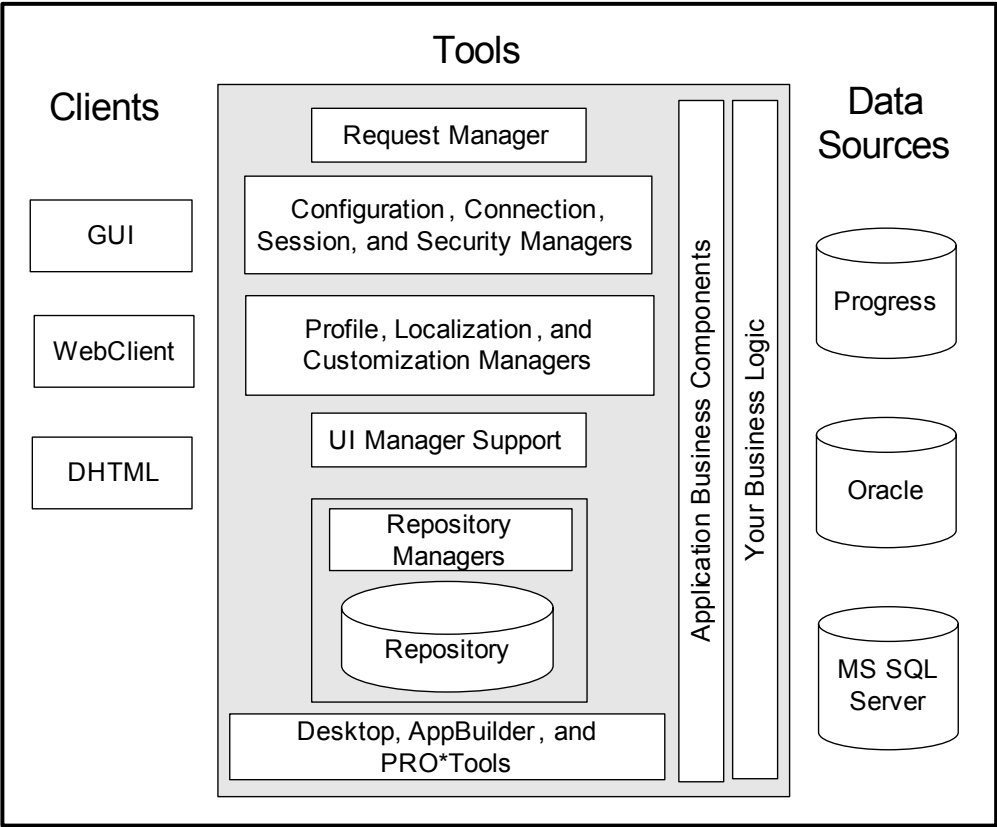


Figure 1–1: Progress Dynamics run-time architecture

This chapter describes the following components that make up Progress Dynamics:

- [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 Progress deployment models (for example, Progress 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.

1.1 Object repository

The heart of Progress Dynamics is its object repository. The Progress Dynamics Repository consists of a Progress database named `icfdb` whose schema is defined to store information about a Progress 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 a Progress application. It is the Repository, and the toolset which 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
- Customizable sequence and reference number generation
- Security data
- Localization data
- Online help integration data
- Multi-media data

1.2 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 4GL 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 Progress 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 Progress database schemas with Progress Dynamics-compliant standards and built-in referential integrity validation

1.3 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:

- **Progress SmartDataObject™** — Typically handles transactions on data in a single database table. SDOs can be either static or dynamic objects.
- **Progress SmartBusinessObject™** — 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 Progress 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.
- **Progress SmartDataViewer™** — Visualizes and accepts user changes of data in a single record or view. SDVs can be either static or dynamic objects.
- **Progress SmartDataBrowser™** — Visualizes and accepts user changes of data in multiple records or views. SDBs can be either static or dynamic objects.

1.4 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, and JMS.
- 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 need to install and configure the Progress OpenEdge™ Studio including Progress Dynamics. See the [Progress Dynamics Installation Guide](#) for complete instructions.

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](#)

2.1 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

Directory	Contents
\Tutorial	Main directory for tutorial files
\Tutorial\modules	Tutorial product module objects
\Tutorial\databases	Tutorial databases
\Tutorial\databases\icfdb	Tutorial Repository database
\Tutorial\databases\dynsports	Tutorial application database

2.2 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.

Copy `<install-dir>\gui\icf\icfconfig.xml` to `<wrk>\Tutorial`.

Later in the tutorial, you create a customized version of the `icfconfig.xml` file for a session type that runs the sample application. For now, the standard configuration file contains the necessary information to start a development session for the tutorial.

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.

Copy `<install-dir>\bin\icf.ini` to `<wrk>\Tutorial`.

icf.pf

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

Copy `<install-dir>\icf.pf` to `<wrk>\Tutorial`.

2.3 Creating startup scripts and shortcuts

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

2.3.1 Copying the shortcuts to a Desktop folder

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

To create the Desktop folder:

- 1 ♦ Right-click on your Windows desktop and choose **New→Folder**.
- 2 ♦ Enter **Progress Dynamics Tutorial** as the folder's name.
- 3 ♦ Double-click the folder to open it.
- 4 ♦ Choose **Start→Programs→Progress Dynamics**.

NOTE: This procedure assumes you chose to install the shortcuts for your Progress Dynamics environment 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**

2.3.2 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.

Copy `startdbms.bat` and `stopdbms.bat` from your `<install-dir>\bin` directory to your `<wrk>\Tutorial` directory, where `<install-dir>` is your Progress Dynamics install directory and `<wrk>` is your Progress Dynamics working directory.

2.3.3 Modifying the shortcuts

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

Modifying the development startup shortcut

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 Progress Version 9.1D 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 ♦ Choose **OK**.

Modifying the database server startup shortcut

To modify the 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 “<wrk>\Tutorial\startdb.bat” in the **Target** field.
- 4 ♦ Type “<wrk>\Tutorial” in the **Start in** field.
- 5 ♦ Choose **OK**.

Modifying the database server shutdown shortcut

To modify the 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 “<wrk>\Tutorial\stopdb.bat” in the **Target** field.
- 4 ♦ Type “<wrk>\Tutorial” in the **Start in** field.
- 5 ♦ Choose **OK**.

2.3.4 Modifying the database scripts

The database scripts that you copied need to 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\dynsports\dynsports"  
-S dynsports
```

- 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.

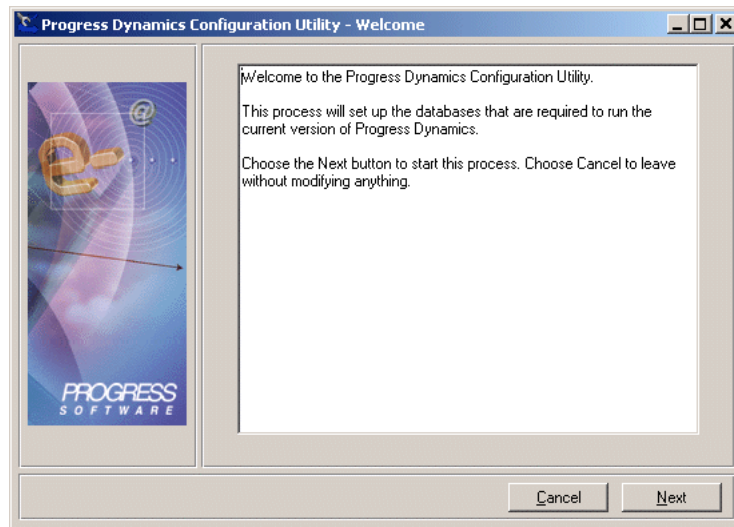
2.4 Creating a tutorial Repository

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.

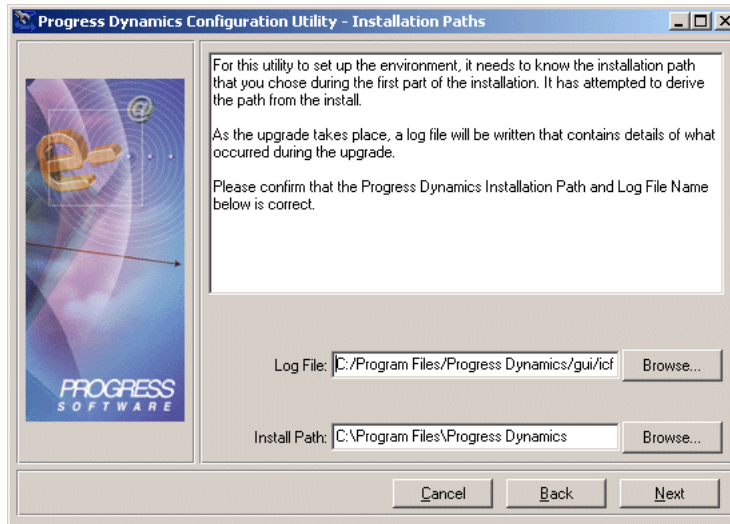
Setting up a functional Repository requires that many steps be performed in the correct order. Because of this complexity, you can only create a Repository using the Progress Dynamics Configuration Utility (DCU).

To create a Repository to use with the tutorial:

- 1 ♦ Choose **Start**→**Programs**→**Progress Dynamics**→**Dynamics Configuration Utility** to launch the DCU:



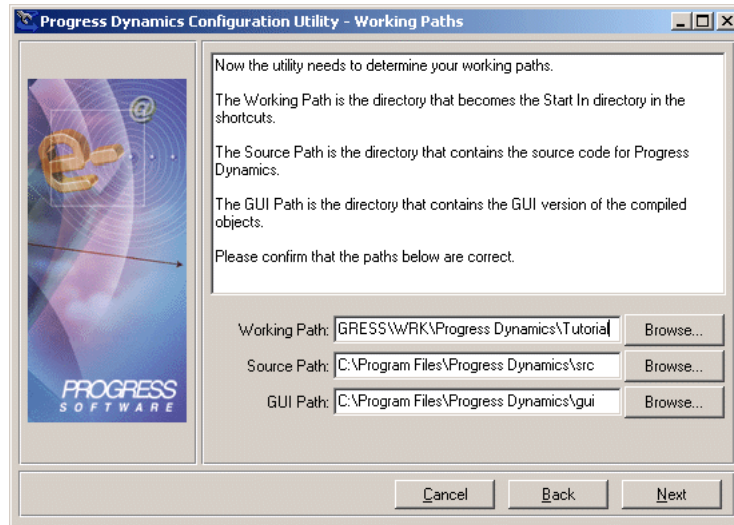
- 2 ♦ Choose **Next**. The **Installation Paths** page appears:



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.

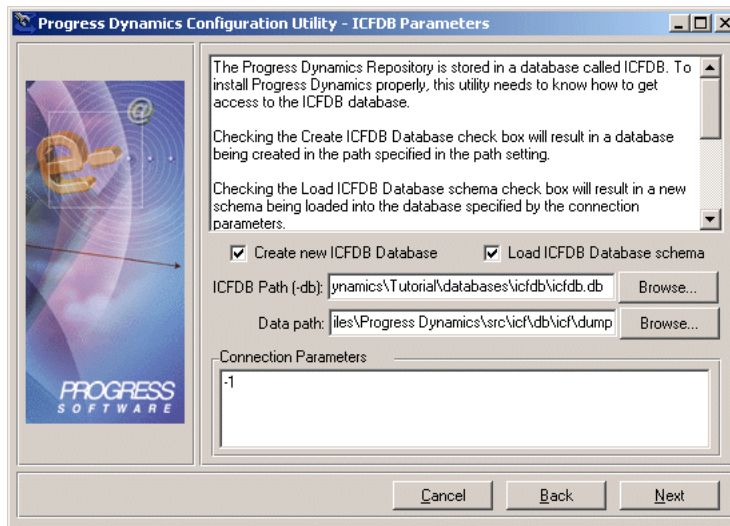
- 3 ♦ Choose **Next**. The **Working Paths** page appears. The DCU attempts to derive the working paths automatically. If it does not derive the correct paths, you can use the Browse button to set the correct paths.

- 4 ♦ Set the **Working Path** field to `<wrk>\Tutorial`, where `<wrk>` is your Progress Dynamics working directory, using the **Browse** button:



- 5 ♦ Choose **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\icf\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:

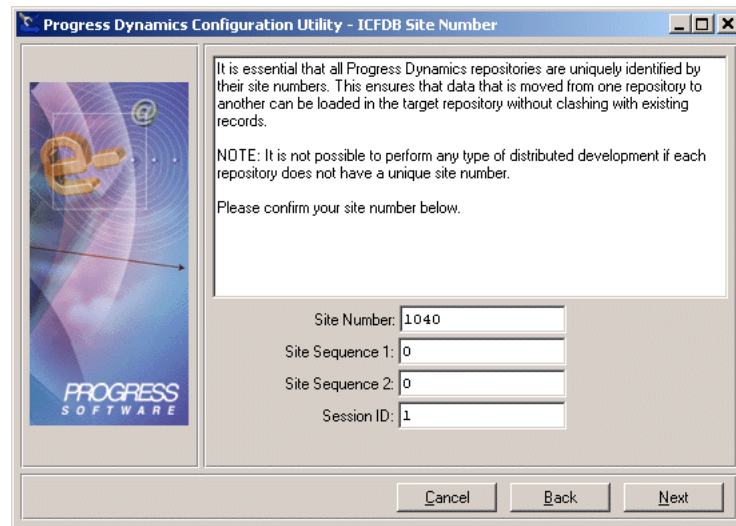


- 9 ♦ Choose **Next**. The **ICFDB Site Number** page appears.

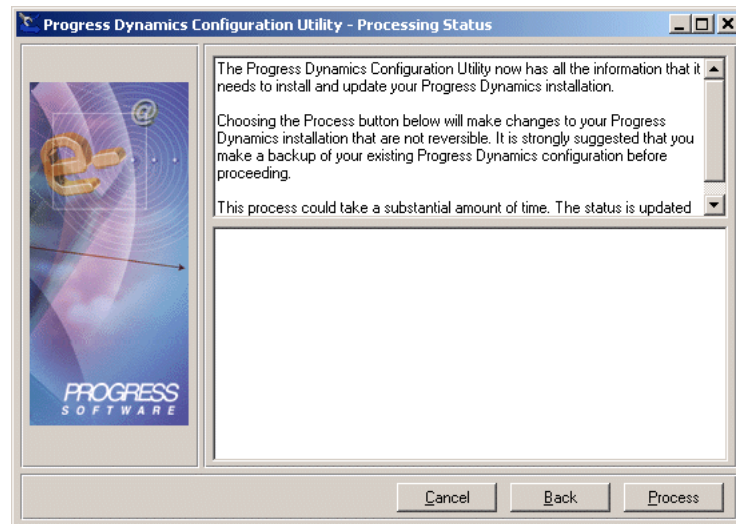
Every Progress Dynamics Repository database should have a unique site number assigned to it. The site number is combined with a database sequence value to form the key value for every record in the Repository database. The unique site number of each Repository database enables dynamic object definitions 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 *Getting Started with Progress Dynamics* tutorial users.

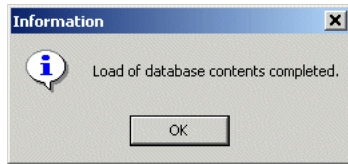
- 10 ♦ Type **1040** for the **Site Number**. You can accept the default values for the other fields:



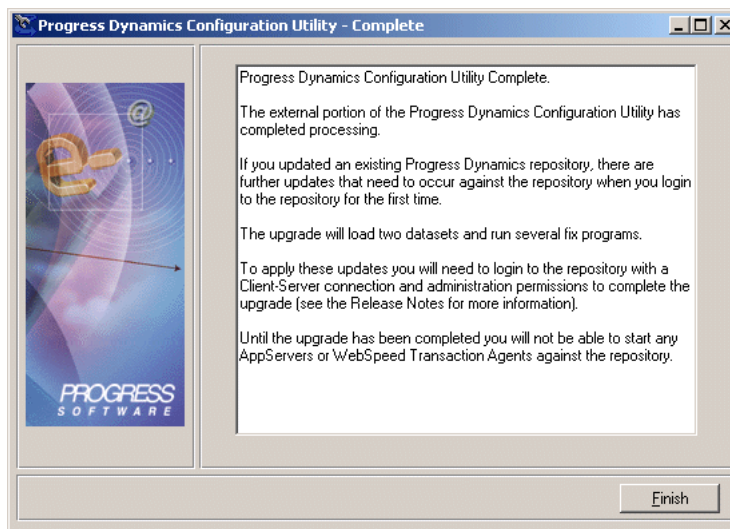
- 11 ♦ Choose **Next**. The **Processing Status** page appears:



- 12 ♦ Choose **Process** to start the creation of the tutorial Repository. The following message appears when the process is complete:



- 13 ♦ Choose **OK** to clear the message dialog box.
- 14 ♦ Choose **Finish** on the **Complete** page to close the DCU:



Your tutorial Repository is now ready to use.

2.5 Creating the application database

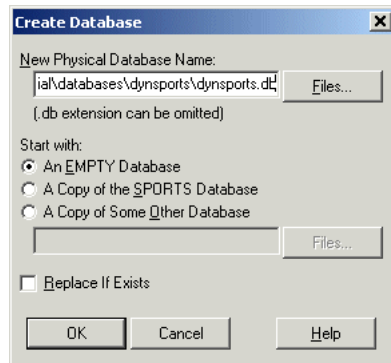
The next step in setting up your development environment is to create the application database. During the tutorial, you define application screens and menus for the DynSports database. This database was designed using the ERwin data modeling tool. Progress Dynamics comes with an ERwin template for creating 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 Progress Version 9.1E installation.

2.5.1 Creating an empty database

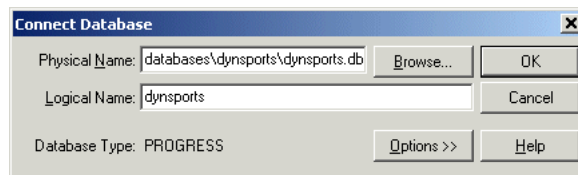
To create a DynSports database to use with the tutorial:

- 1 ♦ Choose **Start→Programs→Progress→Data Administration** to launch the Data Administration tool.
- 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:



- 4 ♦ Choose **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 **dynsports**. 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:



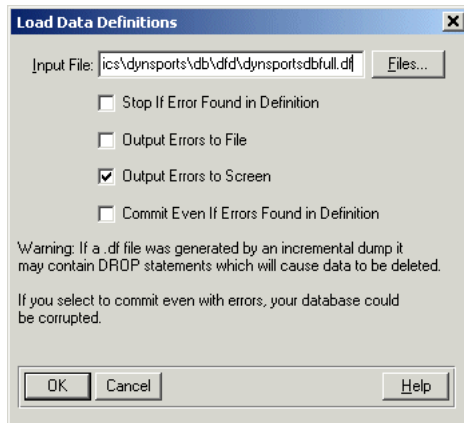
- 5 ♦ Choose **OK** to connect the database.

2.5.2 Loading the database schema

Next, you need to load the data definitions for the application database.

To load the DynSports database schema:

- 1 ♦ Choose **Admin**→**Load Data and Definitions**→**Data Definitions (.df)**. The Load Data Definitions window appears.
- 2 ♦ Select **<install-dir>\Tutorial\dynamics\dynsports\db\dfd\dynsportsdbfull.df**, where **<install-dir>** is your Progress Dynamics install directory, as the **Input File**. This file contains the complete schema definitions for the dynsports database:



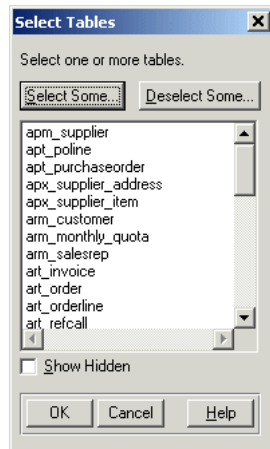
- 3 ♦ Leave the other settings as they are and choose **OK**.
- 4 ♦ Choose **OK** to clear the load completion message.

2.5.3 Loading the data

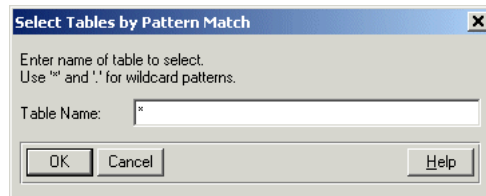
Next, you need to load the table contents for the application database.

To load the DynSports table contents:

- 1 ♦ Choose **Admin**→**Load Data and Definitions**→**Table Contents (.d)**. The Select Tables window appears:

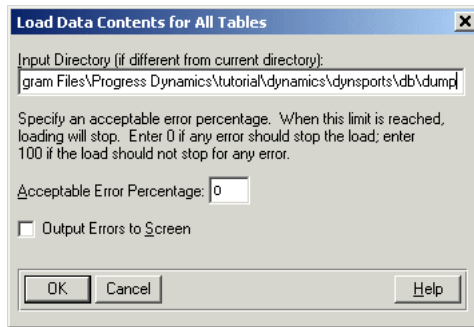


- 2 ♦ Choose the **Select Some** button:



- 3 ♦ Select all the tables using the “*” wild card, and choose **OK**.
- 4 ♦ Choose **OK**. The Load Data Contents for all Tables dialog box appears.

- 5 ♦ Type `<install-dir>\Tutorial\dynamics\dynsports\db\dump` as the **Input Directory**, where `<install-dir>` is your Progress Dynamics install directory. This directory contains the .d files that contain the table data:



- 6 ♦ Leave the other settings as they are and choose **OK**. The load process can take some time.
- 7 ♦ Choose **OK** to clear the load completion message.
- 8 ♦ 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`.

2.6 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 need to 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 Progress Dynamics 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 need to 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.

2.7 Setting your PROPATH

Progress sessions use a special path to search for files. The path can be stored in 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 Progress application development environment. You need to start your tutorial session to access the PRO*Tools.

2.7.1 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.

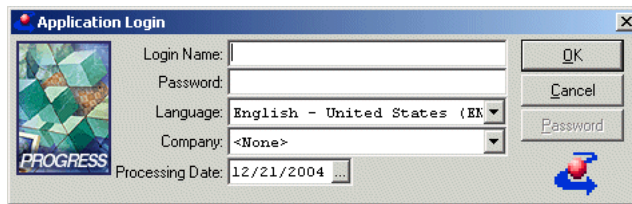
You can login to the framework and an application using a particular language. You can translate both the development tools and your finished application into any number of languages. You add some translations to the sample application and create a new user who uses the translations in [Chapter 5, “Customizing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

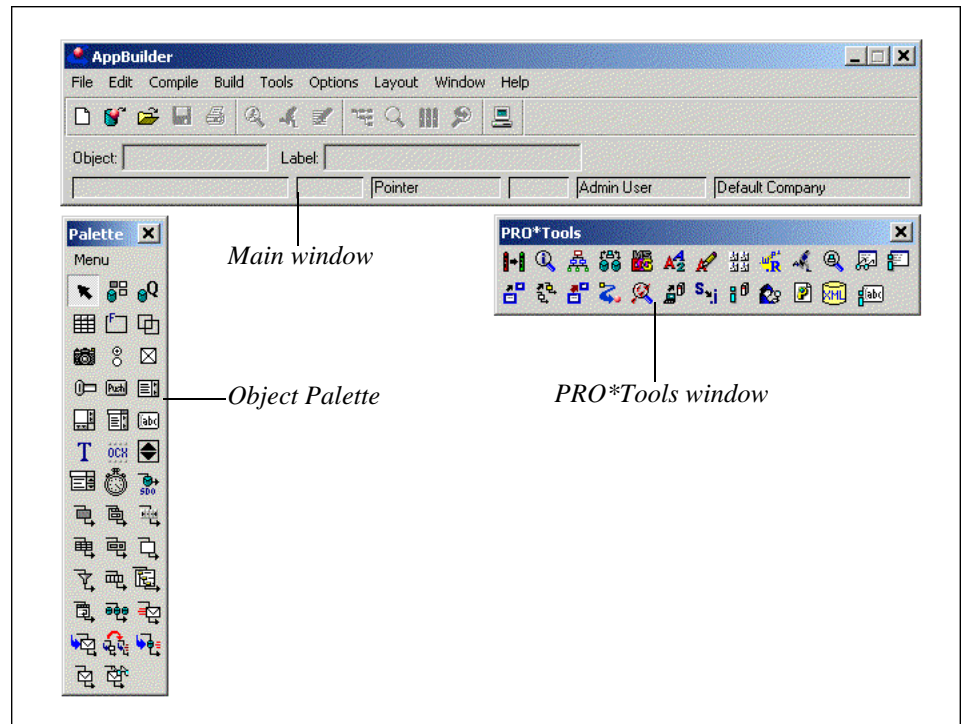
Progress Dynamics runs in different modes depending on the application configuration and context. In this tutorial, you use the framework in development mode. Development mode provides the correct tools and options to develop the sample application.

To start your tutorial environment:

- 1 ♦ Open the **Progress Dynamics Tutorial** folder on 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 choose **OK**. The AppBuilder windows appear:

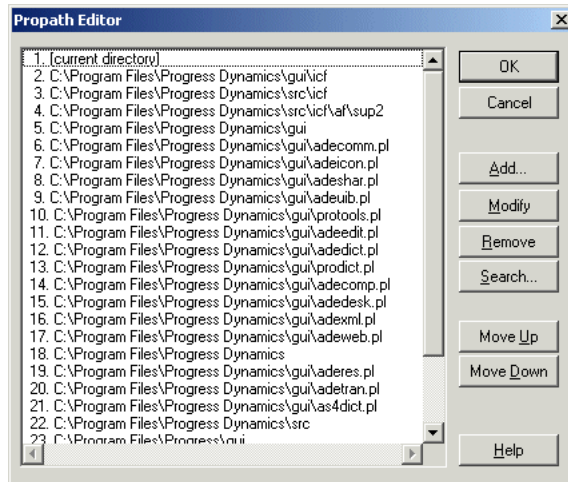


2.7.2 Using the Propath Editor

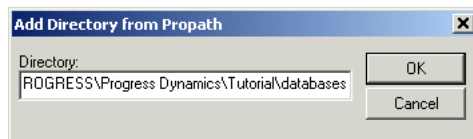
The PROPATH for your Progress Dynamics session is defined in your `icf.ini` file. Because you copied the file from your installed version, almost all the directories you need for the tutorial are already listed in the PROPATH. You only need to add the locations of the DynSports database triggers and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

To add the directories to the tutorial PROPATH:

- 1 ♦ Choose the **PROPATH** icon  on the PRO*Tools window. The **Propath Editor** appears:

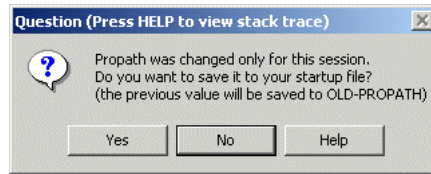


- 2 ♦ Choose **Add**. The **Add Directory from Propath** dialog appears.
- 3 ♦ Type **<wrk>\Tutorial\databases** in the **Directory** field, where **<wrk>** is your Progress Dynamics working directory:



- 4 ♦ Choose **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 ♦ Choose **OK** to close the Propath Editor. A dialog box appears asking if you want to save the changes in your startup file:



- 8 ♦ Choose **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.

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 the [Progress Dynamics Developer's Guide](#) and the [Progress Dynamics Programming Handbook](#).

This chapter covers the following topics:

- [Logging in to the framework tools](#)
- [Using the AppBuilder with Progress Dynamics](#)
- [Connecting to the application database](#)
- [Creating a new product and modules](#)
- [Importing tables as entities into the Repository](#)
- [Generating application objects](#)
- [Viewing your application objects](#)

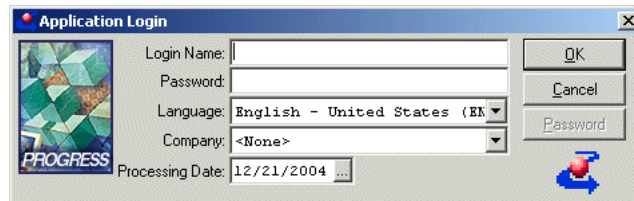
3.1 Logging in to the framework tools

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 on 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 choose **OK**. The AppBuilder windows appear.

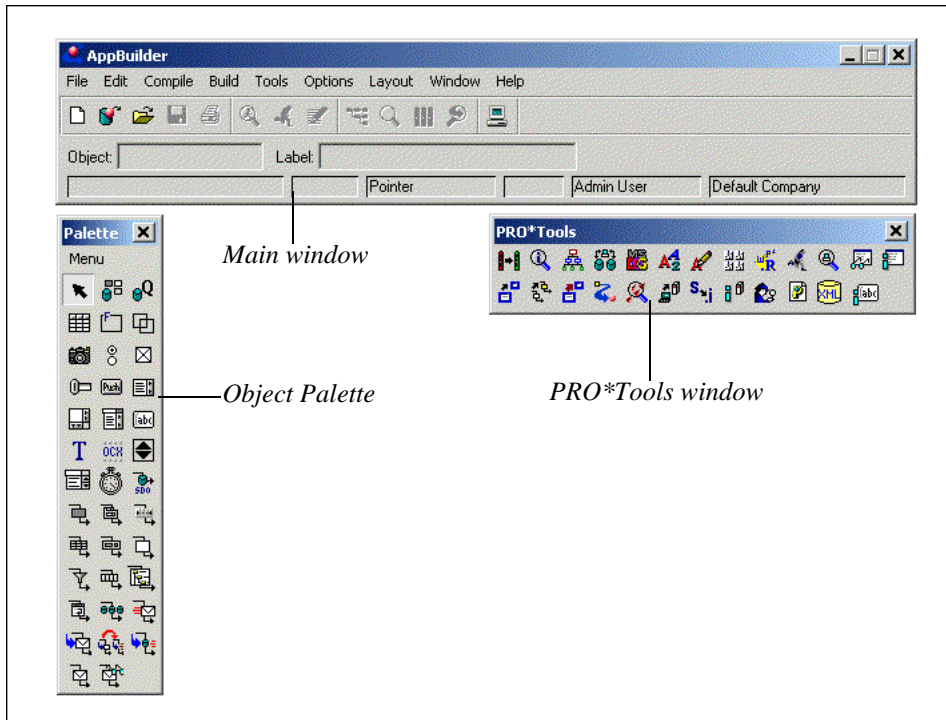
3.2 Using the AppBuilder with Progress Dynamics

The Progress Dynamics AppBuilder is an extended version of the Progress AppBuilder, with additional menus and tools. The AppBuilder is the primary visual programming tool in the Progress Application Development Environment (ADE). In Progress Dynamics, you create and assemble application components in the AppBuilder. The AppBuilder’s Section Editor makes it easier to write Progress 4GL 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 Progress AppBuilder, it consists of the three windows shown in the following figure: 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.

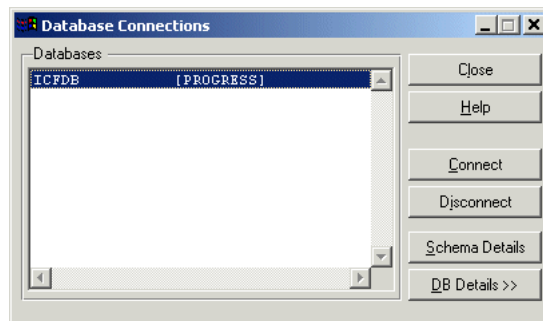
3.3 Connecting to the application database

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.

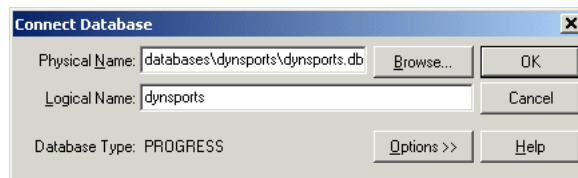
You must connect to the DynSports database before starting to work on the tutorial. In [Chapter 5, “Customizing the Application,”](#) you build a session type into the Configuration Manager to automatically make the connection when the finished application starts. The standard ICFDEV session type already 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:

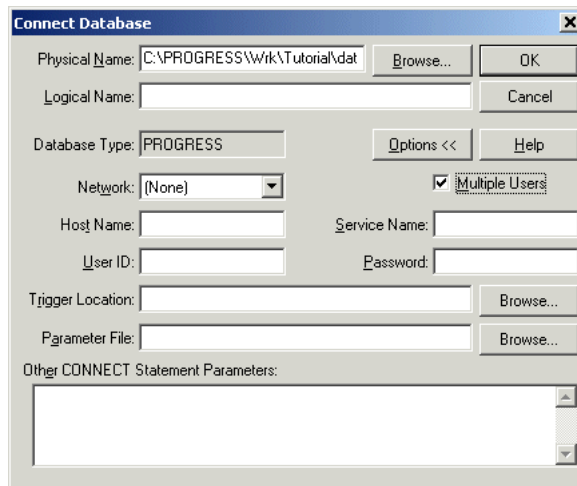
- 1 ♦ From the AppBuilder main menu, select **Tools→Database Connections**. The Database Connections dialog box appears:



- 2 ♦ Choose the **Connect** button.
- 3 ♦ Browse for the local DynSports database:



- 4 ♦ Choose the **Options>>** button and select the **Multiple Users** toggle box:



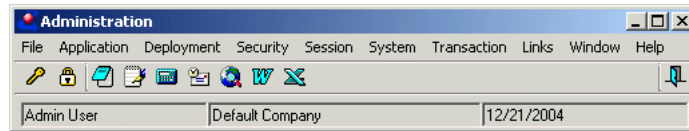
- 5 ♦ Choose **OK** and then close the Database Connections dialog box.

3.4 Creating a new product and modules

To help organize the objects in your application, particularly for deployment, Progress Dynamics supports a hierarchy of what are called *Products* and *Product Modules*. Using this hierarchy helps you keep track of and allows you to more easily locate all of your components. You can use them to describe any useful type of organization for your application. You must assign all dynamic application objects to a Module. For more information on using product and product modules, see the chapter on preparing to build application objects in the [Progress Dynamics Developer's Guide](#).

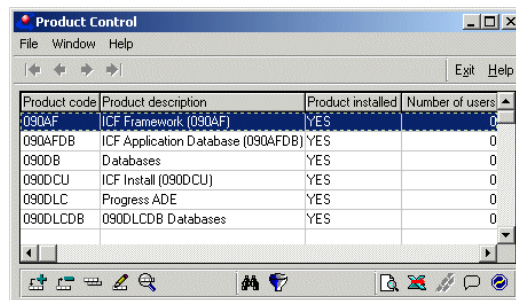
To define a DynSports Product and several Product Modules under it:

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

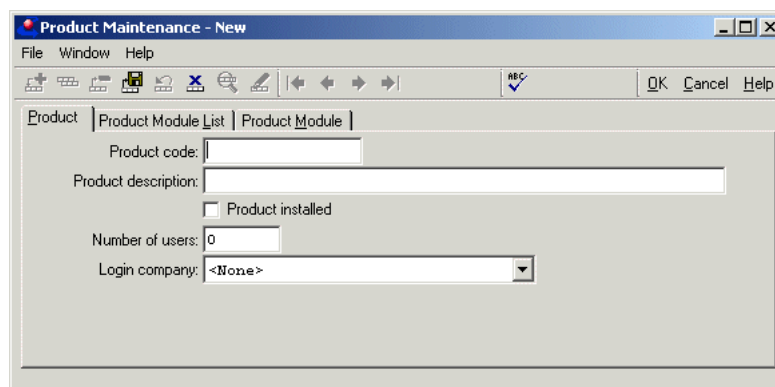


The Administration window provides access to tools for configuring and maintaining your application development environment. The tools include functions such as application structure, deployment, security, localization, and entity control. You will use some of these later in the tutorial.

- 1 ♦ In the Administration window, choose **Application→Product Control**. The Product Control window appears:




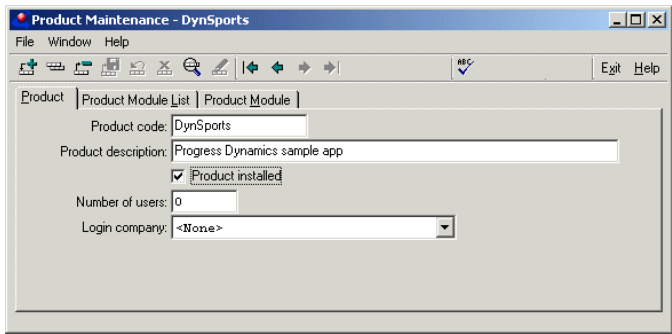
- 2 ♦ Choose the **Add** button . The Product Maintenance dialog box appears:




- 3 ♦ Set the values shown in the following table in the **Product Maintenance** dialog box, and leave the default settings in the other fields:

Field	Value
Product code	DynSports
Production description	Progress Dynamics sample app
Product installed	Selected

- 4 ♦ Choose the **Save** button  in the toolbar. The dialog box should look like the following:

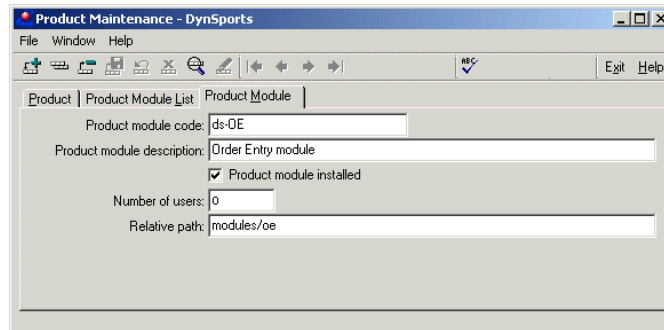


- 5 ♦ Choose the **Product Module** tab, and choose the **Add** button  above the folder.
- 6 ♦ Set the values shown in the following table on the tab:

Field	Value
Product module code	ds-OE
Production module description	Order Entry module
Product module installed	Selected
Number of users	0
Relative path	modules\oe

The Relative Path directory is the subdirectory under your working directory where the framework saves physical objects such as static viewers, browses, and SDOs.

- 7 ♦ Choose the **Save** button. The dialog box should look like the following:



- 8 ♦ Follow the procedure in [Step 5](#) through [Step 7](#) to add the Product Modules shown in the following table to the DynSports product:

Product module code	Description	Relative path
ds-EMP	Employee and Department information	modules\emp
ds-Entity	DynSports entity and datafield info	modules\entity
ds-General	Objects used across the whole app	modules\general

Select the **Product module installed** toggle box for all the modules.

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

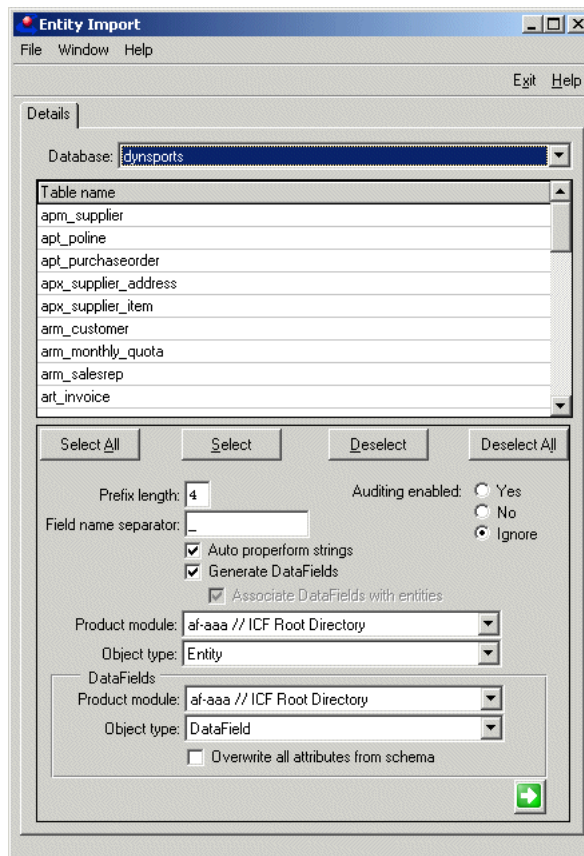
3.5 Importing tables as entities into the Repository

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:



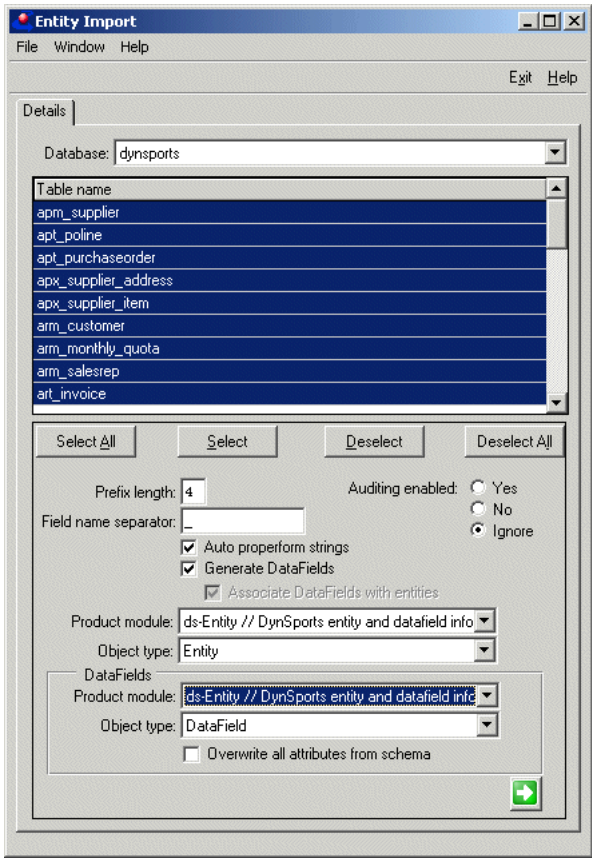
Note that the tables in the DynSports database start with a three-letter acronym (TLA) followed by an underscore. The first two letters identify the organizational structure, such as “AP” for the accounts payable module and “HR” for the human resources module. The third letter indicates the relative 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 the [Progress Dynamics Developer’s Guide](#).

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

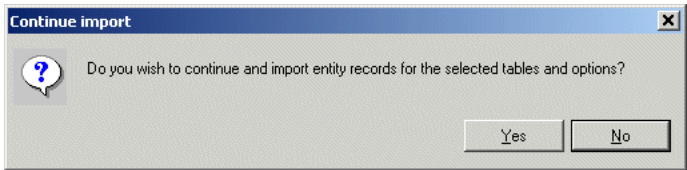
NOTE: For this application, you are only using a few tables in the DynSports database that relate to customer maintenance and order entry. The other tables are not used in the tutorial, but you might want them later to experiment or follow examples in the programming guides.

- 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.

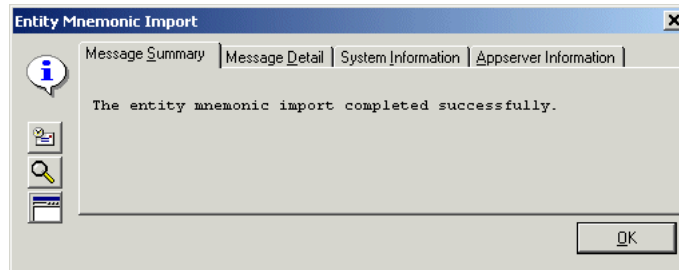
- 7 ♦ Leave all the other settings at their defaults. The dialog box should look like the following:



- 8 ♦ Choose the **Import** button . The Continue Import dialog box appears:






- 9 ♦ Choose **Yes**. The import process takes a while. After it completes, a message dialog box appears confirming successful completion:



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 tabs to display:

- Message summary
- Message detail
- System information
- AppServer information (if connected to a Progress AppServer)

The message dialog box also contains buttons that let you view a stack trace  of the 4GL procedure that generated the message, and send an email message  if your PC is correctly configured. The final button  toggles the dialog between full screen and normal modes.

- 10 ♦ Choose **OK**, and exit the Entity Import window.

3.6 Customizing the entity data

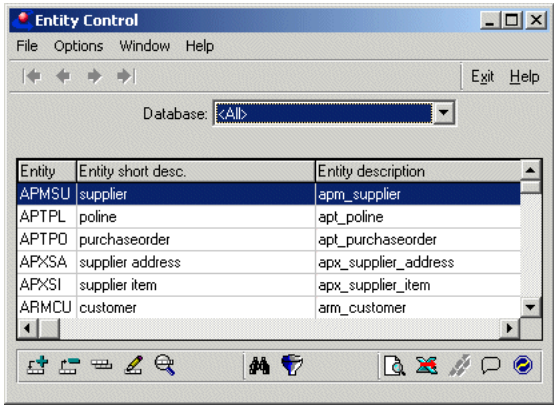
Once you have imported default data into your entity tables from the application database schema, you can customize that data as needed. You can change the fields, field formats, or the order in which the fields appear.

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.

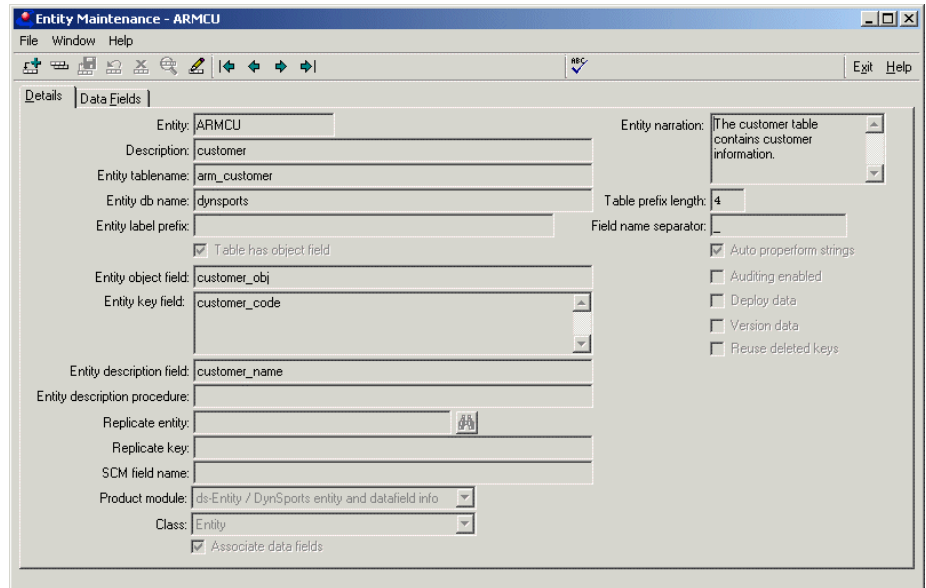
To customize the entity data:

- 1 ♦ Choose **System→Entity Control** from the **Administration** window. The Entity Control dialog box appears:




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

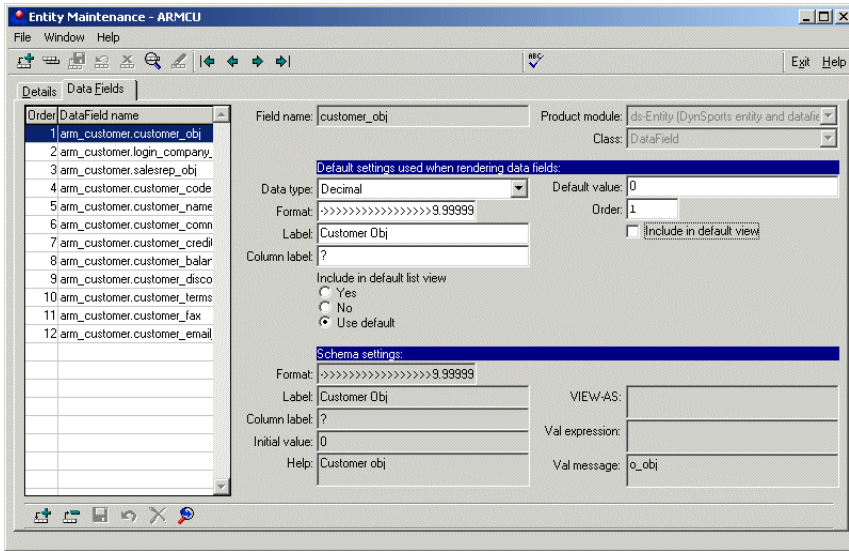
- 3 ♦ Double-click the **ARMCU** record. The Entity Maintenance dialog box appears:



The image shows the 'Entity Maintenance - ARMCU' dialog box. It has a menu bar (File, Window, Help) and a toolbar with various icons. The 'Details' tab is selected, showing fields for Entity (ARMCU), Description (customer), Entity tablename (arm_customer), Entity db name (dynsports), Entity label prefix, Entity object field (customer_obj), Entity key field (customer_code), Entity description field (customer_name), Entity description procedure, Replicate entity, Replicate key, SCM field name, Product module (ds-Entity / DynSports entity and datafield info), and Class (Entity). There are also checkboxes for 'Table has object field', 'Auto perform strings', 'Auditing enabled', 'Deploy data', 'Version data', 'Reuse deleted keys', and 'Associate data fields'. The 'Data Fields' tab is also visible, showing a list of fields with their respective data types and lengths.


- 4 ♦ Choose the **Modify record** button .
- 5 ♦ Select the **arm_customer.customer_obj** record on the **Data Fields** tab.
- 6 ♦ Deselect the **Include in default view** toggle box. The other object fields are foreign fields from other database tables. You will use them as the targets for SDFs. So, unlike the customer_obj field, you want them to appear in the default view.

- 7 ♦ Choose the **Save** button in the **object instance toolbar** at the bottom of the folder window. The window should look like the following:



- 8 ♦ Set the default labels for the other fields as follows:

Field instance	Label
arm_customer.login_company_obj	Login Company
arm_customer.salesrep_obj	Sales Rep
arm_customer.customer_code	Customer code
arm_customer.customer_name	Customer name
arm_customer.customer_comments	Comments
arm_customer.customer_credit_limit	Credit limit
arm_customer.customer_balance	Balance
arm_customer.customer_discount	Discount
arm_customer.customer_terms	Terms
arm_customer.customer_fax	Fax
arm_customer.customer_email_address	Email

Notice that you also have access to the Dynamic Properties sheet  for each field from the *Object Instance* toolbar of this page. The Object Instance toolbar is the toolbar at the bottom of the page. If there are properties that you always set, setting them here might be easier than doing it after you generate the objects.

Now, you need to make the same changes for the other entities needed in the tutorial.

To customize the remaining entities for the tutorial:

- 1 ♦ Select the **ARMSR** record in the **Entity Control**. The Entity Maintenance dialog box repopulates with the records for the new entity.
- 2 ♦ Select **arm_salesrep.salesrep_obj** on the **Data Fields** tab.
- 3 ♦ Deselect the **Include in default view** toggle box, and save your change.
- 4 ♦ Select the **ARTOL** record in the **Entity Control**. The Entity Maintenance dialog box repopulates with the records for the new entity.

- 5 ♦ Select **art_orderline.orderline_obj** on the **Data Fields** tab.
- 6 ♦ Deselect the **Include in default view** toggle box, and save the change.
- 7 ♦ Set the default labels for the other fields as follows:

Field instance	Label
art_orderline.status_obj	Status
art_orderline.item_obj	Item
art_orderline.order_obj	Order
art_orderline.orderline_line_number	Line number
art_orderline.orderline_price	Price
art_orderline.orderline_qty	Quantity
art_orderline.orderline_discount	Discount
art_orderline.orderline_extended_price	Extended price

- 8 ♦ Select the **ARTOR** record in the **Entity Control**. The Entity Maintenance dialog box repopulates with the records for the new entity.
- 9 ♦ Select **art_order.order_obj** on the **Data Fields** tab.
- 10 ♦ Deselect the **Include in default view** toggle box, and save the change.

- 11 ♦ Set the default labels for the other fields as follows:

Field instance	Label
art_order.status_obj	Status
art_order.customer_obj	Customer
art_order.salesrep_obj	Sales rep
art_order.warehouse_obj	Warehouse
art_order.order_code	Order code
art_order.order_date	Order date
art_order.order_promise_date	Promise date
art_order.order_ship_date	Ship date
art_order.order_carrier	Carrier
art_order.order_instructions	Instructions
art_order.order_po	PO
art_order.order_terms	Terms
art_order.order_creditcard	Credit card
art_order.order_incoming	Incoming

- 12 ♦ Close the Entity Maintenance and Control windows, then restart your Progress Dynamics session.

Before generating your application objects, you need to 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.

3.7 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:

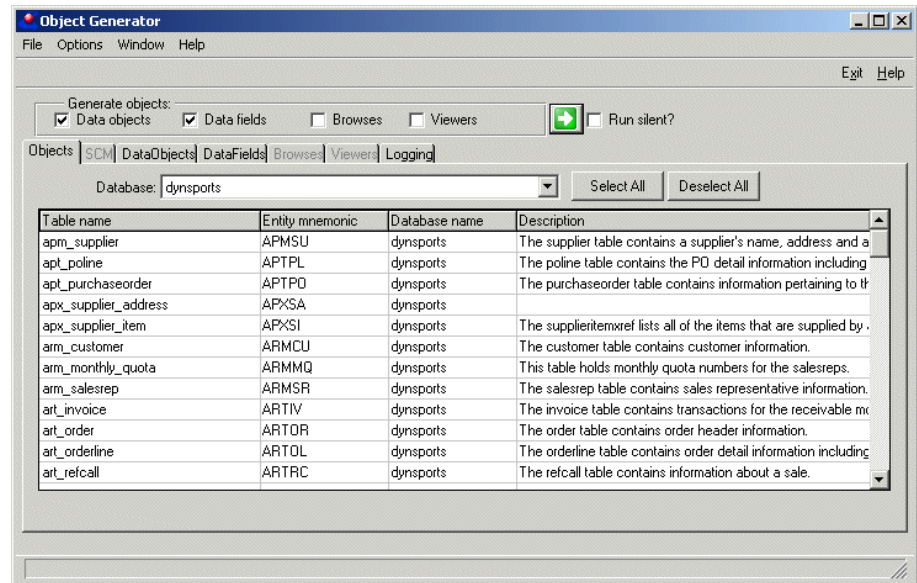
- **SmartDataObject (SDO)** — Moves data from the database on the server to the user session on the client where it can be displayed and updated.
- **Browse** — Lets you scroll through, select, and update records from an SDO's database query.
- **Viewer** — Lets you display and update details for a record in a Progress frame.

For more information on these objects, see the [Progress Dynamics Developer's Guide](#).

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 Browsers, and `viewv` for Viewers. You can modify this naming convention and other defaults 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:



- 2 ♦ Select the **DataObjects**, **Browses**, and **Viewers** toggle boxes in the **Generate Objects** section. 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 on the **Objects** tab.
- 4 ♦ Choose the **DataObjects** tab.

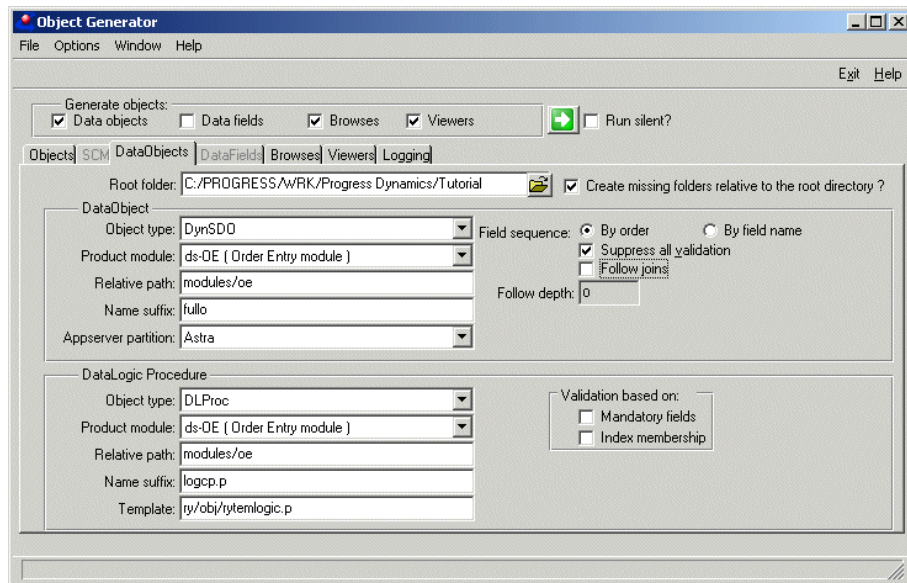
The information on this tab is used to create dynamic SDOs and data logic procedures (DLPs). The generated DLP has standard validation code for the entity. For your own applications, you should review this code and edit it to fit your needs. Generally, you put your business logic in the DLPs.

- 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.

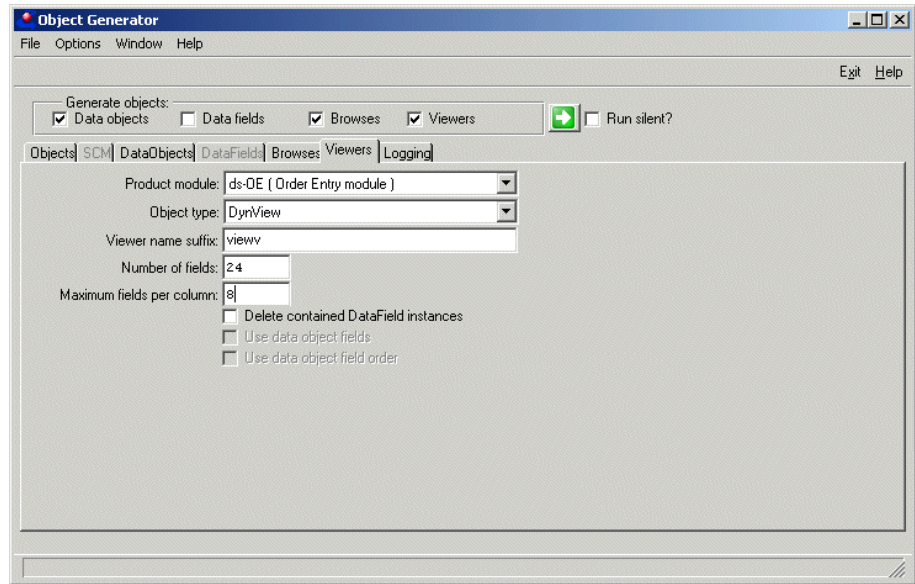
- 6 ♦ Select **ds-OE** from the **Product Module** combo box in the DataObject section. The **Relative Path** is automatically set to the directory you chose when you defined the Product Module. The fields in the DataLogic Procedure section also update automatically.
- 7 ♦ Select **Astra** from the **AppServer partition** combo box.
- 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 at their default settings, as shown:




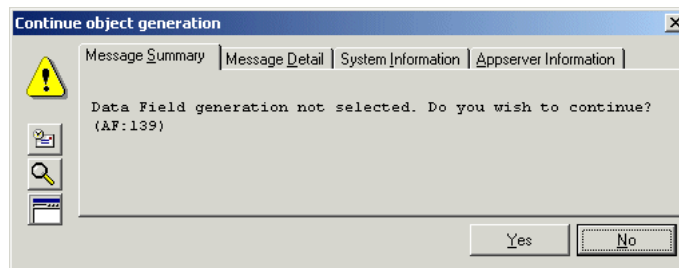
- 9 ♦ Choose the **Viewers** tab.

- 10 ♦ Change the **Number of Fields** to **24** and the **Maximum Fields per Column** to **8**, as shown:



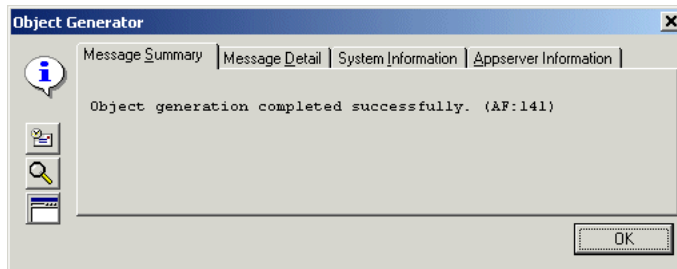
The default values are too large for the viewers you need in the tutorial. Viewers with a maximum of three columns of eight fields work better.

- 11 ♦ Choose the **Start** button  to begin generating your SDOs, Browses, and Viewers. A message dialog box appears warning you that datafields will not be generated, as shown:

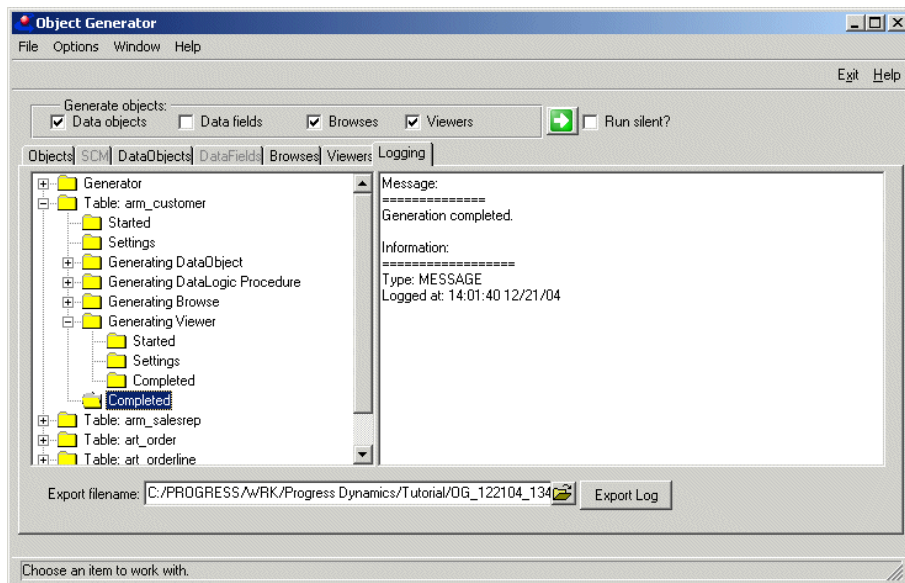


This is not a problem, since you generated data fields during the Entity Import phase.

- 12 ♦ Choose **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:



- 13 ♦ Choose **OK** to clear the message dialog box. You can export the Object Generator 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:



- 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.

3.8 Viewing your application objects


The Object generator created several kinds of static files and dynamic objects for you. The SDOs, Browsers, 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 `<wrk>\Tutorial\databases\dynsports\oe` and `<wrk>\Tutorial\databases\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.

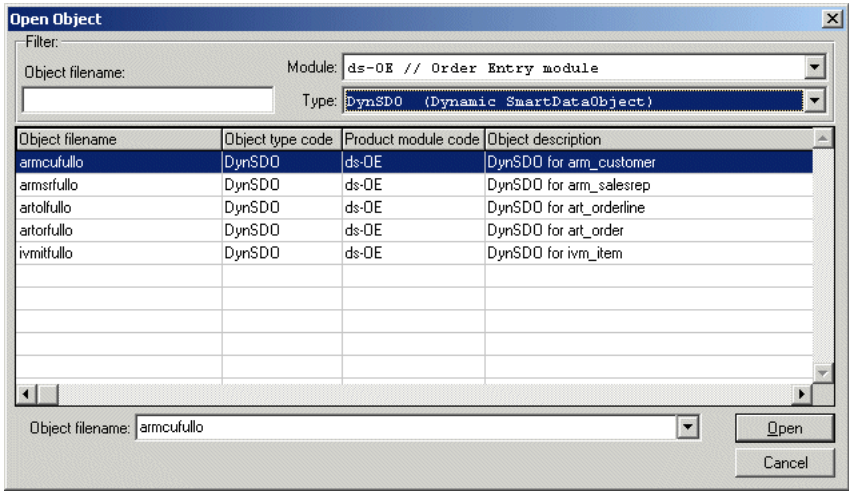
3.8.1 Viewing an SDO

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

To view the Customer SDO:

- 1 ♦ Choose the **Open Object** button  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 from combining the tables' entity names with the suffixes specified in the Object Generator, as shown:

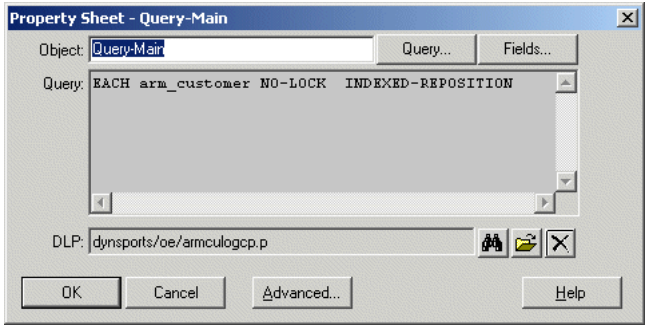


NOTE: If you type the first few letters of a filename in the **Object Filename** field, the browse filters on objects that begin with the letters you typed.

- 5 ♦ Double-click on its row in the browse or choose **Open**. A design window for the SDO appears:



- 6 ♦ Double-click inside the design window to look at the query the Object Generator created, 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, all fields are included in the list. Also by default, all fields, except the Progress Dynamics Object ID fields, are marked as updatable. 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.

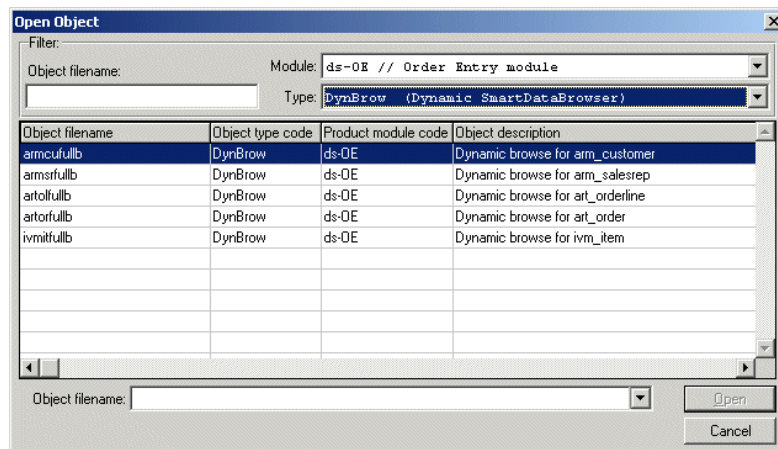
- 7 ♦ Choose **Cancel** and then close the SDO's design window.

3.8.2 Viewing a dynamic Browse

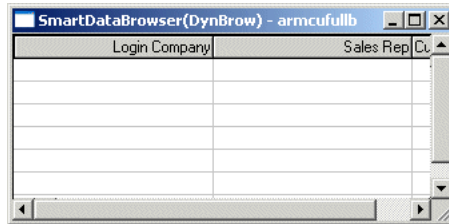
You can also open a design window for a dynamic Browse through the Open Object tool.

To access the design window:

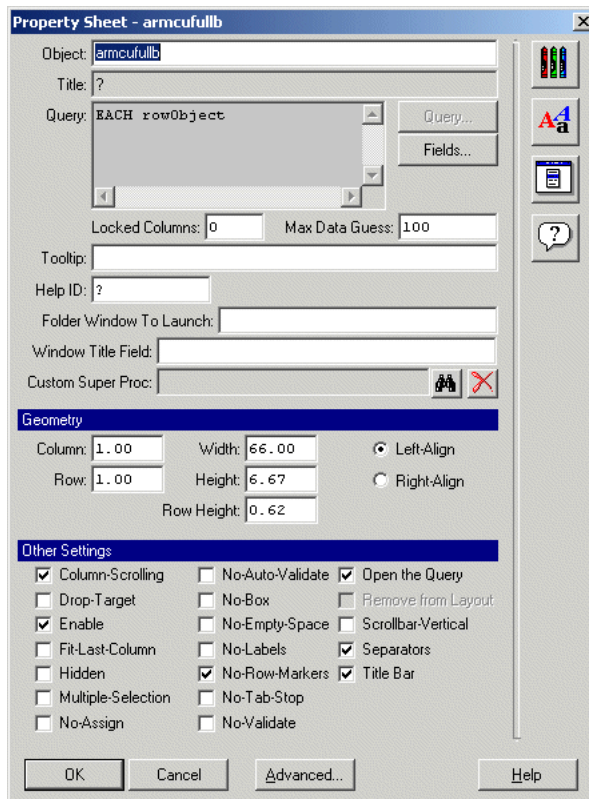
- 1 ♦ Choose the **Open Object** button , and select **DynBrowse (Dynamic SmartDataBrowser)** in the **Type** combo box. Note that the AppBuilder retained the Module filter on the ds-OE module. The filtering remains until you change it or exit the session:



- 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 non-visual object like a SDO, the design window for the dynamic Browse allows you to change its appearance:



- 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 a common set of code to create an object from the data in the Repository. If you changed 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 the 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.

Building the Sample Application

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 4GL 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 the *Progress Dynamics Developer's Guide* and the *Progress Dynamics Programming Handbook*.

As you work through this tutorial, you will be:

- [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.

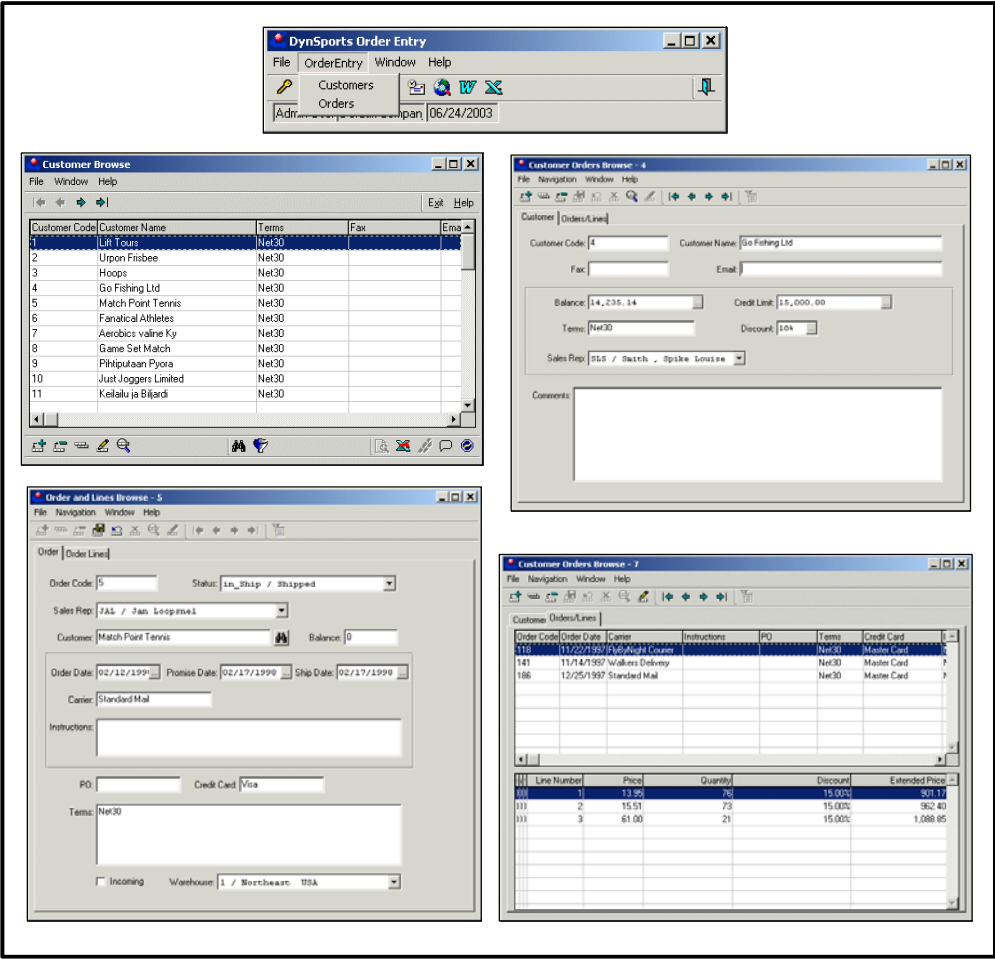


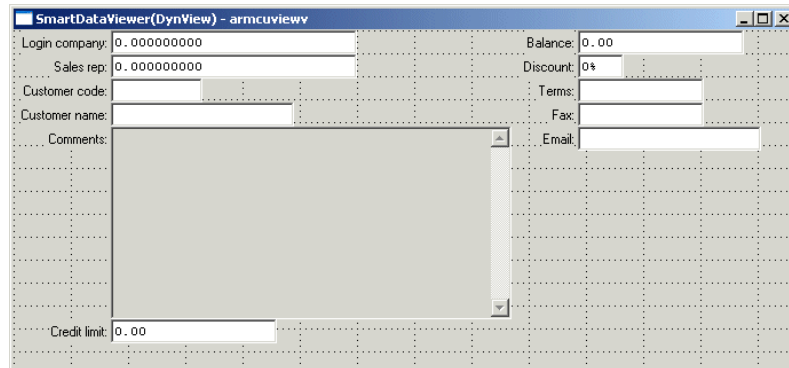
Figure 4-1: Completed sample application windows

4.1 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, **armcuvievw**, then choose **Open**. The SmartDataViewer design window appears:

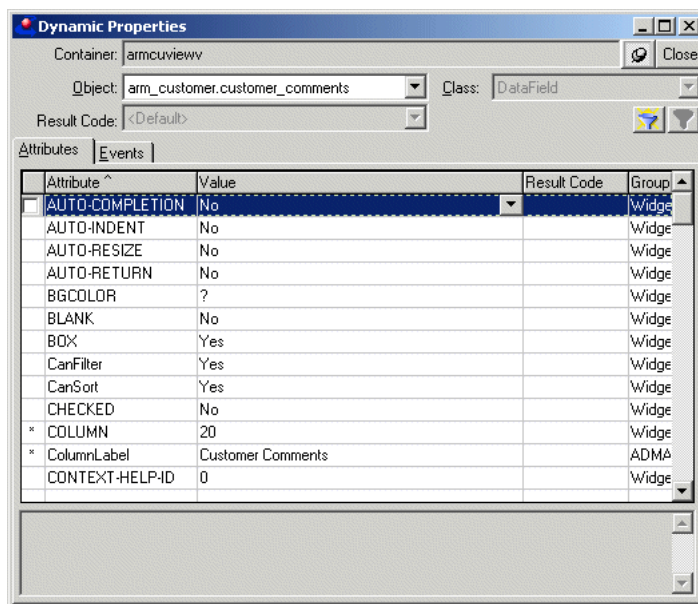


The layout of the viewer might differ slightly depending on your user preferences. The individual field labels are specified through the schema of the DynSports database. The schema also specified that the Comments field be displayed as an editor instead of a fill-in.

- 3 ♦ Select the **Comments** field and position it at the bottom of the viewer.

- 4 ♦ Choose **Window→Dynamic Properties** from the AppBuilder main window.

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

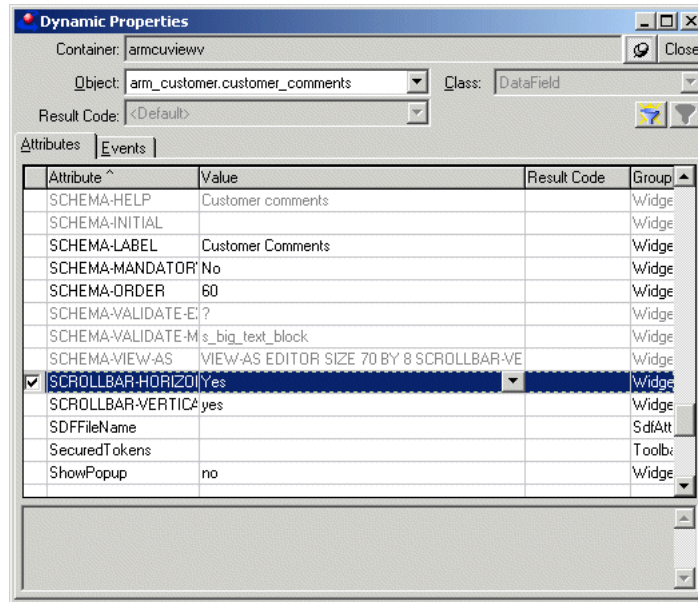


The fields at the top of the property sheet describe the object: the container in which the object exists, the object's full name, the object's class, and a list of any possible result codes defined in the system. Result codes are used to customize your application for different UI types, different user 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 updatable. The Events tab displays all allowable events for the current object. See the *Progress Dynamics Developer's Guide* for more information on using result codes, specific attributes, and events.

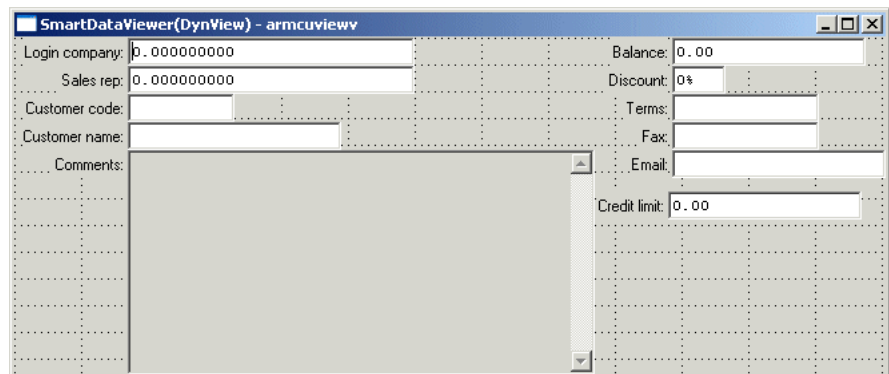
NOTE: User-defined attributes stored in the Repository are not supported in static object property sheets. You can modify user-defined attributes only through this property sheet and the Repository Object Maintenance window.

- 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 attribute's value has been changed from its default value.

- 7 ♦ Change the value of the **Scrollbar-Vertical** row to **Yes**, if necessary, as shown:




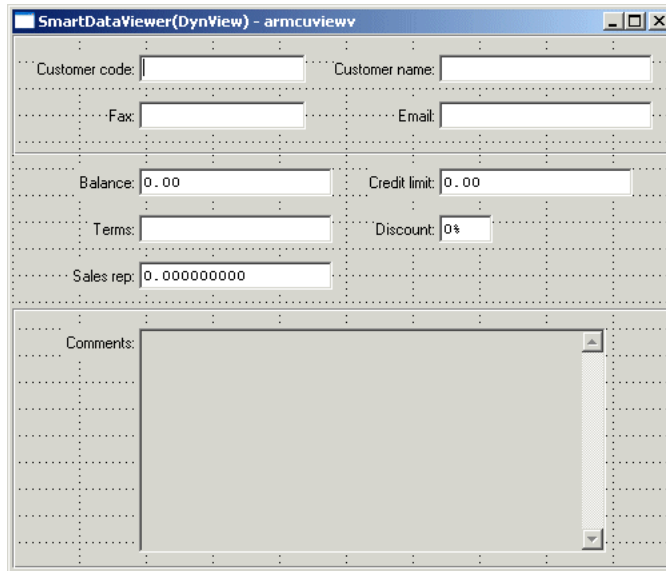
- 8 ♦ Close the properties sheet. Your design window should now look something like the following:



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**.
- 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 the following:



The screenshot shows a window titled "SmartDataViewer(DynView) - armcuvievv". The window contains a form with the following fields:

- Customer code:
- Customer name:
- Fax:
- Email:
- Balance:
- Credit limit:
- Terms:
- Discount:
- Sales rep:
- Comments:

- 11 ♦ Choose **Save** on the AppBuilder toolbar.



4.1.1 Adding a dynamic Combo to your viewer

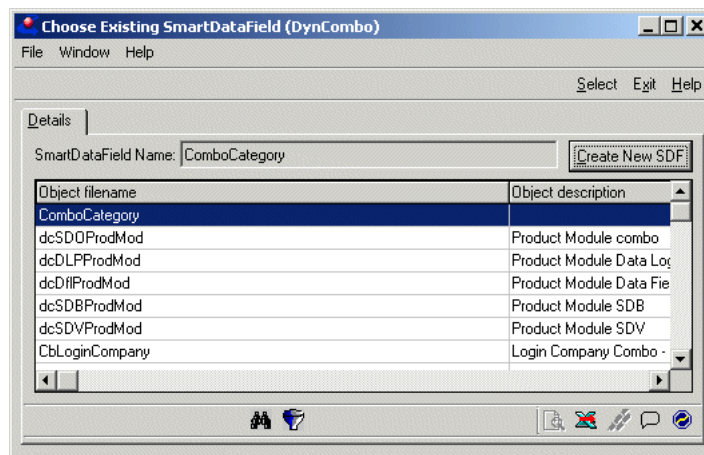
The most common addition to a viewer is a selection list for a *foreign key field*. A foreign key field is a field in one table that matches a unique key field in another table. An example in the DynSports database is the SalesRep field in the Customer table, which is a foreign key for the SalesRep key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

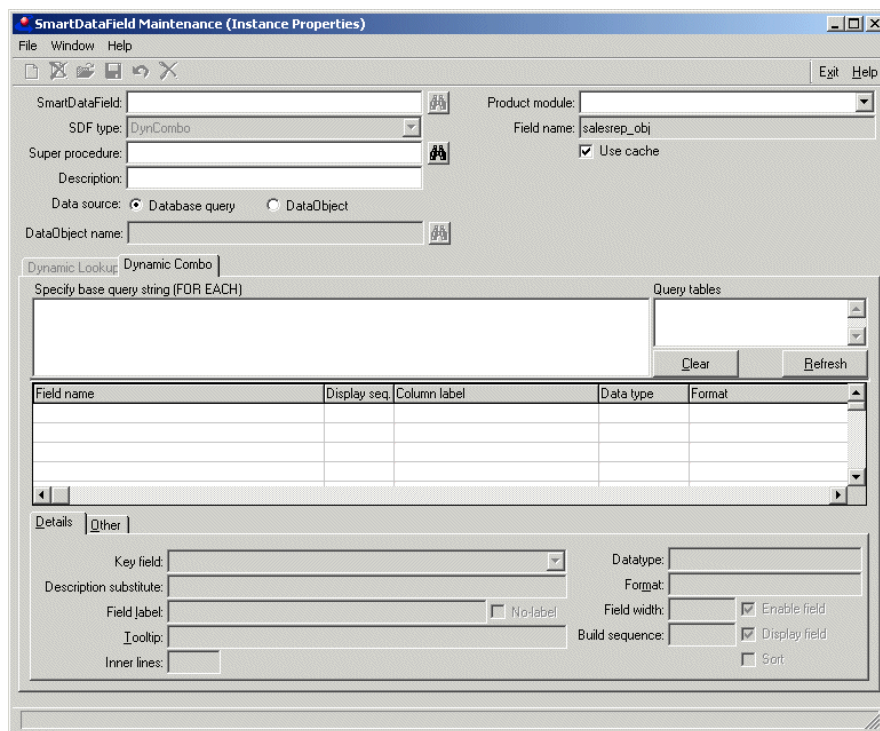
Part of the best practices recommendation for database design in Progress Dynamics is to include a unique key field on each table that is used only as a relationships target. These are the object (_obj) fields. Generally, you do not want the object field that uniquely identifies a table to appear in 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 on 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 ♦ Click on the **Sales Rep** field in the viewer to replace it with the dynamicCombo object. After the AppBuilder automatically sets up the proper connections, the Choose Existing SmartDataField dialog box appears:



- 4 ♦ Choose the **Create New SDF** button. The SmartDataField Maintenance window appears:



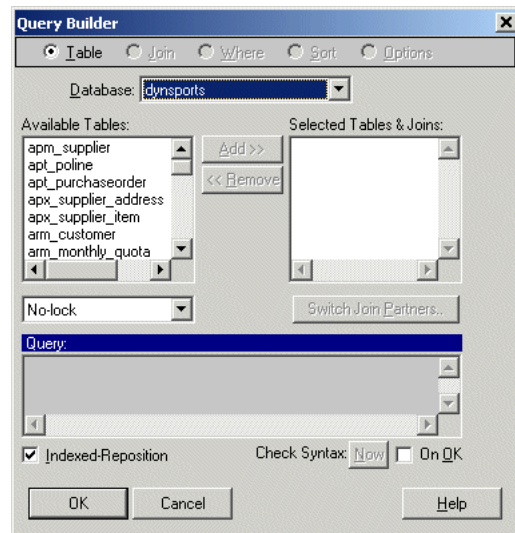
The image shows the 'SmartDataField Maintenance (Instance Properties)' window. It has a menu bar with 'File', 'Window', and 'Help'. Below the menu bar are icons for file operations and a 'Exit' button. The main area contains several fields and controls:

- SmartDataField:** A text field with a help icon.
- SDF type:** A dropdown menu currently set to 'DynCombo'.
- Super procedure:** A text field with a help icon.
- Description:** A text field.
- Data source:** Radio buttons for 'Database query' (selected) and 'DataObject'.
- DataObject name:** A text field with a help icon.
- Product module:** A dropdown menu.
- Field name:** A text field set to 'salesrep_obj'.
- Use cache:** A checked checkbox.
- Dynamic Lookup:** A tabbed interface with 'Dynamic Lookup' and 'Dynamic Combo' tabs. The 'Dynamic Combo' tab is active.
- Specify base query string (FOR EACH):** A large text area.
- Query tables:** A list box with 'Clear' and 'Refresh' buttons.
- Table:** A table with columns: Field name, Display seq, Column label, Data type, and Format. It is currently empty.
- Details / Other:** A tabbed interface with 'Details' and 'Other' tabs. The 'Details' tab is active.
- Details tab fields:**
 - Key field:** A dropdown menu.
 - Description substitute:** A text field.
 - Field label:** A text field with a 'No label' checkbox.
 - Tooltip:** A text field.
 - Inner lines:** A text field.
 - Datatype:** A text field.
 - Format:** A text field.
 - Field width:** A text field with an 'Enable field' checkbox.
 - Build sequence:** A text field with a 'Display field' checkbox.
 - Sort:** A checkbox.

- 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.

There 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 the [Progress Dynamics Developer's Guide](#).

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



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 ♦ Choose the **Add>>** button. The Query Builder creates a query for the table.
- 12 ♦ Choose **OK**. The SmartDataField Maintenance window refreshes with the details of the table.
- 13 ♦ Select **arm_salesrep.salesrep_obj** for the **Key field**.
- 14 ♦ 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.
- 15 ♦ Type **2** in the **Display seq** field for **arm_salesrep.salesrep_name** in the browse.

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

The screenshot shows a window titled "SmartDataViewer(DynView) - armcuvewv". The form inside has a grid-like layout with the following fields:

- Customer code: [text box]
- Customer name: [text box]
- Fax: [text box]
- Email: [text box]
- Balance: 0.00 [text box]
- Credit limit: 0.00 [text box]
- Terms: [text box]
- Discount: 0% [text box]
- Sales rep: [dropdown menu]
- Comments: [large text area]

- 19 ♦ Close the viewer design window.

4.1.2 Adding a dynamic Lookup to a viewer

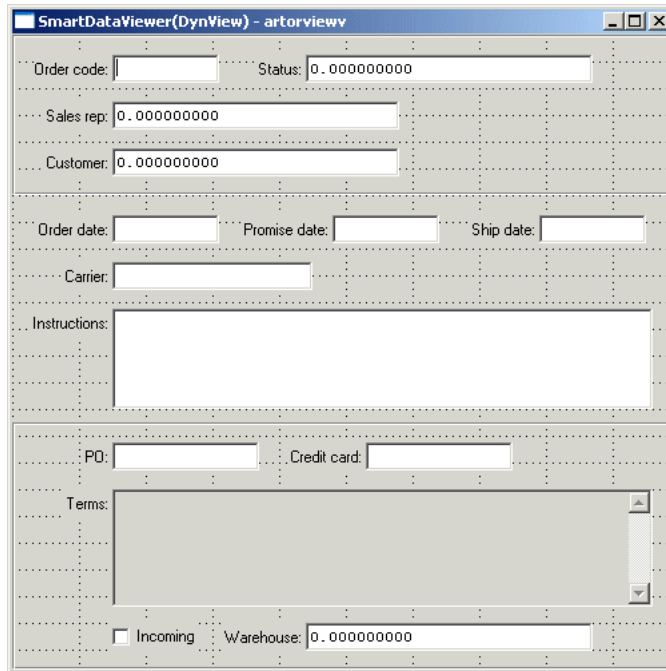
A dynamic Lookup can represent a larger set of records than you can display in a dynamic Combo. In this exercise, you add a dynamic Lookup object to the dynamic viewer for the Order table. The dynamic Lookup launches a separate dynamic browse of Customer records. Using the dynamic Lookup, you can assign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


Laying out the Order viewer

To add the dynamic Combo to the Order viewer:

- 1 ♦ Choose **File→Open Object** in the AppBuilder main window.
- 2 ♦ Select the dynamic Viewer, **artorvieww**, then choose **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:




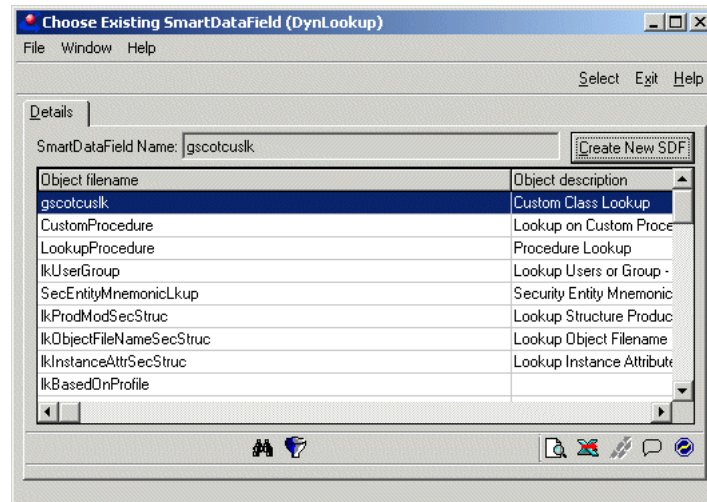
- 4 ♦ Right-click on the **SmartDataField** icon  on the **Object Palette** and select **DynamicCombo**.
- 5 ♦ Click on the **Sales Rep** field to replace it with the dynamicCombo object. The Choose Existing SmartDataField dialog box appears.
- 6 ♦ Double-click **SalesrepCombo** on the **Details** tab.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

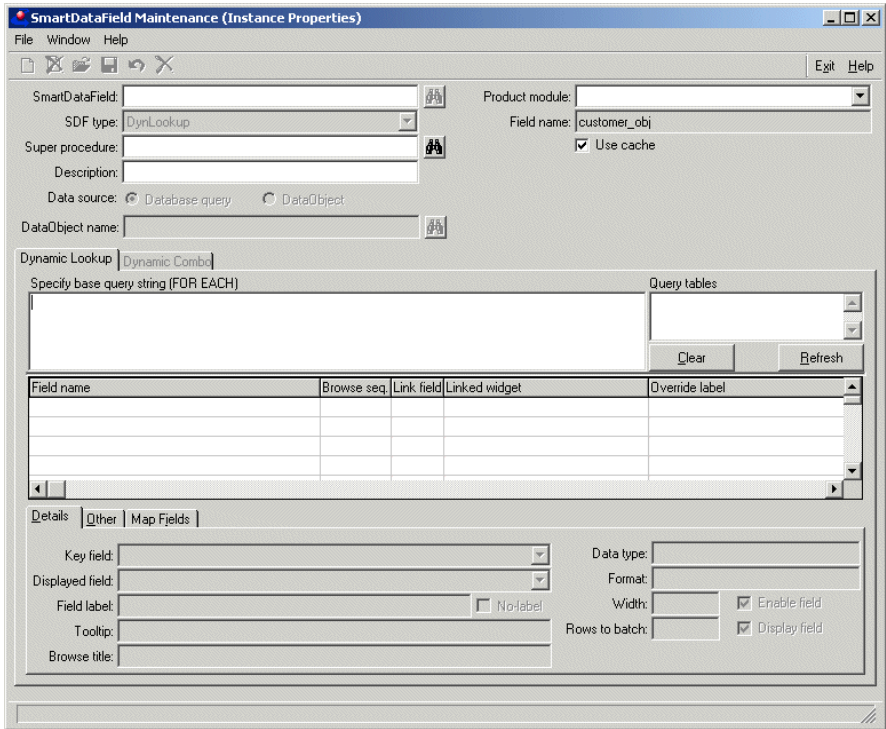
Adding a Customer dynamic Lookup

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 ♦ Click on 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 ♦ Choose the **Create New SDF** button. The SmartDataField Maintenance dialog box appears:



The image shows the 'SmartDataField Maintenance (Instance Properties)' dialog box. It has a menu bar (File, Window, Help) and a toolbar. The main area is divided into several sections:

- SmartDataField:** A text field with a dropdown arrow.
- SDF type:** A dropdown menu showing 'DynLookup'.
- Super procedure:** A text field.
- Description:** A text field.
- Data source:** Radio buttons for 'Database query' (selected) and 'DataObject'.
- DataObject name:** A text field.
- Product module:** A dropdown menu.
- Field name:** A text field showing 'customer_obj'.
- Use cache:** A checked checkbox.
- Dynamic Lookup:** A tabbed section with 'Dynamic Lookup' selected. It contains a 'Specify base query string (FOR EACH)' text area and a 'Query tables' list box with 'Clear' and 'Refresh' buttons.
- Table:** A table with columns: Field name, Browse seq, Link field, Linked widget, and Override label.
- Details:** A tabbed section with 'Details' selected. It contains fields for Key field, Displayed field, Field label, Tooltip, Browse title, Data type, Format, Width, Rows to batch, and checkboxes for 'Enable field' and 'Display field'.

- 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 ♦ Choose the **Refresh** button. The remaining area of the property sheet is filled in with default choices for the arm_customer table, as shown:

SmartDataField Maintenance (Instance Properties)

File Window Help

SmartDataField: CustomerCodeLookup Product module: ds-Entity / DynSports entity and datafield info

SDF type: DynLookup Field name: customer_obj

Super procedure: Description: Customer code dynamic lookup

Data source: ☒ Database query ☐ DataObject

DataObject name:

Dynamic Lookup Dynamic Combo

Specify base query string (FOR EACH)

FOR EACH arm_customer NO-LOCK INDEXED-REPOSITION

Query tables

arm_customer

Clear Refresh

Field name	Browse seq	Link field	Linked widget	Override label
arm_customer.customer_balance	0	NO		
arm_customer.customer_code	0	NO		
arm_customer.customer_comments	0	NO		
arm_customer.customer_credit_limit	0	NO		

Details Other Map Fields

Key field: Displayed field: Field label: ☐ No-label

Tooltip: Press F4 for Lookup

Browse title: Lookup

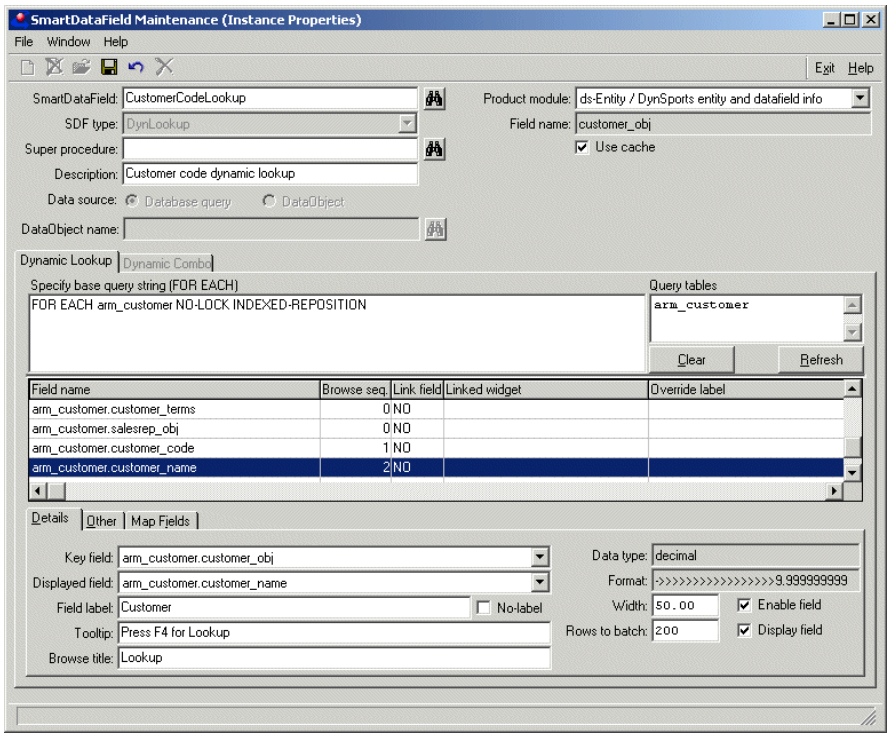
Data type: Format: Width: 50.00 ☒ Enable field

Rows to batch: 200 ☒ Display field

- 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 fields to display in the Customer browse that the dynamic Lookup launches and set their Display Sequences, as shown in the following table:

Field name	Display sequence
arm_customer.customer_code	1
arm_customer.customer_name	2

You can sort on a browse column by clicking on the **column header**. The following window 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 that displays the field’s value.

- 17 ♦ Type **armcufullo** for the **Maintenance SDO** and **oeCustFoldWin** for the **Maintenance object**. You will build a browse window called **oeCustFo1dWin** in the “[Creating folder windows](#)” section. 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:

SmartDataField Maintenance (Instance Properties)

File Window Help

SmartDataField: CustomerCodeLookup Product module: ds-Entity / DynSports entity and datafield info

SDF type: DynLookup Field name: customer_obj

Super procedure: Description: Customer code dynamic lookup

Data source: ☒ Database query ☐ DataObject

DataObject name:

Dynamic Lookup Dynamic Combo

Specify base query string (FOR EACH)

FOR EACH arm_customer NO-LOCK INDEXED-REPOSITION

Query tables

arm_customer

Clear Refresh

Field name	Browse seq.	Link field	Linked widget	Override label
arm_customer.customer_balance	0 YES		fiCustomerBalance	
arm_customer.login_company_obj	0 NO			
arm_customer.customer_comments	0 NO			
arm_customer.customer_credit_limit	0 NO			

Details Other Map Fields

Maintenance SDO: armcufullo

Maintenance object: oeCustFoldWin


Parent field: ☒ Popup lookup browse on ambiguous find

Parent filter query: ☐ Popup lookup browse on unique ambiguous find

☐ Popup lookup browse if no match could be found

☐ Blank out invalid value

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 ♦ Choose **Save** and exit the maintenance dialog box.
- 19 ♦ Select the **Fill In**  from the Object Palette for the field for the Linked Widget your defined for your Lookup.

- 20 ♦ Drop the fill-in onto the design window next to the binoculars icon representing the dynamic Lookup, as shown:

The screenshot shows a window titled "SmartDataViewer(DynView) - artorview.vv". It contains a form with the following fields and controls:

- Order code: [text box]
- Status: [text box] 0.000000000
- Sales rep: [dropdown menu]
- Customer: [text box] [binoculars icon]
- Fill 1: [text box]
- Order date: [text box]
- Promise date: [text box]
- Ship date: [text box]
- Carrier: [text box]
- Instructions: [text area]
- PD: [text box]
- Credit card: [text box]
- Terms: [text area]
- ☐ Incoming
- Warehouse: [text box] 0.000000000

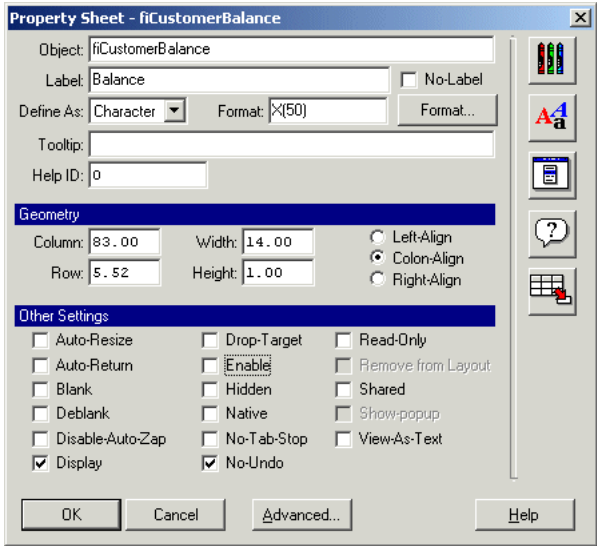
- 21 ♦ Type **fiCustomerBalance** for the **Object** and **Balance** for the **Label** in the AppBuilder main window, as shown:

The screenshot shows the AppBuilder main window. The "Object" field is set to "fiCustomerBalance" and the "Label" field is set to "Balance". The window also shows a menu bar (File, Edit, Compile, Build, Tools, Options, Layout, Window, Help) and a toolbar with various icons.

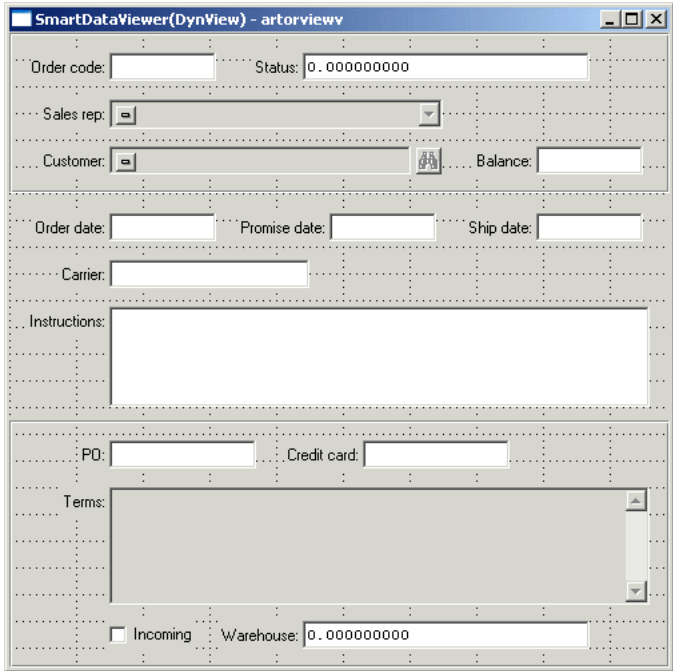
The dynamic Lookup makes the correct association with the Linked Widget name you specified when you defined your lookup, and displays the customer_balance field in this position.

- 22 ♦ Double-click the field to bring up its property sheet. Users should not edit the contents of this field, so you need to disable it.

23 ♦ Deselect the **Enable** toggle box, as shown:



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



25 ♦ Choose **Save**.

4.1.3 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 by creating dynamic Combos for the Status and Warehouse fields. You might want to read that section to refresh your memory before beginning.

Adding the Status dynamic Combo

To create the Status Combo:

- 1 ♦ Select **DynamicCombo** from the **Object Palette**.
- 2 ♦ Click on the **Status** field in the viewer.
- 3 ♦ Choose the **Create New SDF** button.
- 4 ♦ Set the values on the **SmartDataField Maintenance** dialog box, as shown in the following table:

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

- 5 ♦ Choose **Save** and exit the SmartDataField Maintenance dialog box.

Adding the Warehouse dynamic Combo

To create the Warehouse Combo:

- 1 ♦ Select **DynamicCombo** from the **Object Palette**.
- 2 ♦ Click on the **Warehouse** field in the viewer.
- 3 ♦ Choose the **Create New SDF** button.
- 4 ♦ Set the values on the **SmartDataField Maintenance** dialog box, as shown in the following table:

Field	Value
SmartDataField	WarehouseCombo
Description	Warehouse dynamic combo
Product module	ds-Entity
Specify base query string	FOR EACH ivm_warehouse NO-LOCK INDEXED-REPOSITION
Display seq. of ivm_warehouse.warehouse_code	1
Display seq. of ivm_warehouse.warehouse_name	2
Key field	ivm_warehouse.warehouse_obj
Field label	Warehouse
Tooltip	Select warehouse from list

- 5 ♦ Choose **Save** and exit the SmartDataField Maintenance dialog box.

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

The screenshot shows a window titled "SmartDataViewer(DynView) - artorvieww". The form is organized into several sections with dotted grid lines. The top section contains "Order code:" (text box), "Status:" (dropdown), "Sales rep:" (dropdown), "Customer:" (dropdown with a small icon), and "Balance:" (text box). The middle section contains "Order date:" (text box), "Promise date:" (text box), "Ship date:" (text box), and "Carrier:" (text box). Below this is a large "Instructions:" text area. The bottom section contains "PD:" (text box), "Credit card:" (text box), "Terms:" (large text area with a scrollbar), and "Incoming" (checkbox) next to "Warehouse:" (dropdown).

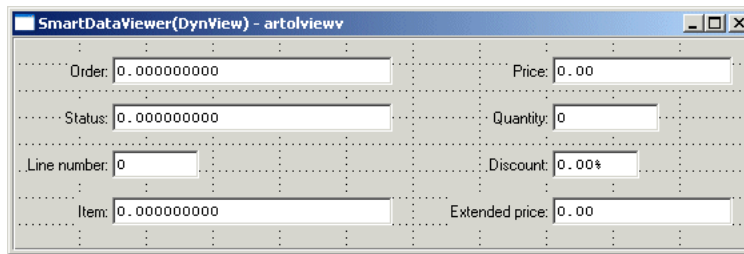
- 7 ♦ Close the viewer design window.

4.1.4 Modifying the Orderline viewer

The Orderline viewer also has several object fields on it. You need to 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 choose **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:



The screenshot shows a window titled "SmartDataViewer(DynView) - artolviewv". Inside the window is a form with eight text input fields arranged in a grid. The fields are labeled as follows:

Order: 0.000000000	Price: 0.00
Status: 0.000000000	Quantity: 0
Line number: 0	Discount: 0.00%
Item: 0.000000000	Extended price: 0.00

Adding the Order dynamic Lookup

To create the Order Lookup:

- 1 ♦ Select **DynamicLookup** from the **Object Palette**.
- 2 ♦ Click on the **Order** field in the viewer.
- 3 ♦ Choose the **Create New SDF** button.

- 4 ♦ Set the values on the **SmartDataField Maintenance** dialog box, as shown in the following table:

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

¹ You build this window later in the [“Creating an order maintenance window”](#) section.

- 5 ♦ Choose **Save** and exit the SmartDataField Maintenance dialog box.

Adding the Status dynamic Combo

To add the Status Combo:

- 1 ♦ Select **DynamicCombo** from the **Object Palette**.
- 2 ♦ Click on 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 ♦ Choose **Save** and exit the dialog box.

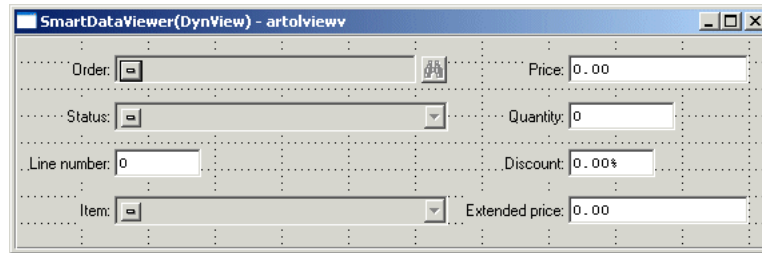
Adding the Item dynamic Combo

To create the Item Combo:

- 1 ♦ Select **DynamicCombo** from the **Object Palette**.
- 2 ♦ Click on the **Item** field in the viewer.
- 3 ♦ Choose the **Create New SDF** button.
- 4 ♦ Set the values on the **SmartDataField Maintenance** dialog box, as shown in the following table:

Field	Value
SmartDataField	ItemCombo
Description	Item dynamic combo
Product module	ds-Entity
Specify base query string	FOR EACH ivm_item NO-LOCK INDEXED-REPOSITION
Display seq. of ivm_item.item_code	1
Display seq. of ivm_item.item_name	2
Key field	ivm_item.item_obj
Field label	Item
Tooltip	Select item from list

- 5 ♦ Choose **Save** and exit the SmartDataField Maintenance dialog box. Your Orderline viewer should now look something like the following:



The image shows a dialog box titled "SmartDataViewer(DynView) - artolviewv". It contains several input fields arranged in a grid-like layout. The fields are: "Order:" with a dropdown menu, "Price:" with a text box containing "0.00", "Status:" with a dropdown menu, "Quantity:" with a text box containing "0", "Line number:" with a text box containing "0", "Discount:" with a text box containing "0.00%", "Item:" with a dropdown menu, and "Extended price:" with a text box containing "0.00".


- 6 ♦ Save your changes and close the design window.

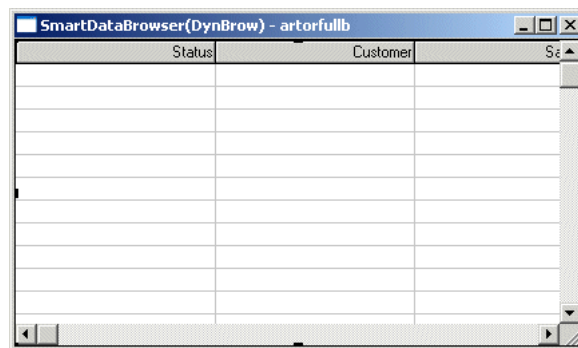
4.1.5 Hiding object fields in browses

The last piece of editing is to stop the object fields from being displayed in the browses. While the object fields are useful as targets for dynamic Lookups and Combos in the viewers, they are not useful in a browser. You can edit a browse's property sheet to hide fields that you do not want to appear in the browse. The information in the field is still available from the SDO for code such as triggers to run against, but it is not shown to the user.

Modifying the Order browse

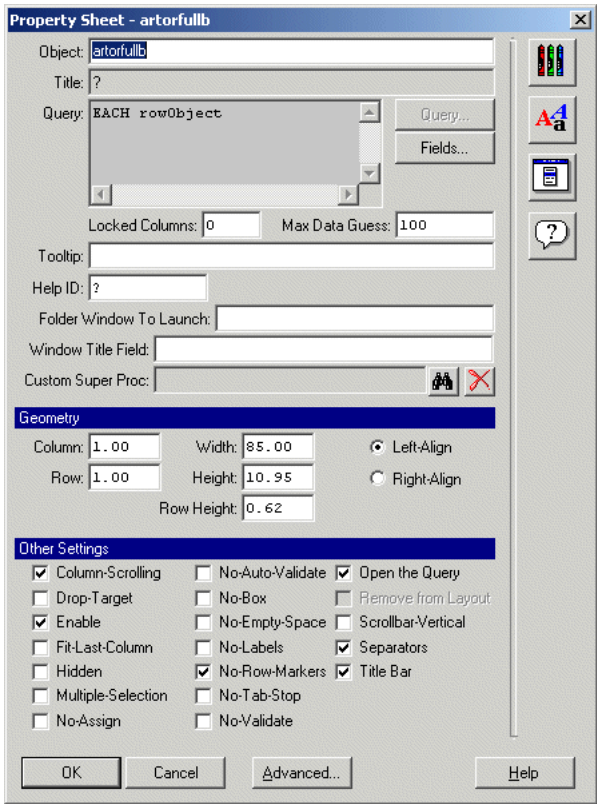
To hide the object fields on the Order browse:

- 1 ♦ Choose the **Open Object** icon  on the AppBuilder main window.
- 2 ♦ Double-click **artorfullb** to open the browse's design window:

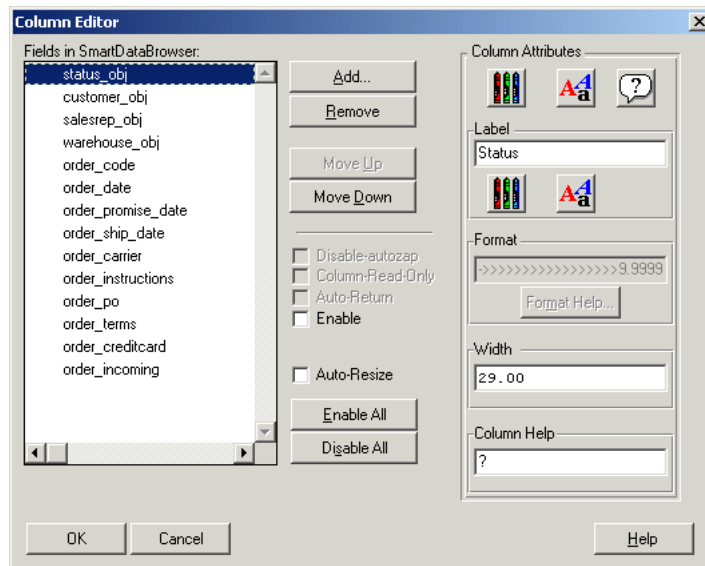


The image shows a design window titled "SmartDataBrowser(DynBrow) - artorfullb". It displays a table with three columns: "Status", "Customer", and "Sg". The table has multiple empty rows, indicating it is a data browser. The window includes standard window controls (minimize, maximize, close) and a scroll bar on the right side.

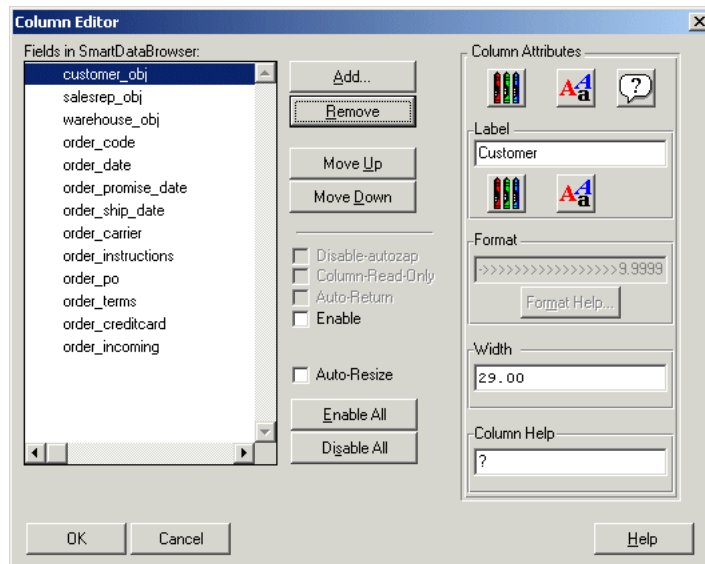
- 3 ♦ Double-click in the design window to open the property sheet:



- 4 ♦ Choose the **Fields** button to open the Column Editor:



- 5 ♦ Select **status_obj** in the **Fields in SmartDataBrowser** list.
- 6 ♦ Choose the **Remove** button. The field is removed from the list of displayed fields, as shown:



- 7 ♦ Remove the following other fields:
 - **customer_obj**
 - **salesrep_obj**
 - **warehouse_obj**
- 8 ♦ Choose **OK** to close the Column Editor.
- 9 ♦ Choose **OK** to close the property sheet and save your changes.
- 10 ♦ Save and close the browse's design window.

Modifying the Orderline browse

To hide the object fields on the Orderline browse:

- 1 ♦ Choose the **Open Object** icon on the AppBuilder main window.
- 2 ♦ Double-click **artolfullb** to open the browse's design window.
- 3 ♦ Open the property sheet's Column Editor and remove the following fields from the displayed field list:
 - **status_obj**
 - **item_obj**
 - **order_obj**
- 4 ♦ Choose **OK** to close the Column Editor.
- 5 ♦ Choose **OK** to close the property sheet and save your changes.
- 6 ♦ Save and close the browse's design window.

4.2 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.

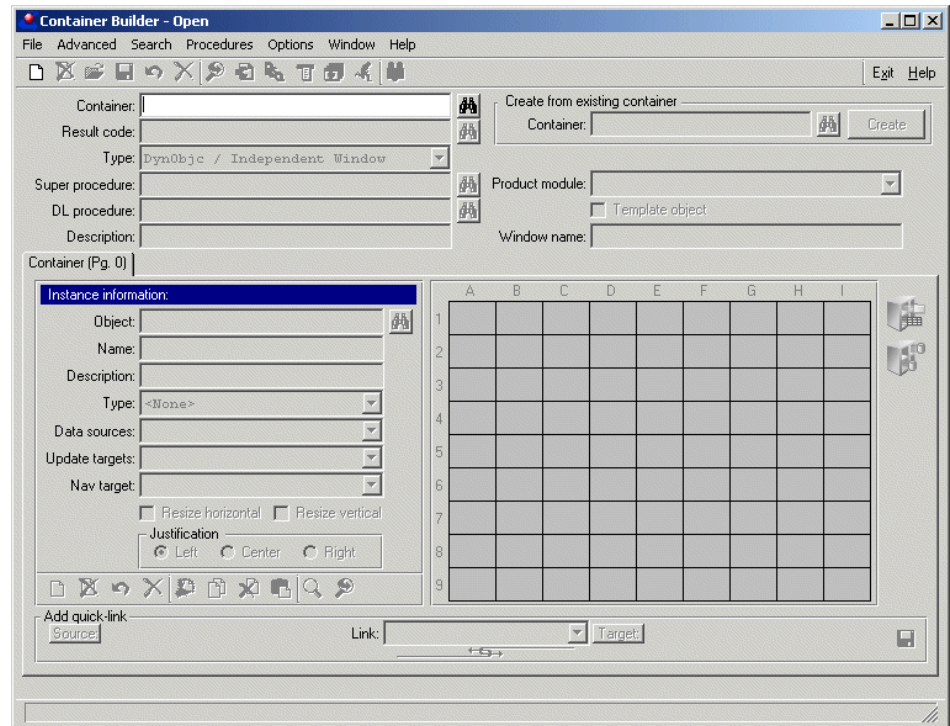
The Container Builder is the preferred tool for building complex containers in Progress Dynamics. The Container Builder manages the layout of objects and the links between them. When you save the container, these relationships are stored as records in the Repository.

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 the [Progress Dynamics Developer's Guide](#) for in-depth information on the Container Builder.


4.2.1 Defining the basic window

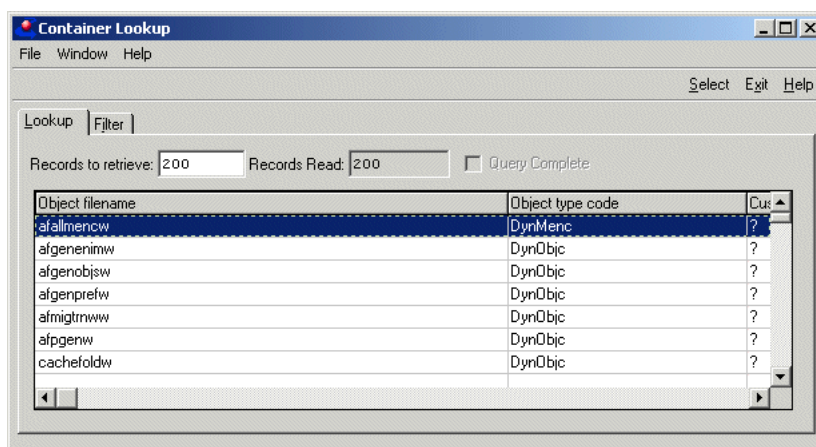
To create a Browse Window:

- 1 ♦ Select **Build**→**Container Builder** from the AppBuilder main window. The Container Builder appears. 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.

- 2 ♦ Choose the **New** button in the Container Builder toolbar to create a new container.
- 3 ♦ Choose the **Lookup** button  in the **Create from existing container** group. The Container Lookup dialog box appears:



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.

- 4 ♦ 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**.




- 5 ♦ Choose **Create**. This creates an instance of the template object for you to modify. Note that the **Type** is set as **DynObjc / Independent Window**.
- 6 ♦ Type **oeCustBrowseWin** in the **Container** field.

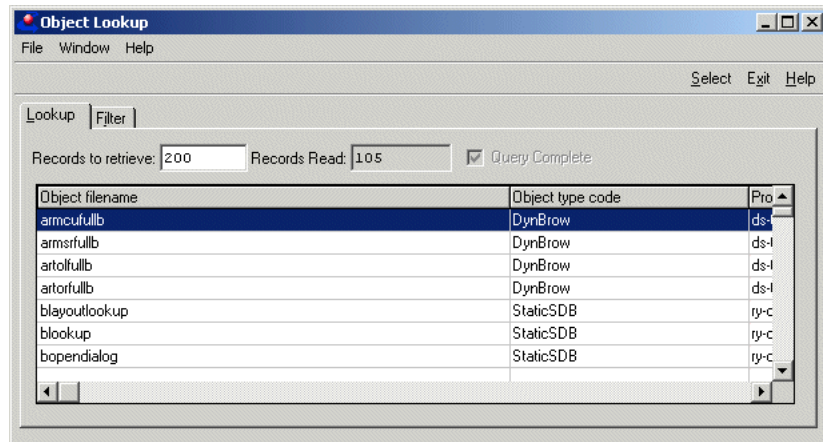
- 7 ♦ Type **Customer selection browse** as the **Description** for the new window.
- 8 ♦ Select **ds-OE** for the **Product Module**.
- 9 ♦ Type **Customer Browse** for the **Window Name**.


4.2.2 Replacing template objects on the folder page

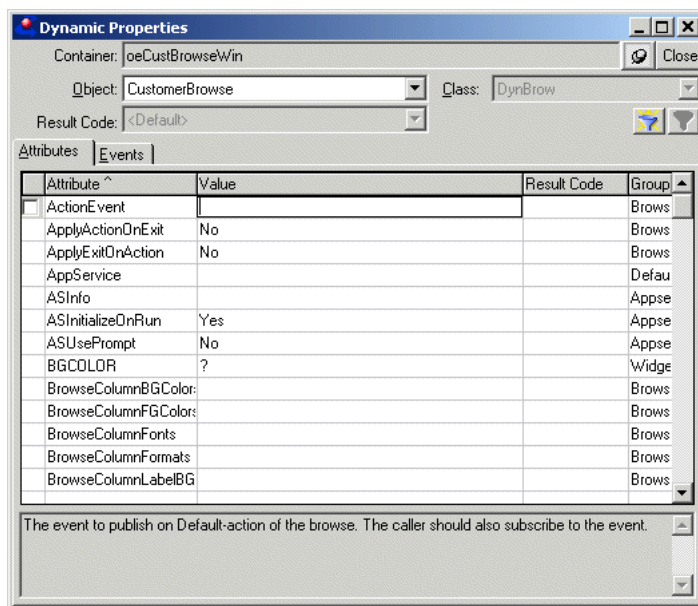
The right-hand side of the Container tab folder shows a grid. The visual grid acts as placeholder and layout for objects like toolbars, browses, and viewers. The nonvisual grid acts as placeholder for data sources like SDOs and SBOs. As shown in the grid, the window has three visual objects, two toolbars 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 ♦ Choose the **Show all visual objects** button  in the Container folder.
- 2 ♦ Choose **rytemfullb**, the browse template object , in the grid. The Instance information displays the data for the object.
- 3 ♦ Choose the **Lookup** icon  beside the **Object** field. The Object Lookup dialog box appears, filtered to display 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 ♦ Choose the **Dynamic Properties** button  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 property sheet for the browse appears:

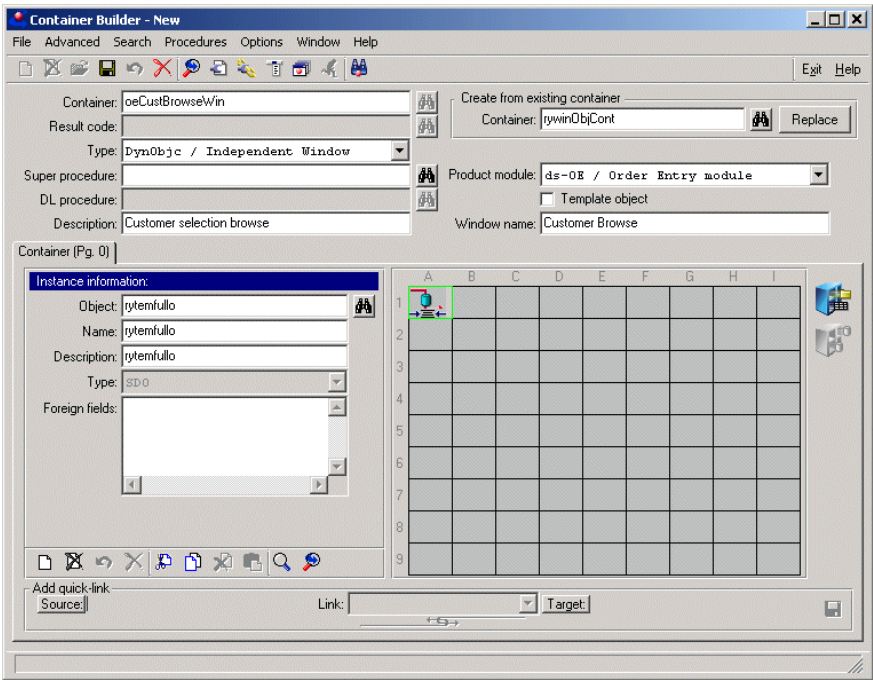



Attribute ^	Value	Result Code	Group
ActionEvent			Brows
ApplyActionOnExit	No		Brows
ApplyExitOnAction	No		Brows
AppService			Defau
ASInfo			Appse
ASInitializeOnRun	Yes		Appse
ASUsePrompt	No		Appse
BGCOLOR	?		Widge
BrowseColumnBGColor			Brows
BrowseColumnFGColor			Brows
BrowseColumnFonts			Brows
BrowseColumnFormats			Brows
BrowseColumnLabelBG			Brows

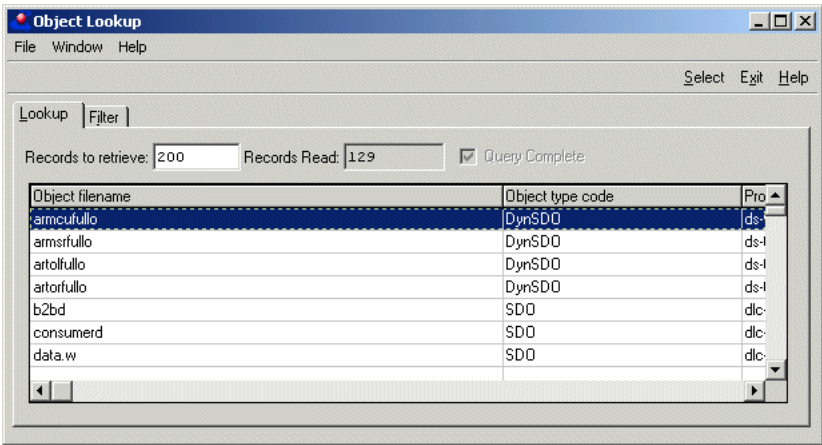
The event to publish on Default-action of the browse. The caller should also subscribe to the event.

- 7 ♦ Type **oeCustFoldWin** for the **FolderWindowToLaunch** attribute. This is the Customer Maintenance window you will build in the [“Creating folder windows”](#) section. 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.

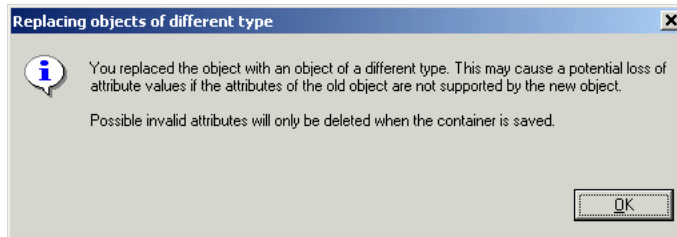
- 9 ♦ Choose the **Show all non-visual objects** button  in the Container folder. The grid now displays the template's SDO:



- 10 ♦ Choose the **Lookup** button  next to the **Object** field. The Object Lookup dialog box filtered to display only SDOs:




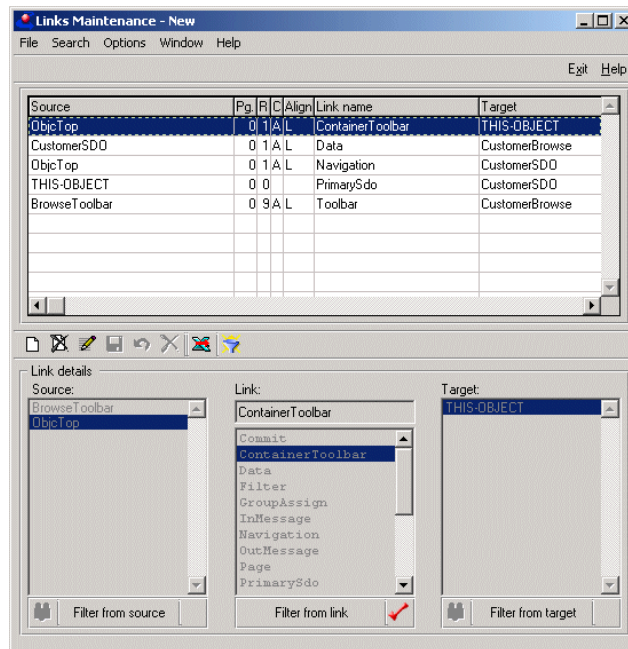
- 11 ♦ Double-click **armcufullo**, the customer SDO, to select it. You automatically return to the Container Builder, and the following message dialog box appears:



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 some objects do not have all of 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 ♦ Choose **OK** to continue.
- 13 ♦ Type **CustomerSDO** as the **Name** for this instance of the armcufullo SDO.

- 14 ♦ Choose the **Links Maintenance** button  in the container toolbar. The Links Maintenance dialog box appears:



SmartObjects communicate through named connections called Progress 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 the [Progress Dynamics Developer's Guide](#) for more 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 links in the following table are defined for the objects on the oeCustBrowseWin window. Click on the **Source** column header to sort the links:

Source	Link name	Target	Description
BrowseToolbar	Toolbar	CustomerBrowse	Enables toolbar to execute browse functions
CustomerSDO	Data	CustomerBrowse	Passes data values to browse
ObjcTop	ContainerToolbar	THIS-OBJECT	Enables toolbar to execute container functions
ObjcTop	Navigation	CustomerSDO	Notifies SDO to navigate through data records
THIS-OBJECT	PrimarySDO	CustomerSDO	Indicates to container that CustomerSDO is primary SDO

NOTE: THIS-OBJECT refers to the container, oeCustBrowseWin.

- 16 ♦ Exit the Links Maintenance dialog box.
- 17 ♦ Choose **Save**, and exit the Container Builder.

4.2.3 Creating an Order Browse

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

Table 4–1: Order Browse window properties

For . . .	Substitute . . .	With . . .
Container name	oeCustBrowseWin	oeOrderBrowseWin
Container description	Customer selection browse	Order selection browse
Window name	Customer Browse	Order Browse
Template browse	armcufullb	artorfullb
Browse instance name	CustomerBrowse	OrderBrowse
FolderWindowToLaunch attribute	oeCustFoldWin	oeOrderFoldWin
Template SDO	armcufullo	artorfullo
SDO instance name	CustomerSDO	OrderSDO

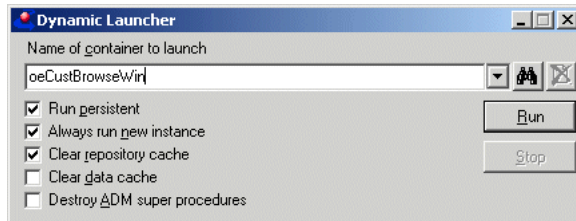
When you complete building your Order Browse window, you are ready to see the results of your work. You can now run the windows and see what they look like.

4.2.4 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 ♦ Select **Compile→Dynamic Launcher** from the AppBuilder main window.
- 2 ♦ Type **oeCustBrowseWin** in the **Name of Container to Launch** field:

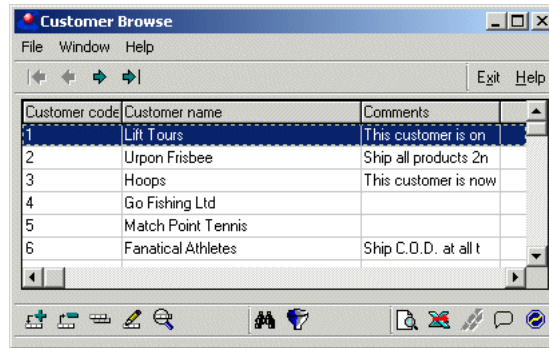



The Dynamic Launcher has the following options:

- 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 the object definition data cached on the client from the Repository. This lets you to see the effects of your latest work. Leave this toggle box selected whenever you launch 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 the 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.

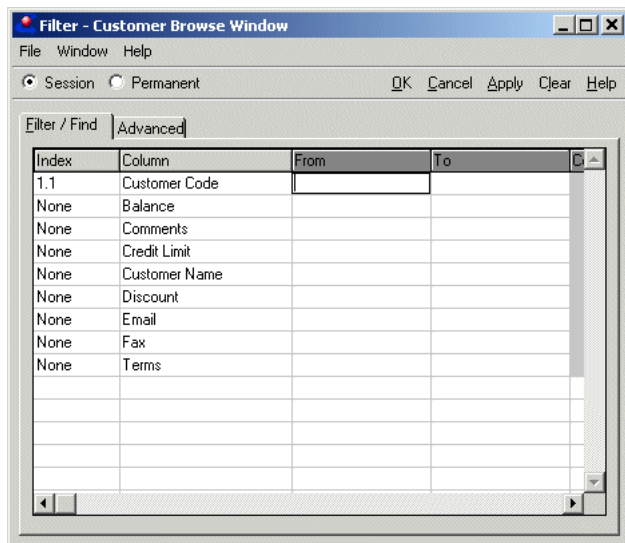
NOTE: Occasionally, you can clear the cache, restart the super procedures, and still not get the results you expect. If that happens, restart your session. That clears any remaining cached information.

- 3 ♦ Choose **Run**. The Browser window now appears:



- 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 ♦ Choose the **Filter Records** button  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.

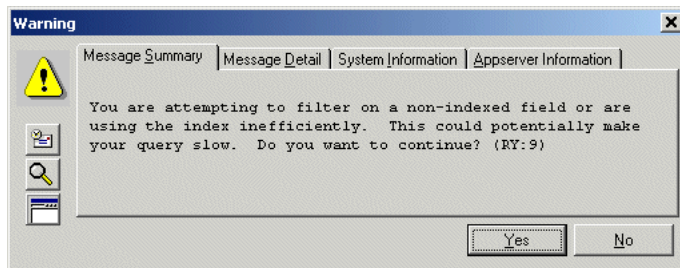
Entering a value in just the **From** column filters all values greater than or equal to that value. Scroll the browser to the right, or resize it, and you can see that fields with a word index can be filtered by contained words, and all character fields can be filtered on a character string match:





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 what are 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:



- 6 ♦ Choose **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 choose the **Find** button  to reposition to the first record matching the criteria you enter.

The other standard buttons export values from the browse to a XML document or an Excel™ spreadsheet, 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.

4.3 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 ♦ Select **Build→Container Builder** from the AppBuilder main window. The Container Builder appears.
- 2 ♦ Choose the **New** button in the Container Builder toolbar.
- 3 ♦ Type **rywinFolder** in the **Container** field in the **Create from existing container** section, and press the **TAB** key.

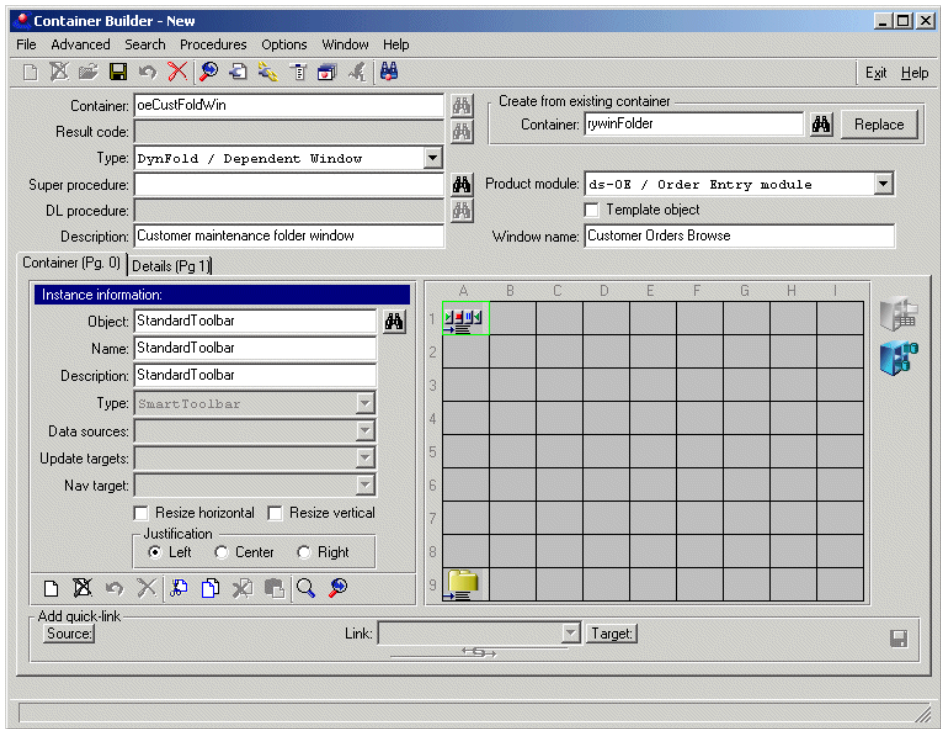
The rywinFolder object is a template for a dependent window. This rywinFolder template has a Page 0 with a Navigation and Update toolbar (StandardToolbar), the tab folder itself, and a folder Page 1 where a Viewer goes to display and update records.


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

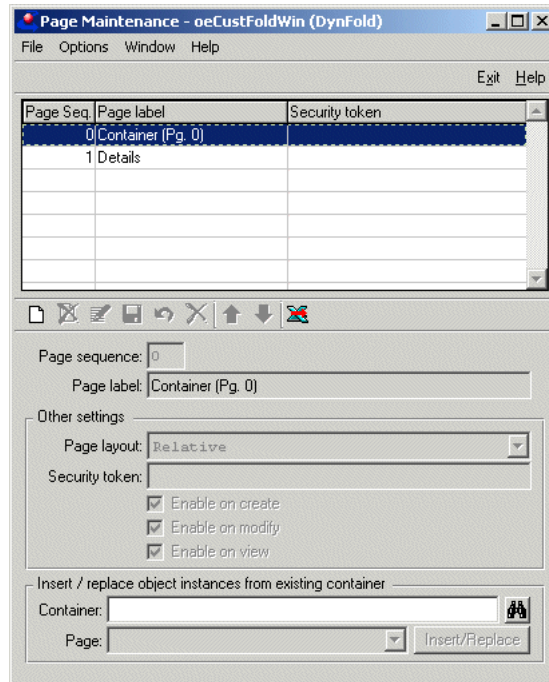
- 4 ♦ Choose the **Create** button to create the folder window.


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:

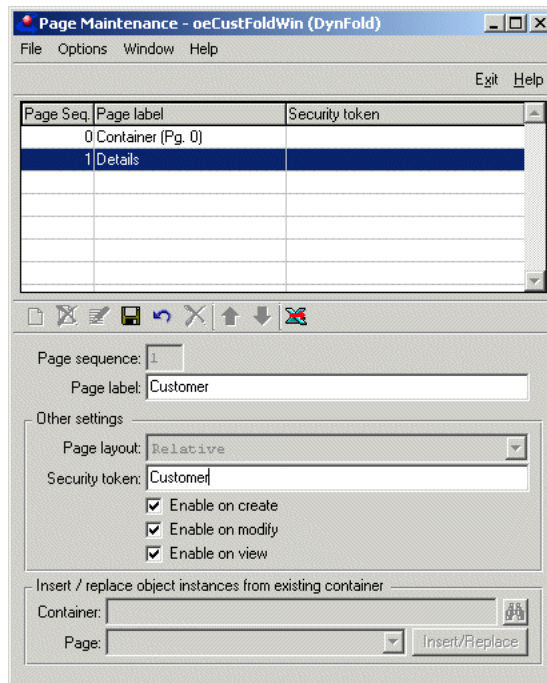


- 9 ♦ Choose the **Page Maintenance** button  in the Container Builder toolbar. The Page Maintenance window appears. This tool allows you to manage data about pages in your folder windows, as shown:



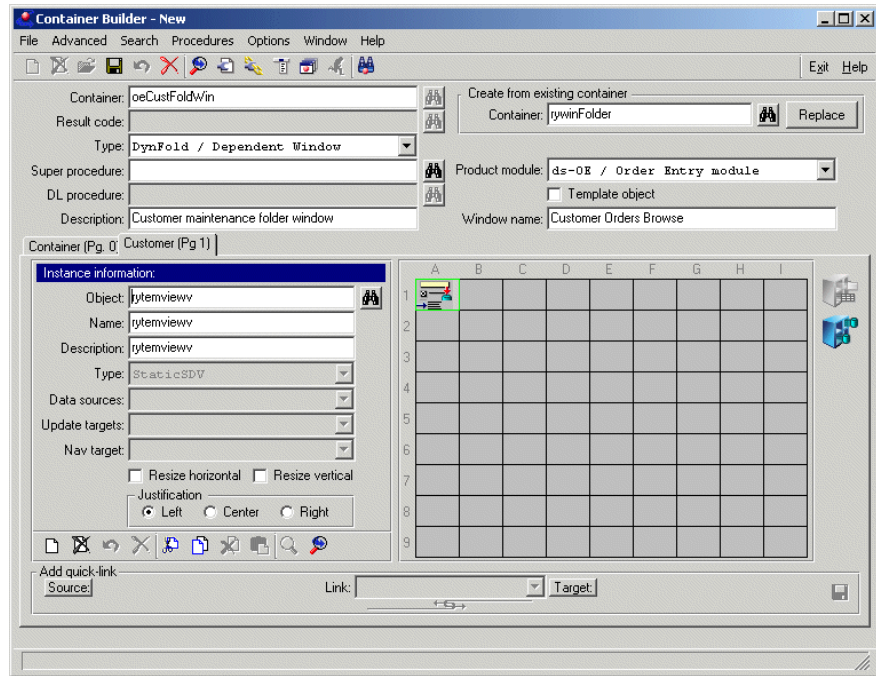
- 10 ♦ Select the **Details** record, and choose the **Modify** button .


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



- 12 ♦ Choose **Save**, and exit the Page Maintenance window.

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



- 14 ♦ Select the template viewer icon  in the grid.
- 15 ♦ Type **armcuvviewv**, your Customer Viewer, in the **Object** field and press the **TAB** key.
The Container Builder substitutes the dynamic Viewer for the template object.
- 16 ♦ Choose **OK** to clear the **Replacing objects of different type** message box.
- 17 ♦ Type **CustomerViewer** for the **Name**.
- 18 ♦ Choose **Save** in the container toolbar.


Now you have filled in the Folder template as it was defined. You can also expand upon what the template gave you by adding more folder pages and objects.

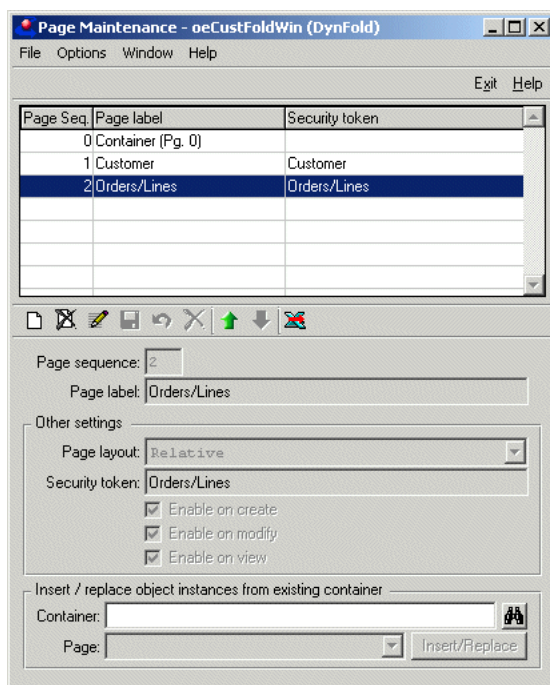
4.3.1 Adding another page to the folder window

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.

Adding a page

To add a new page to your folder window:

- 1 ♦ Choose the Page Maintenance button . The Page Maintenance window appears.
- 2 ♦ Choose **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 ♦ Choose **Save**. Note that the Security token is automatically set to the Page label, as shown:



Page Seq	Page label	Security token
0	Container (Pg. 0)	
1	Customer	Customer
2	Orders/Lines	Orders/Lines

Page sequence: 2

Page label: Orders/Lines

Other settings:

Page layout: Relative

Security token: Orders/Lines

☒ Enable on create

☒ Enable on modify

☒ Enable on view

Insert / replace object instances from existing container

Container:

Page:

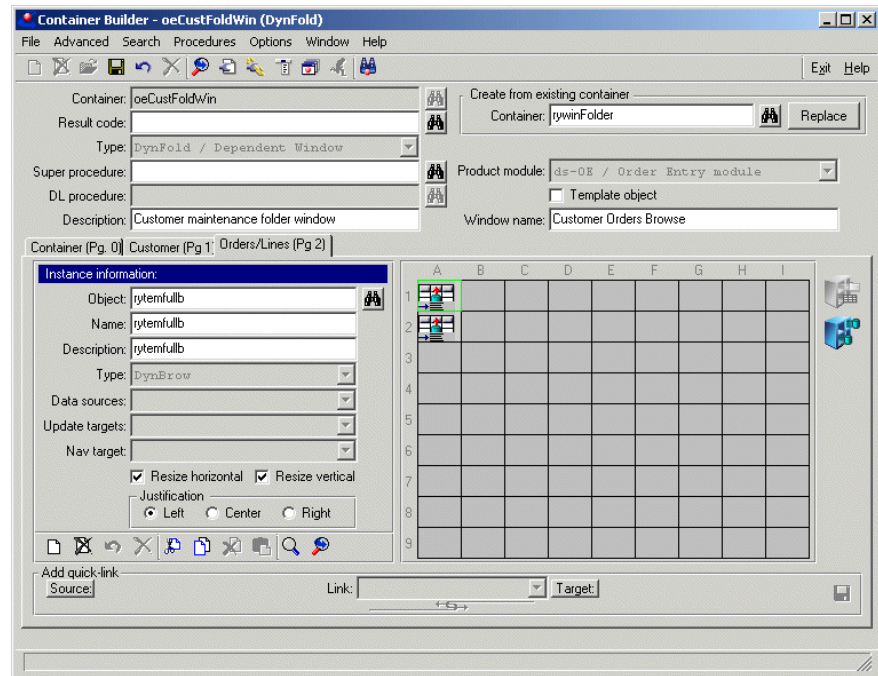
- 5 ♦ Type **rywinparentchild** in the **Container** field and press the **TAB** key.
- 6 ♦ Choose the **Insert/Replace** button, and exit the Page Maintenance window.

Modifying the new page

To modify the new folder page:

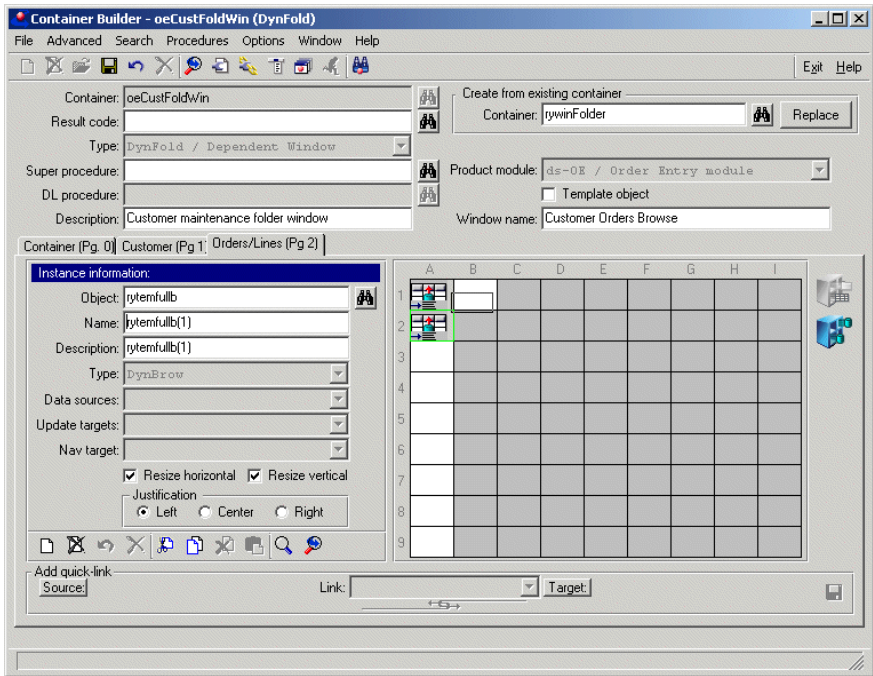
- 1 ♦ Choose the **Orders/Lines (Pg 2)** tab in the Container Builder.


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

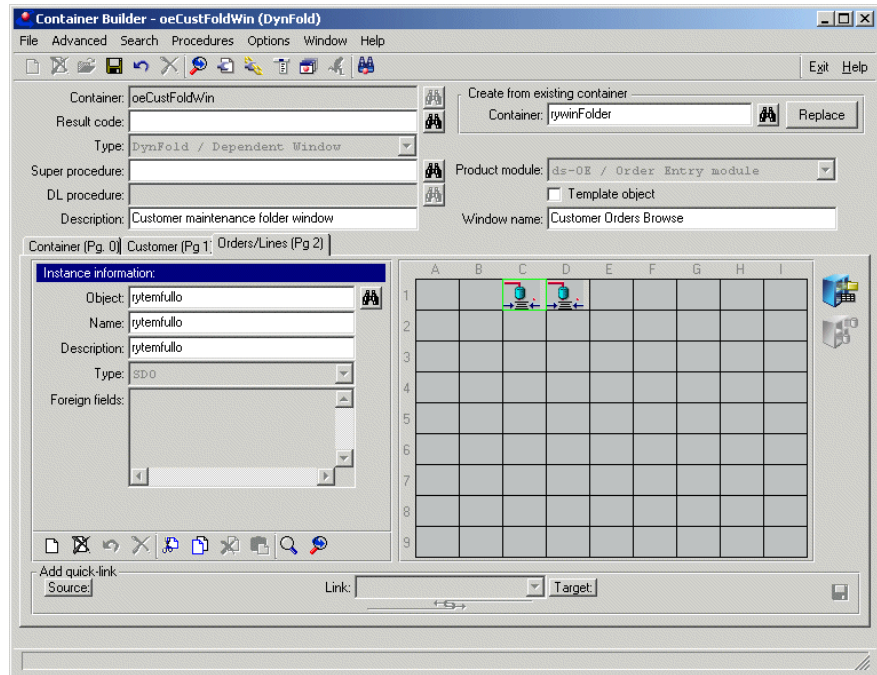


- 2 ♦ Select the icon in **Cell A-2** on the grid. You can change the layout of the window by rearranging these icons.

- 3 ♦ Left-click the icon, and drag and drop it on **Cell B-1**, as shown:



- 4 ♦ Choose the **Show all non-visual objects** button  :



- 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 ♦ Choose **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 **artolfullo**, your Order SDO, in the **Object** field and press the **TAB** key. The Container Builder substitutes the dynamic SDO for the template object.
- 11 ♦ Choose **OK** to clear the **Replacing objects of different type** message box.
- 12 ♦ Type **OrderlineSDO** for the **Name**.

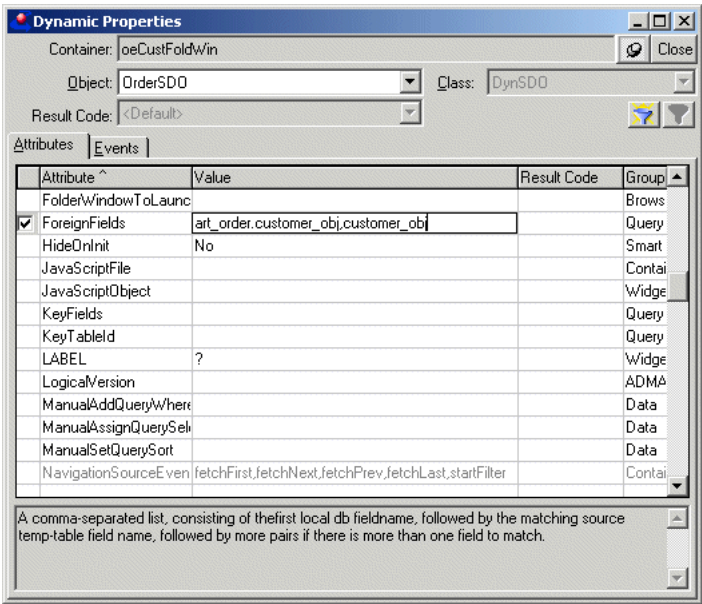
4.3.2 **Setting the SDO foreign fields**

Now that your new page has its SDOs, you need to 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.

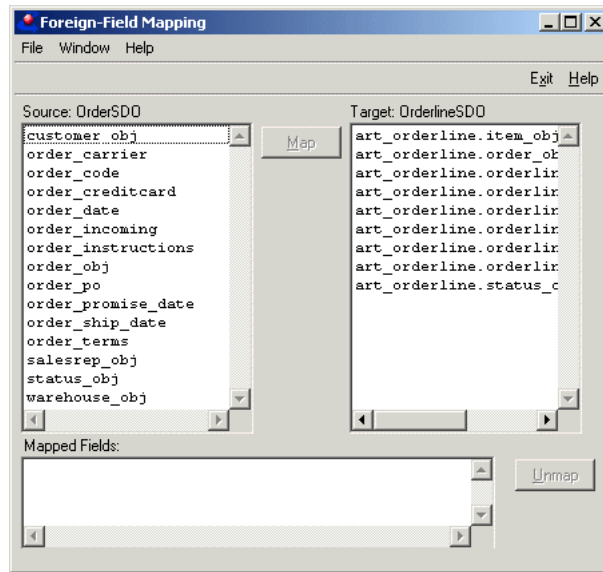
To set the foreign fields:

- 1 ♦ Select the **artorfullo** SDO in the folder’s grid.
- 2 ♦ Choose the **Dynamic Properties** button from the instance toolbar inside the folder page.
- 3 ♦ Type **art_order.customer_obj,customer_obj** for the value of the **ForeignFields** attribute and close the property sheet, as shown:



- 4 ♦ Select the **artolfullo** SDO in the folder’s grid.

- 5 ♦ Choose the **Foreign-Field Mapping** button  beside the Foreign fields editor. The Foreign-Field Mapping dialog box appears:





- 6 ♦ Select **order_obj** in the **Source** column and **art_orderline.order_obj** in the **Target** column.
- 7 ♦ Choose the **Map** button, and exit the Foreign-Field Mapping dialog box.

4.3.3 Adding browsers on the new page

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

To add the browsers:

- 1 ♦ Choose the **Show all visual objects** button  in the Container folder.
- 2 ♦ Select the **DynBrow** object  in **Cell A-1**.
- 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 ♦ Choose **Save** to save your container.

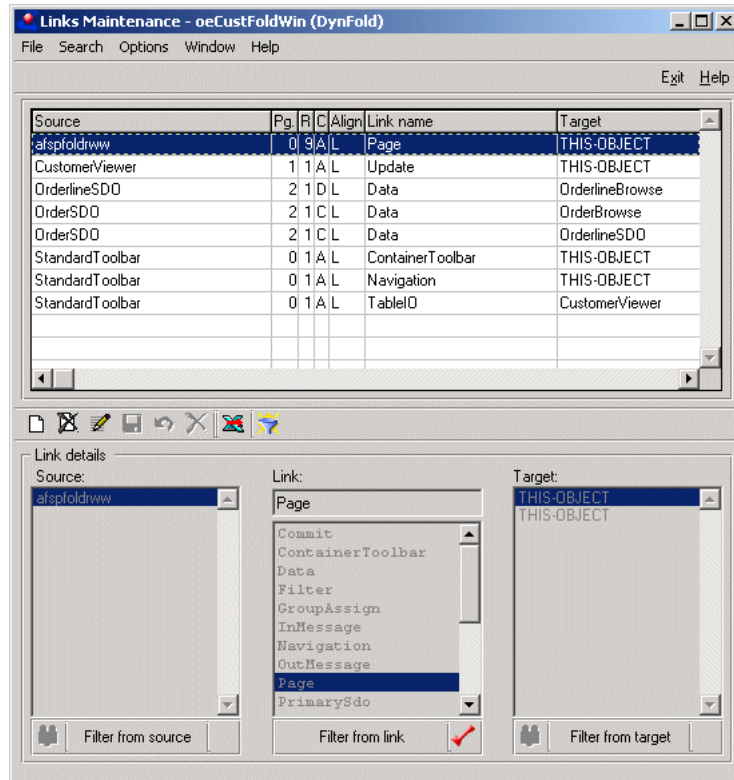
4.3.4 Setting links for the new page

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 need to define a link between your container window and that SDO, so the framework can route the data.

To set the links for the new page:

- 1 ♦ Choose the **Links Maintenance** button  in the Container Builder toolbar. The Links Maintenance window displays all the existing links for your window. These are the links for both template containers, as shown:



- 2 ♦ Choose **New**.
- 3 ♦ Select **THIS-OBJECT** as the **Source**, **Data** as the **Link**, and **OrderSDO** as the **Target**.
NOTE: If you do not see the appropriate choice in the lists, choose the **Filter from source** button under the list.
- 4 ♦ Choose **Save**.
- 5 ♦ Choose **New**.
- 6 ♦ Select **THIS-OBJECT** as the **Source**, **Data** as the **Link**, and **CustomerViewer** as the **Target**.

- 7 ♦ Choose **Save**.
- 8 ♦ Verify that the Links Maintenance window now displays all of the links in the following table:

Source	Link	Target	Description
afspfoldrww	Page	THIS-OBJECT	Enables folder to communicate to container for page handling.
CustomerViewer	Update	THIS-OBJECT	Saves customer changes back to customer SDO.
OrderlineSDO	Data	OrderlineBrowse	Passes orderline data values to orderline browse.
OrderSDO	Data	OrderBrowse	Passes order data values to order browse.
OrderSDO	Data	OrderlineSDO	Passes order data (foreign field key) to order line SDO.
StandardToolbar	ContainerToolbar	THIS-OBJECT	Enables toolbar to execute container functions.
StandardToolbar	Navigation	THIS-OBJECT	Notifies SDO to navigate through data records.
StandardToolbar	TableIO	CustomerViewer	Enables toolbar to request customer viewer to allow changes.
THIS-OBJECT	Data	OrderSDO	Passes customer data (foreign field key) to order SDO.
THIS-OBJECT	Data	CustomerViewer	Passes customer data to customer viewer.

NOTE: 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.

- 9 ♦ Exit the Links Maintenance window.
- 10 ♦ Choose **Save** in the Container Builder toolbar, then exit the Container Builder.

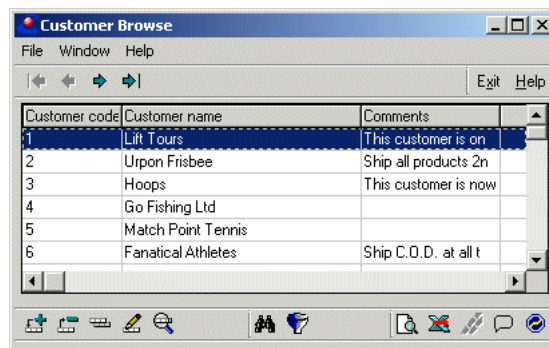
4.3.5 Running your folder window


To see the results of your work, you need to 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.

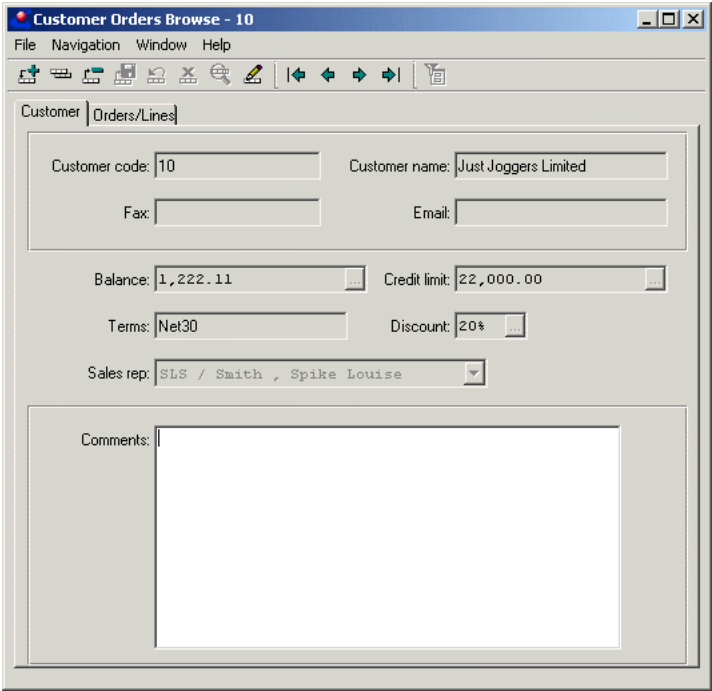
NOTE: Remember that you must have the Repository and the DynSports databases connected to run your application. The Repository is automatically connected by the framework managers when you start Progress Dynamics. However, you have not modified the configuration file yet to instruct the managers to connect 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 choose **Run**. Your Customer Browse appears:



- 3 ♦ Select a customer and choose the **Modify** button  in the bottom toolbar. The Customer Orders Browse appears:



Customer Orders Browse - 10

File Navigation Window Help

Customer Orders/Lines

Customer code: 10 Customer name: Just Joggers Limited

Fax: Email:

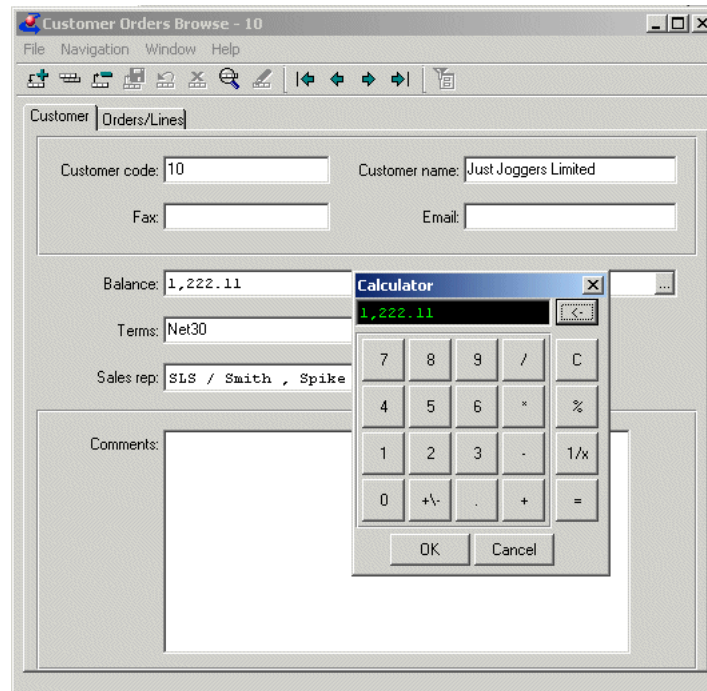
Balance: 1,222.11 Credit limit: 22,000.00

Terms: Net30 Discount: 20%

Sales rep: SLS / Smith, Spike Louise

Comments:

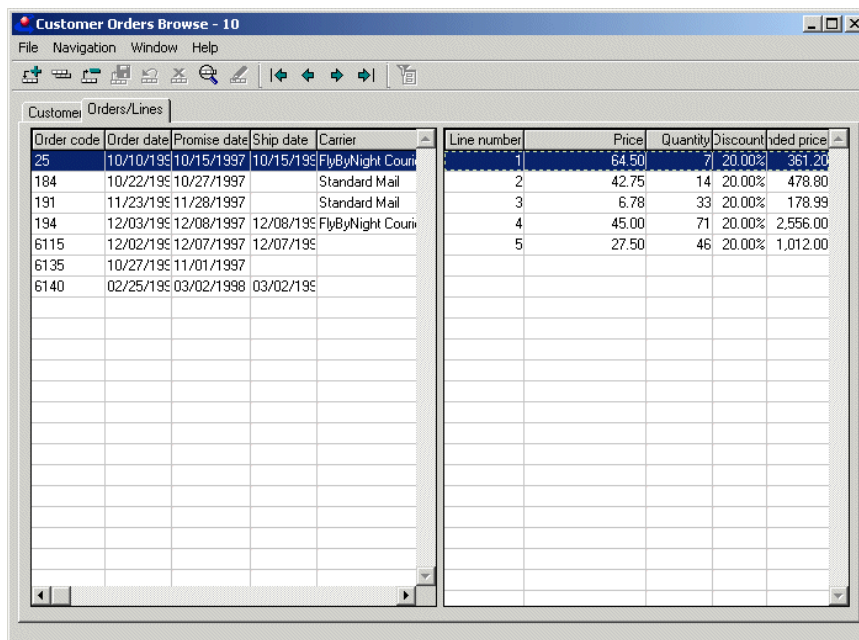
Look over the Customer tab. Note the Sales Rep combo box that you defined for the viewer earlier. If you choose the button next to one of the decimal fields, **Credit Limit** or **Balance**, a calculator appears, as shown:



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 Progress 4GL-based objects, not ActiveX controls.

- 4 ♦ Select the **Orders/Lines** tab.

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, as shown:



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

4.3.6 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, browsers 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.

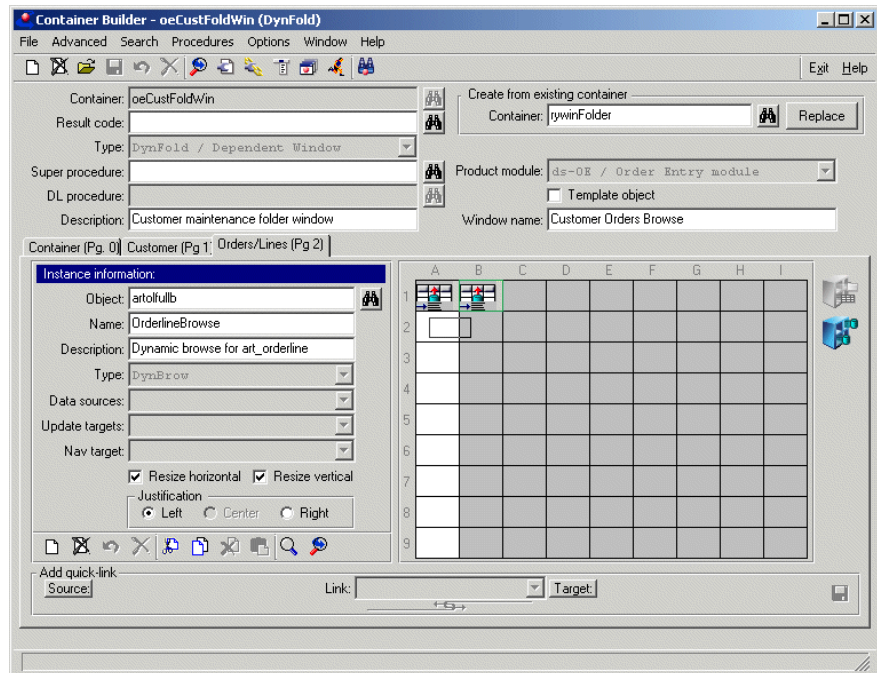
The Parent-Child window apportioned the available space for the two browsers, after laying out the toolbar and the Tab Folder. If you resize the window, the browsers resize. You can easily change these characteristics to get a different arrangement.

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 Browsers.
- 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:

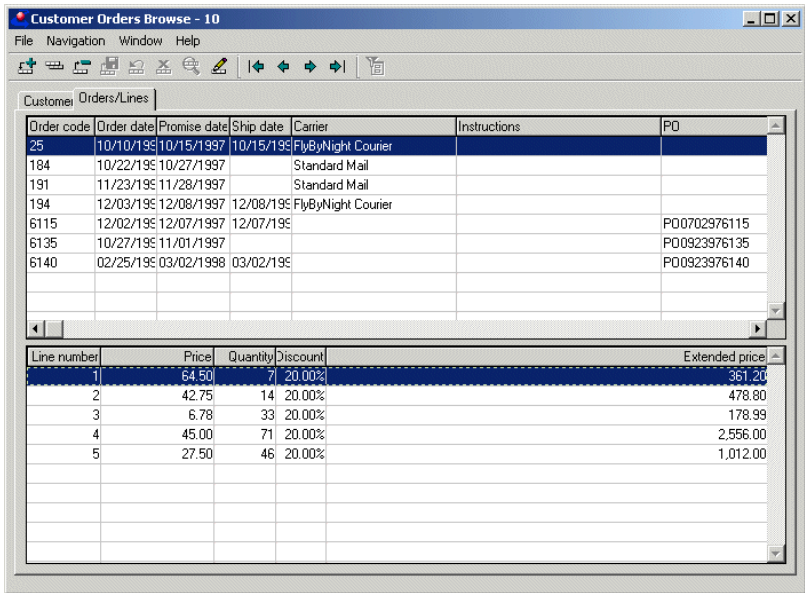


- 7 ♦ Choose **Save**, and exit the Container Builder.

You need to 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 need to 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.

- 8 ♦ Choose **Compile→Dynamic Launcher**. The Dynamic Launcher appears.
- 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:



The screenshot shows a window titled "Customer Orders Browse - 10" with a menu bar (File, Navigation, Window, Help) and a toolbar. The window is divided into two panes. The top pane, labeled "Customer Orders/Lines", contains a table with columns: Order code, Order date, Promise date, Ship date, Carrier, Instructions, and PO. The bottom pane contains a table with columns: Line number, Price, Quantity, Discount, and Extended price.

Order code	Order date	Promise date	Ship date	Carrier	Instructions	PO
25	10/10/1997	10/15/1997	10/15/1997	FlyByNight Courier		
184	10/22/1997	10/27/1997		Standard Mail		
191	11/23/1997	11/28/1997		Standard Mail		
194	12/03/1997	12/08/1997	12/08/1997	FlyByNight Courier		
6115	12/02/1997	12/07/1997	12/07/1997			P00702976115
6135	10/27/1997	11/01/1997				P00923976135
6140	02/25/1998	03/02/1998	03/02/1998			P00923976140

Line number	Price	Quantity	Discount	Extended price
1	64.50	7	20.00%	361.20
2	42.75	14	20.00%	478.80
3	6.78	33	20.00%	178.99
4	45.00	71	20.00%	2,556.00
5	27.50	46	20.00%	1,012.00

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.

4.4 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.

To build an Order maintenance window:

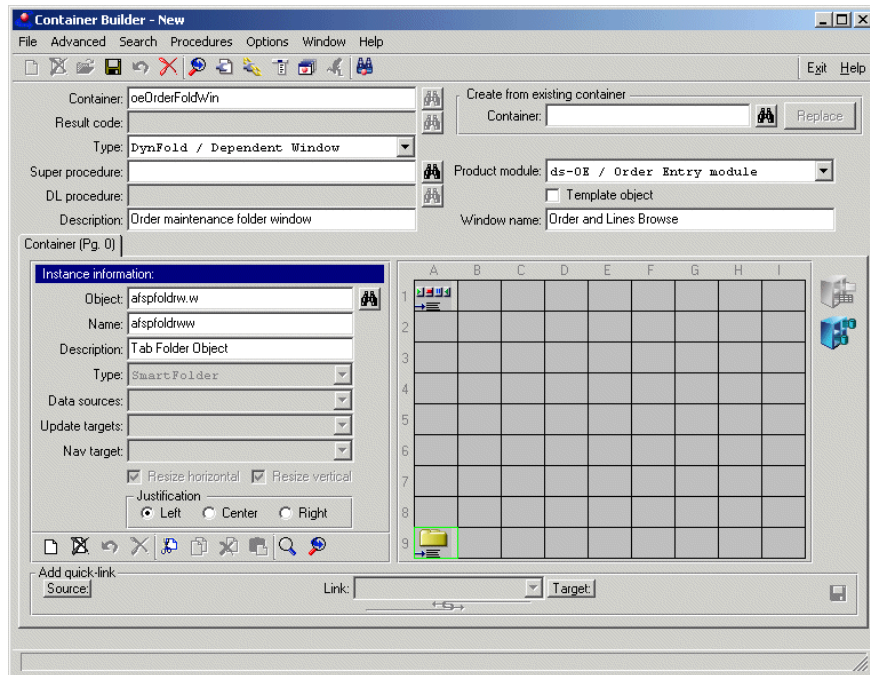
- 1 ♦ Select **Build**→**Container Builder** from the AppBuilder main window. The Container Builder appears.
- 2 ♦ Choose **New** in the Container Builder.
- 3 ♦ Set the values for your new window as shown in the following table:

Field	Value
Container	oeOrderFoldWin
Type	DynFold / Dependent Window
Description	Order maintenance folder window
Product module	ds-OE / Order Entry module
Window name	Order and Lines Browse

- 4 ♦ Add the objects shown in the following table to Page 0 of your folder, using the **New** button in the folder's object instances toolbar:


Object filename	Grid location
StandardToolbar	A1
afspfowdrw.w	A9

The `afspfoldrw.w` file is a Progress SmartFolder™ object. It handles multi-page windows and supports interactions with them. The Container Builder should look something like the following:

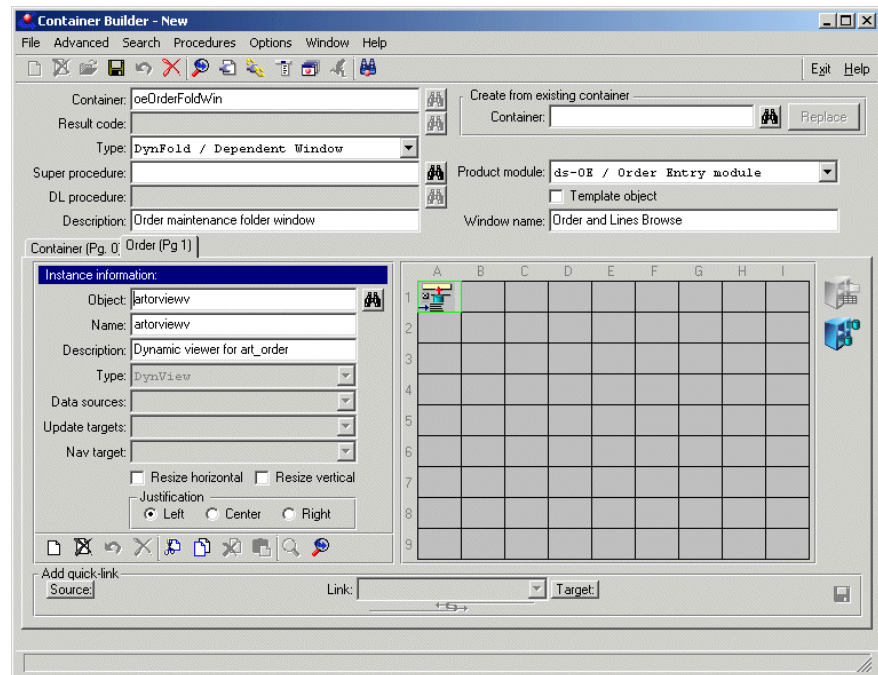


4.4.1 Adding an Order page

To add an Order page to the tab folder:

- 1 ♦ Choose the **Page Maintenance**  button. The Page Maintenance dialog box appears.
- 2 ♦ Choose **New** in the Page Maintenance dialog box.
- 3 ♦ Type **Order** for the **Page label**.
- 4 ♦ Choose **Save**, then exit the Page Maintenance dialog box.
- 5 ♦ Choose the **Order** tab.
- 6 ♦ Choose the **New** button in the folder's object instances toolbar.
- 7 ♦ Type `artorvieww` 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 the following:



- 9 ♦ Type **OrderViewer** for the **Name**, and choose **Save**.

4.4.2 Linking the Order page

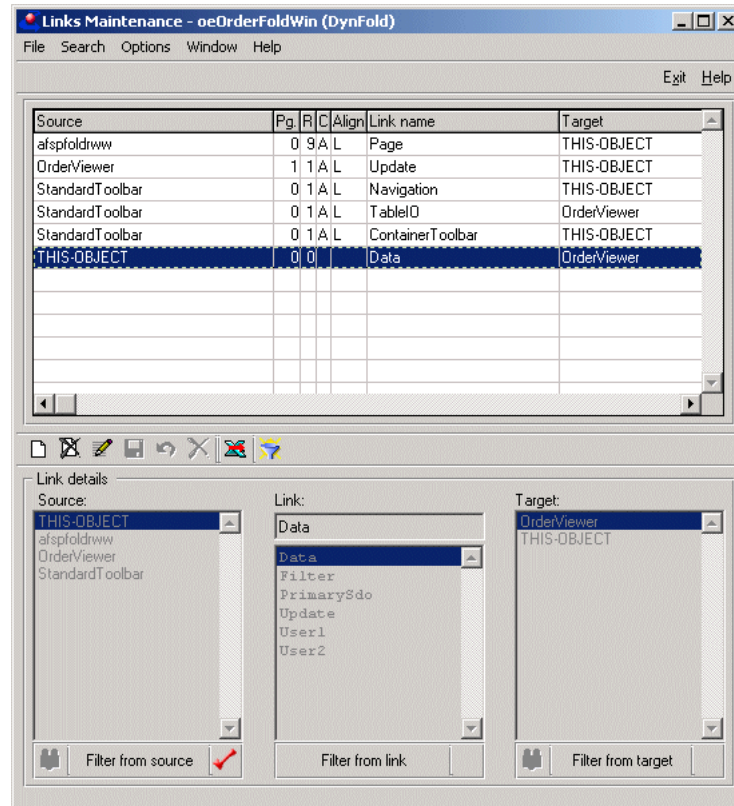
To set up the appropriate links for the Order page:

- 1 ♦ Choose the **Links Maintenance** button . 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:

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

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:



- 4 ♦ Choose **Save** in the Container Builder.

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

4.4.3 Adding an Order lines page

Next, you extend the basic window by adding a new page with a browse of the OrderLines of 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 ♦ Choose the **Page Maintenance** button. The Page Maintenance dialog box appears.
- 2 ♦ Choose **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 ♦ Choose **Save**.
- 5 ♦ Type **rywinbrsdynvw** in the **Container** field and press the **TAB** key.
- 6 ♦ Choose **Insert/Replace**, and then exit the Page Maintenance dialog box.

4.4.4 Adding objects to the Order lines page

Now, you need to 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:

Replace . . .	With . . .	Name
SDO template	artolfullo	OrderlineSDO
Dynamic browse template	artolfullb	OrderlineBrowse
Dynamic viewer template	artolviewv	OrderlineViewer

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 ♦ Choose **Save**.

4.4.5 Linking the Order Lines page

As on the Customer Orders Browse, you need to 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 ♦ Choose the **Links Maintenance** button. The Links Maintenance dialog box appears.
- 2 ♦ Verify that all of the links in the following table exist. Add any links that are missing:

Source	Type	Target
afspfoldrww	Page	THIS-OBJECT
OrderlineSDO	Data	OrderlineBrowse
OrderlineSDO	Data	OrderlineViewer
OrderlineViewer	Update	OrderlineSDO
OrderViewer	Update	THIS-OBJECT
StandardToolbar	ContainerToolbar	THIS-OBJECT
StandardToolbar	Navigation	THIS-OBJECT
StandardToolbar	TableIO	OrderViewer
StandardToolbar(1)	TableIO	OrderlineViewer
StandardToolbar(1)	ContainerToolbar	THIS-OBJECT
StandardToolbar(1)	Navigation	OrderlineSDO
THIS-OBJECT	Data	OrderlineSDO
THIS-OBJECT	Data	OrderViewer

- 3 ♦ Exit the Links Maintenance window.
- 4 ♦ Choose **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:

The screenshot shows a software window titled "Order and Lines Browse - 5". It features a menu bar with "File", "Navigation", "Window", and "Help". Below the menu is a toolbar with various icons. The window is divided into two tabs: "Order" and "Order lines". The "Order" tab is selected, displaying a form with the following fields and values:

- Order code: 5
- Status: in_Ship / Shipped
- Sales rep: JAL / Jan Loopsnel
- Customer: Birdy's Badminton
- Balance: 1189.04
- Order date: 02/12/1998
- Promise date: 02/17/1998
- Ship date: 02/17/1998
- Carrier: Standard Mail
- Instructions: (empty text area)
- PO: (empty text field)
- Credit card: Visa
- Terms: Net30
- Warehouse: 1 / Northeast USA
- ☐ Incoming

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 as an exercise, after you have read the chapter on using the Container Builder in the *Progress Dynamics Developer's Guide*.

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 this way to show you as much of the framework as possible while building a few objects.

4.5 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:

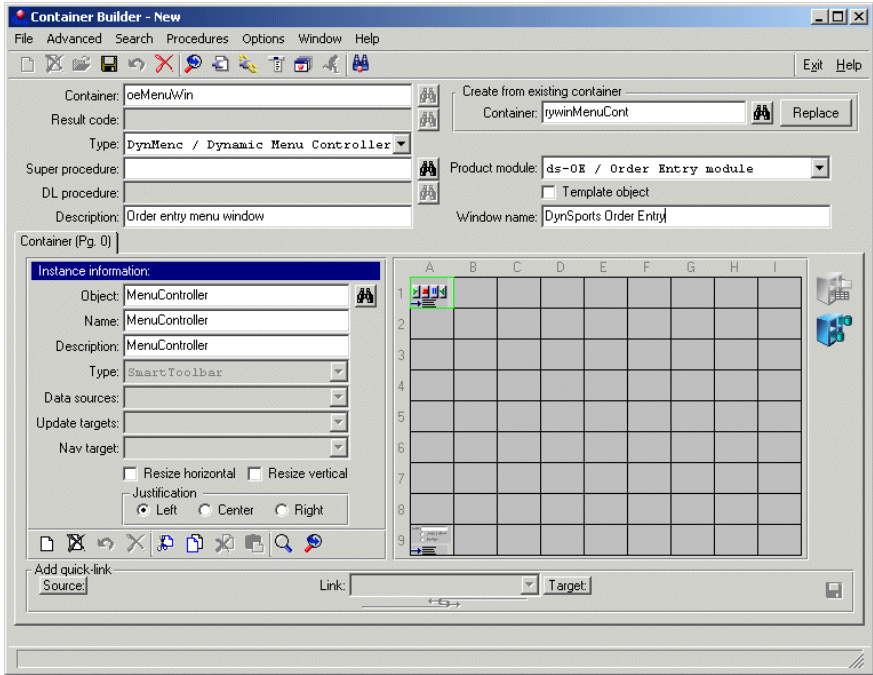
- 1 ♦ Choose **Build**→**Container Builder** from the AppBuilder main window.
- 2 ♦ Choose **New** in the Container Builder.
- 3 ♦ Type **rywinMenuCont** for the **Container** in the **Create from existing container** area and press the **TAB** key. This is the framework's default, and only built-in, template for a menu controller window. However, you can build another template for your applications if you want.

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 ♦ Choose **Create**.
- 5 ♦ Set the values in the Container Builder as shown in the following table:

Field	Value
Container	oeMenuWin
Type	DynMenc / Dynamic Menu Controller
Description	Order entry menu window
Product module	ds-OE / Order Entry module
Window name	DynSports Order Entry

The Container Builder should look like the following:



6 ♦ Choose **Save**, then exit the Container Builder.

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

4.6 Using the Toolbar and Menu Designer

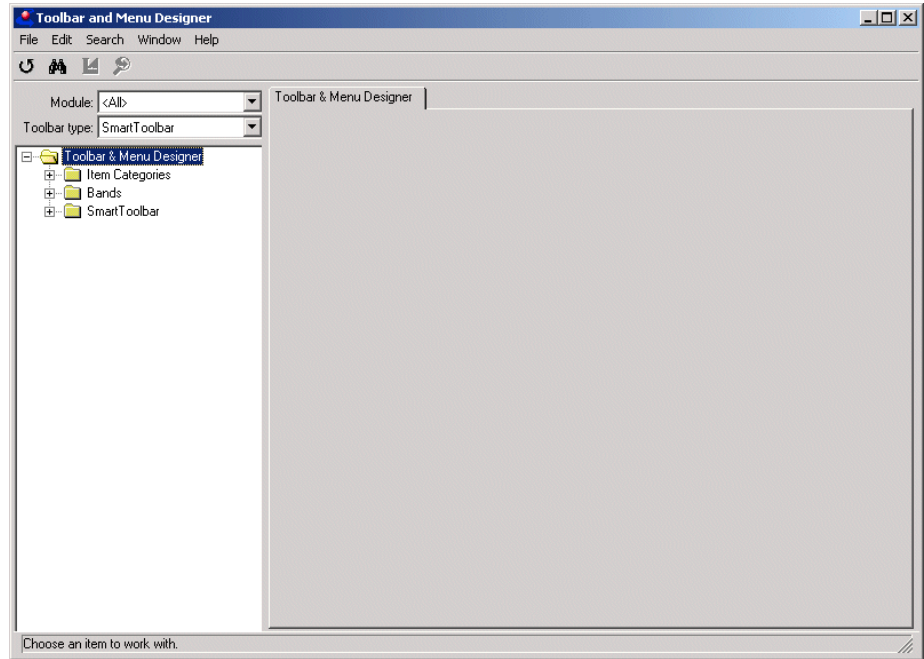
In this section, you use the Toolbar and Menu Designer. You can use this tool to construct a toolbar. You create the toolbar buttons and menu items you need for your application and assign each of them a specific function. For the sample application, the template menu window provides most of what you need. You need to 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 need to 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.

4.6.1 Creating item categories

To build the toolbar:

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

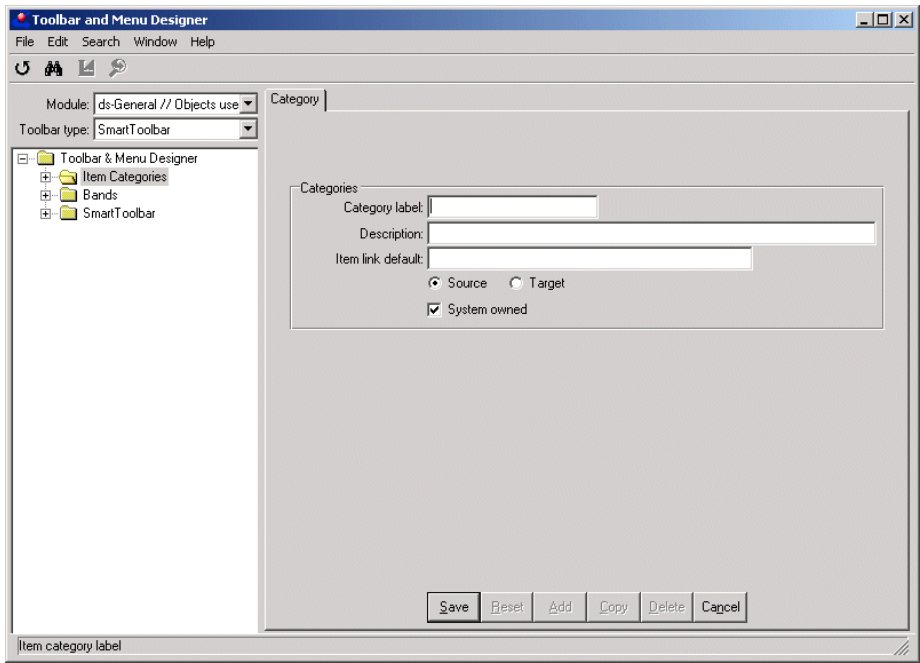


For this sample application, you use the Toolbar and Menu Designer to define some menu items and insert them into your menu window. You can browse a bit through the toolbar buttons and menu items that are part of the framework tools themselves by expanding the Item Categories, Bands, and SmartToolbar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

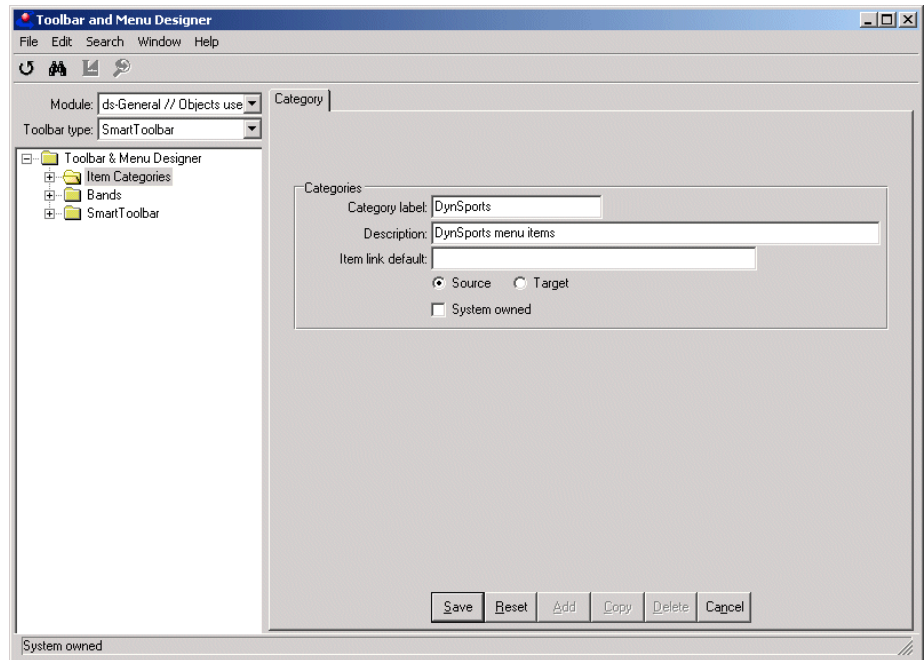
- 3 ♦ Right-click the **Item Categories** node and choose **Add Category** from the pop-up menu that appears. A Category update page appears on the tab folder, as shown:



- 4 ♦ Set the values shown in the following table in the update page. Leave the other settings at their default values:

Field	Value
Category label	DynSports
Description	DynSports menu items
System owned	Deselected

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:



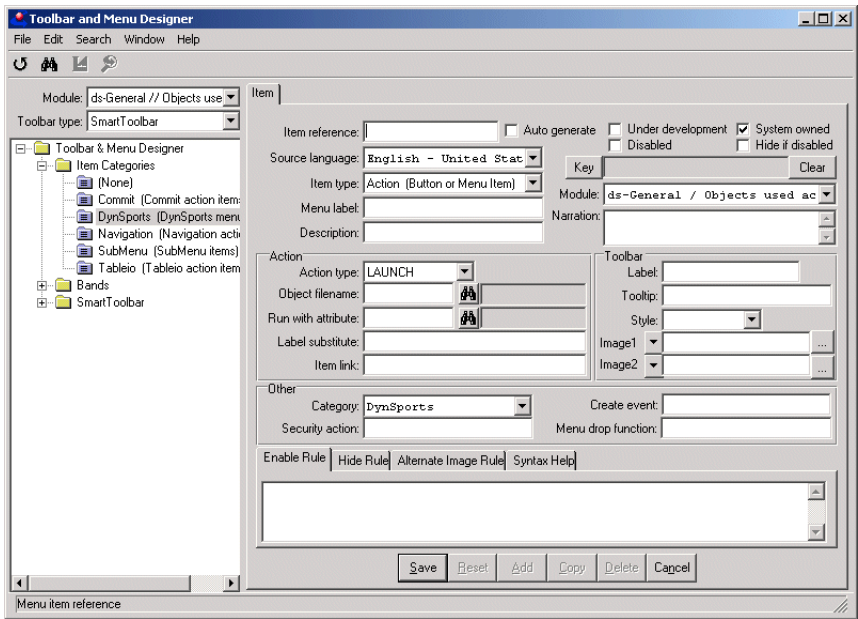
The update page is just a tab folder window like the ones you created. The dynamic windows you have built are realized at run time by one procedure (rydyncontw.w). This procedure reads all the related records out of the Repository and builds a dynamic window based on them. The TreeView layout has its 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 ♦ Choose **Save**.

4.6.2 **Creating menu items**

To create menu items in the DynSports category:

- 1 ♦ Right-click the **DynSports** node that you created.
- 2 ♦ Choose **Add Item** on the pop-up menu. An Item update page appears on the folder:



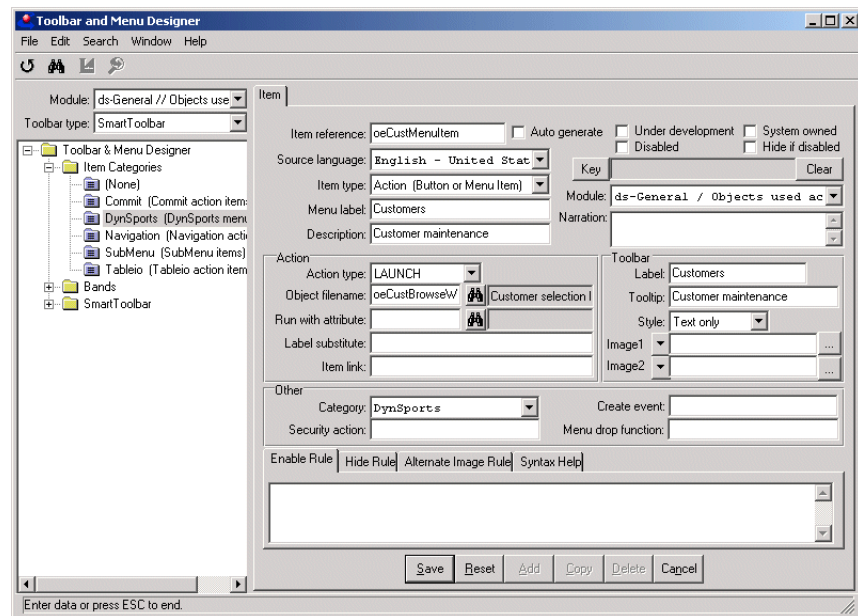
- 3 ♦ Set the values shown in the following table in the update page. Leave the other settings at their default values:

Field	Value
Item reference	oeCustMenuItem
Menu label	Customers
Description	Customer maintenance
System owned	Deselected
Tooltip	Customer maintenance
Style	Text only

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** for the **Object filename** in the **Action** area and press the **TAB** key:



- 5 ♦ Choose **Save**.
- 6 ♦ Right-click the **DynSports** node, and choose **Add Item** on the pop-up menu.

- 7 ♦ Define a menu item for the Order Maintenance window, using the values shown in the following table:

Field	Value
Item reference	oeOrderMenuItem
Menu label	Orders
Description	Order maintenance
System owned	Deselected
Tooltip	Order maintenance
Style	Text only
Item type	Action
Action type	Launch
Object filename	oeOrderBrowseWin

The Item page should look like the following:

The screenshot shows the 'Toolbar and Menu Designer' application window. The 'Item' tab is active, displaying configuration fields for a menu item. The 'Module' is set to 'ds-General // Objects used ac'. The 'Item reference' is 'oeOrderMenuItem'. The 'Source language' is 'English - United States'. The 'Item type' is 'Action (Button or Menu Item)'. The 'Menu label' is 'Orders'. The 'Description' is 'Order maintenance'. The 'Action type' is 'LAUNCH'. The 'Object filename' is 'oeOrderBrowse'. The 'Run with attribute' is 'Order selection brow'. The 'Label substitute' is empty. The 'Item link' is empty. The 'Toolbar' section shows 'Label: Orders', 'Tooltip: Order maintenance', and 'Style: Text only'. The 'Other' section shows 'Category: DynSports', 'Security action' is empty, 'Create event' is empty, and 'Menu drop function' is empty. The 'Enable Rule' tab is selected. The 'Save', 'Reset', 'Add', 'Copy', 'Delete', and 'Cancel' buttons are at the bottom.

- 8 ♦ **Save** your changes.

4.6.3 Creating a submenu label

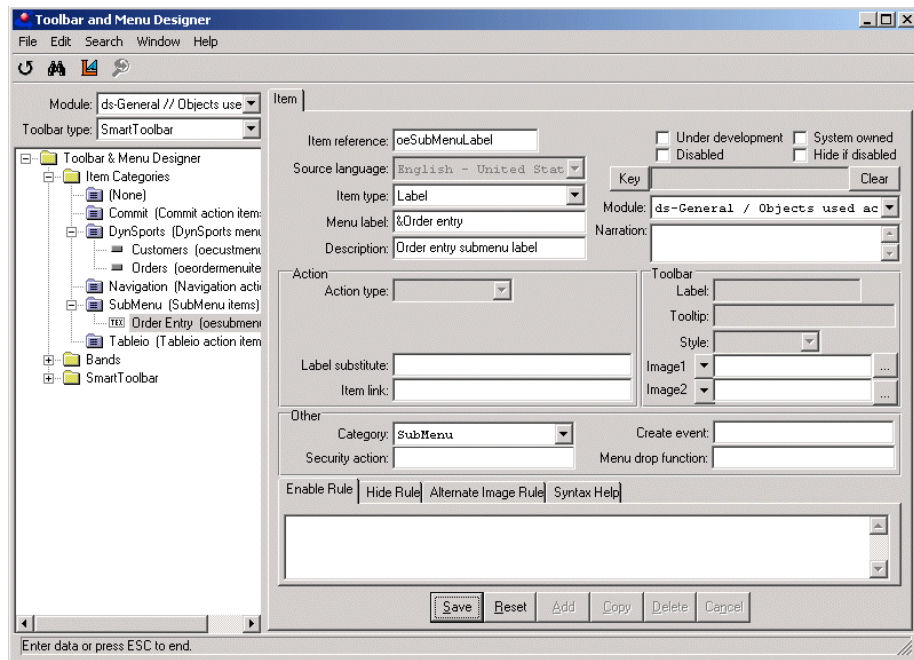
To create a Submenu item to group your new items together:

- 1 ♦ Right-click the **SubMenu** node under the **Item Categories** node, and choose **Add item** on the pop-up menu.
- 2 ♦ Set the values shown in the following table in the update page. Leave the other settings at their default values:

Field	Value
Item reference	oeSubMenuLabel
Item type	Label
Menu label	&Order entry
Description	Order entry submenu label
System owned	Deselected

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 ♦ Choose **Save**. The Toolbar and Menu Designer should look like the following:



4.6.4 Creating a menu band

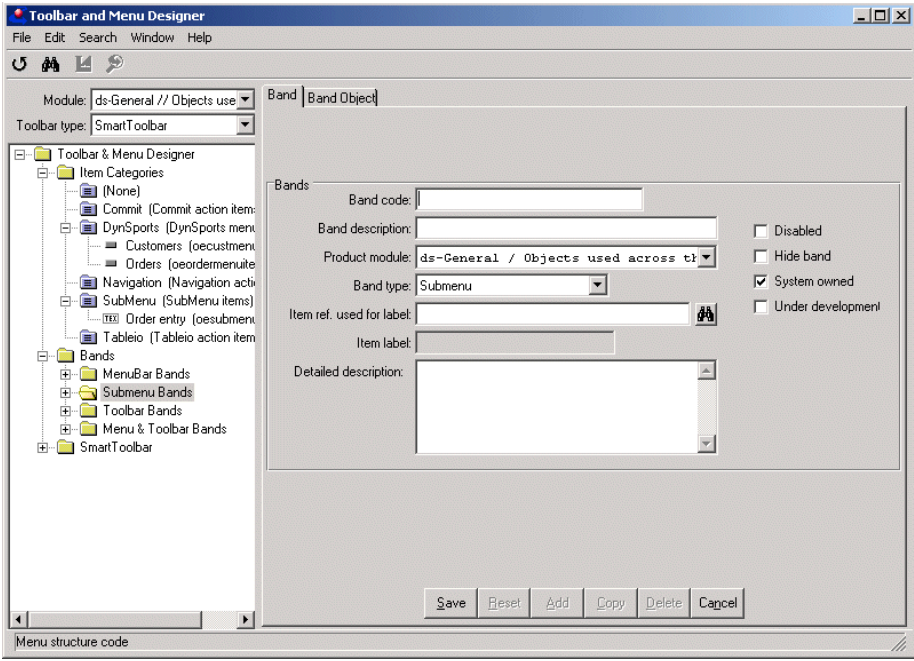
Next you need to group the actions together into a *band*. A band is a set of related actions. You can visualize a band as a submenu (all the items that pop up when you select a top-level menu item) or a set of buttons that appear together in a toolbar. There are four types of bands:

- **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 Progress SmartToolbars™.

To create a band:

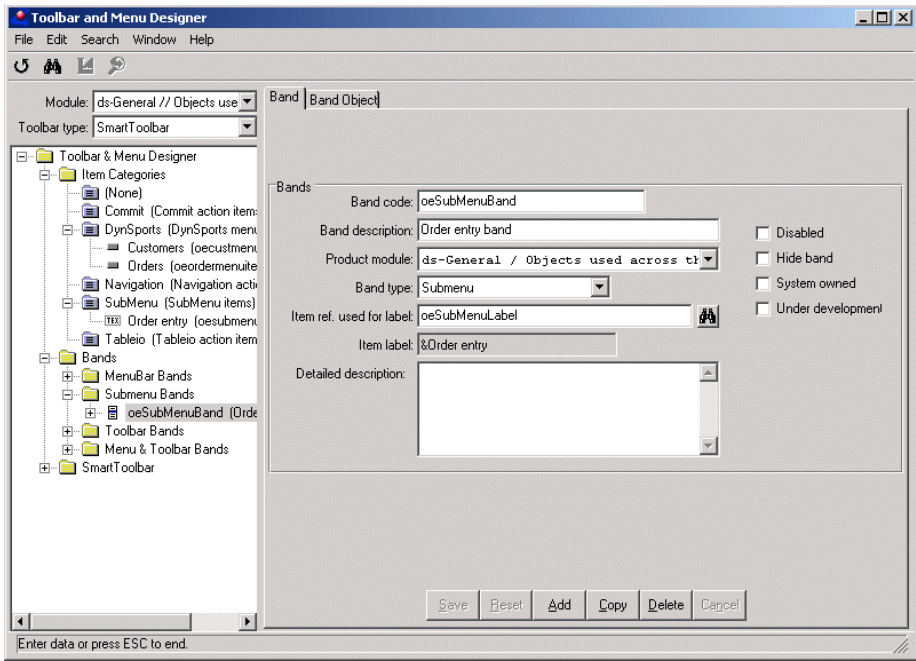
- 1 ♦ Expand the **Bands** node to see the four band types.
- 2 ♦ Right-click on the **Submenu Bands** node, and choose **Add Band** from the pop-up menu. Note that two update pages are created: **Band** and **Band Object**:



- 3 ♦ Set the values shown in the following table on the **Band** update page:

Field	Value
Band code	oeSubMenuBand
Description	Order entry band
Band type	Submenu
Item reference used for label	oeSubMenuLabel
System owned	Deselected

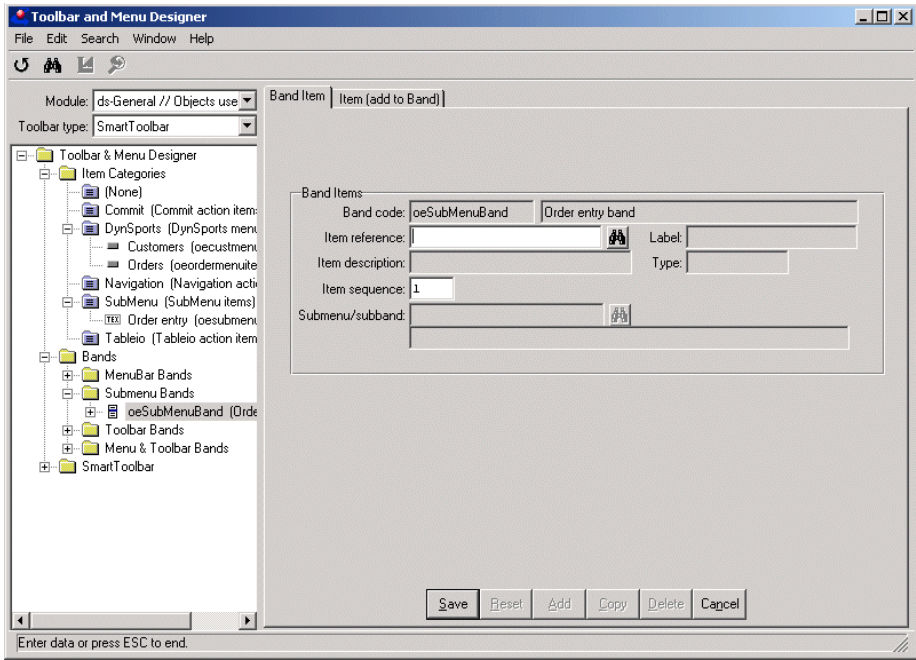
4 ♦ Choose **Save**. The Toolbar and Menu Designer should now look like the following:



4.6.5 Adding band items

To add your items to the band:

- 1 ♦ Right-click on the **oeSubMenuBand** node, and choose **Add Item to Band** from the pop-up menu. Note that two update pages are displayed: **Band item** and **Item (add to Band)**:



- 2 ♦ Set the values shown in the following table on the **Band Item** update page:

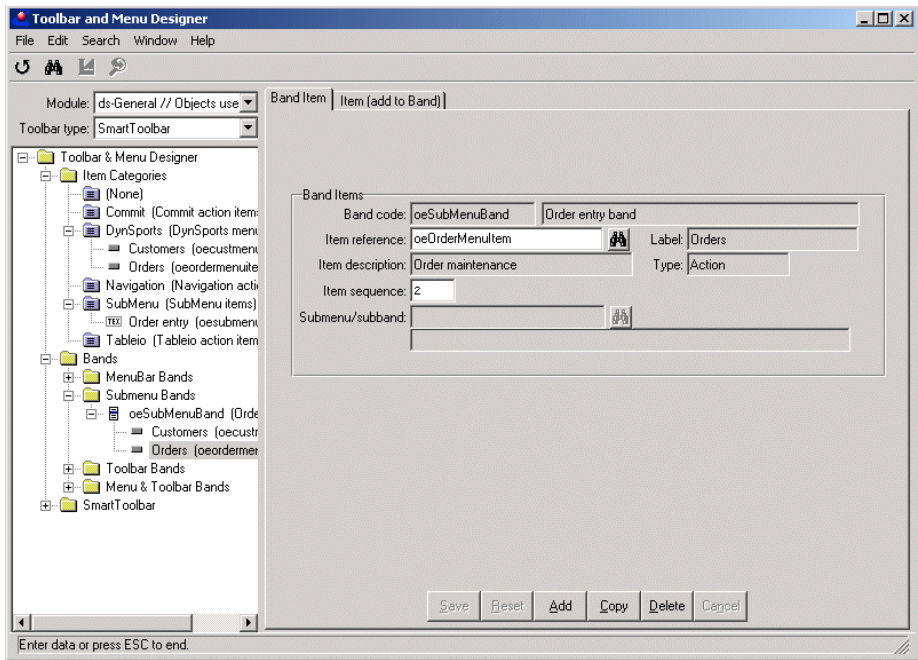
Field	Value
Item reference	oeCustMenuItem
Item sequence	1

- 3 ♦ Choose **Save**.

- 4 ♦ Add the Order menu item using the values shown in the following table:

Field	Value
Item reference	oeOrderMenuItem
Item sequence	2

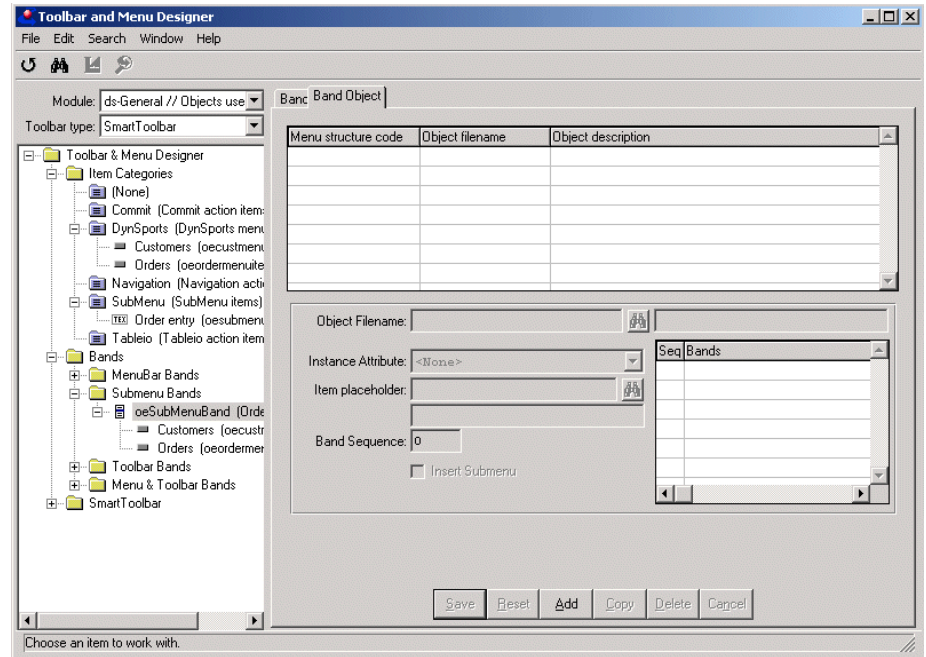
- 5 ♦ Choose **Save**. The Toolbar and Menu Designer should now look like the following:



4.6.6 Adding 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:

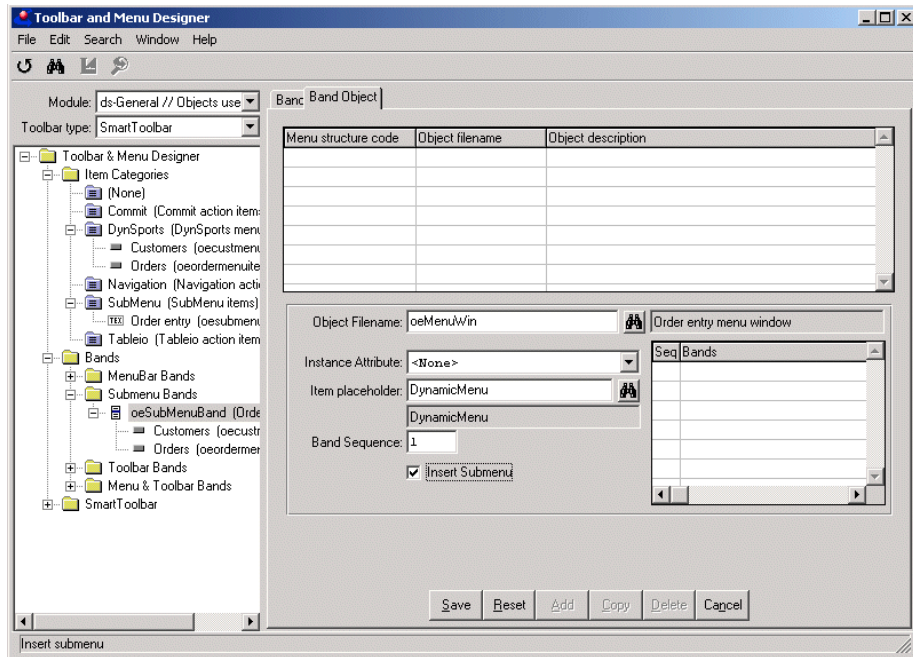


- 2 ♦ Choose the **Add** button.
- 3 ♦ Type **oeMenuWin** for the **Object Filename** and press the **TAB** key.

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 like the following:



- 6 ♦ Choose **Save**, then close the Toolbar and Menu Designer.

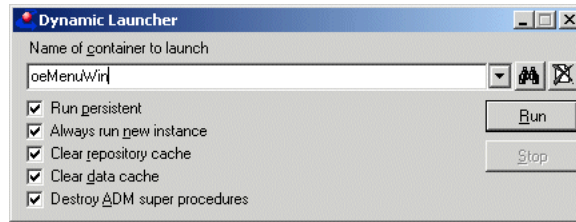
4.7 Running the completed application

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 need to select the **Destroy ADM Super Procedures** toggle box 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:

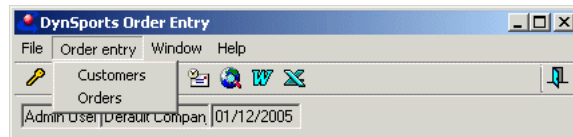
- 1 ♦ Choose **Compile→Dynamic Launcher** from the AppBuilder main window.
- 2 ♦ Type **oeMenuWin** in the Dynamic Launcher.

- 3 ♦ Select the **Destroy ADM Super Procedures** toggle box, then choose **Run**:



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 need to 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:



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

When you run the Order Maintenance window, you can try out the dynamic Lookup you added to the Order Code field as follows:

- 1 ♦ Choose **OrderEntry**→**Orders**.
- 2 ♦ Double-click an order to launch the Order Maintenance window.
- 3 ♦ Choose the **Modify** button on the toolbar.
- 4 ♦ Choose the **Lookup** button for the **Customer** field.
- 5 ♦ 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 the 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.

This chapter discusses the following topics:

- [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](#)

5.1 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

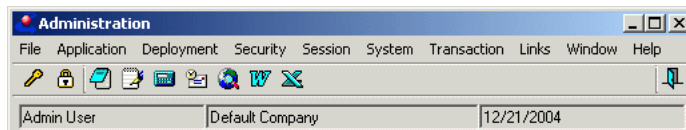
Now that you have built your sample application, you can take a brief look at a few of the capabilities of the Managers. You can begin with delivering translations for your application based on a user's login language.

5.1.1 Using the Localization Manager to translate a window

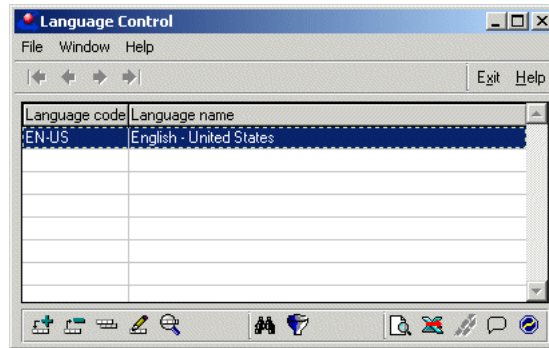
The Progress Dynamics *Localization Manager* lets you translate all visual elements of your application screens (including application messages) into any number of languages. You can also use the Language facility to provide alternative texts for any purpose, such as specializing labels, prompts, and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

To define a new language:

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

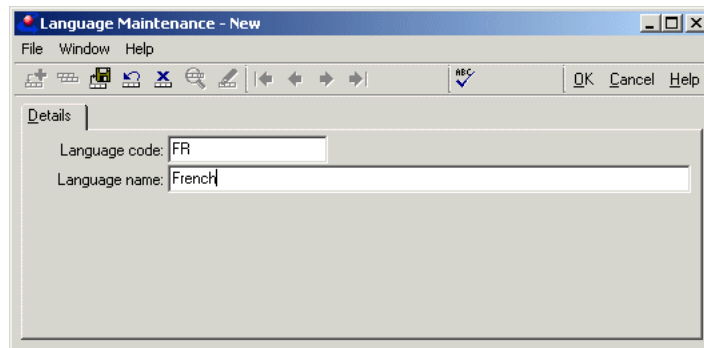


- 2 ♦ Choose **Application**→**Language Control**:



- 3 ♦ Choose the **Add** button . 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:



- 5 ♦ Choose **Save**, then exit the Language Control and Language Maintenance windows.
- 6 ♦ Close the Administration window.

Now that you have a code in the Repository for French, you can store French translations for your application. Then, you can create a user who logs in using the French language. 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 the [*Progress Dynamics Developer's Guide*](#).

Translating widgets

You can translate any application screen while it is running.

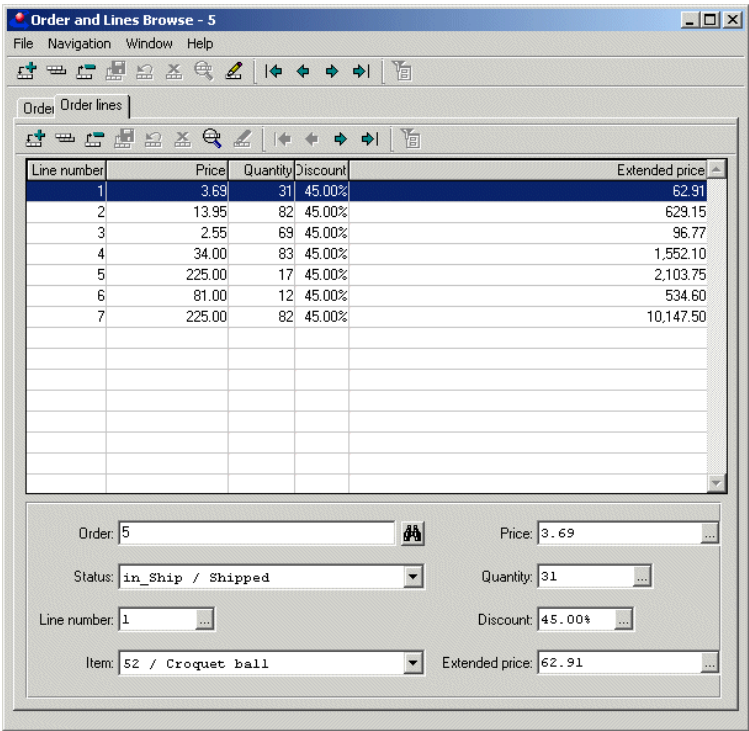
To translate a screen into French:

- 1 ♦ Run **oeMenuWin** from the Dynamic Launcher. Remember to select the **Destroy ADM Super Procedures** toggle box.

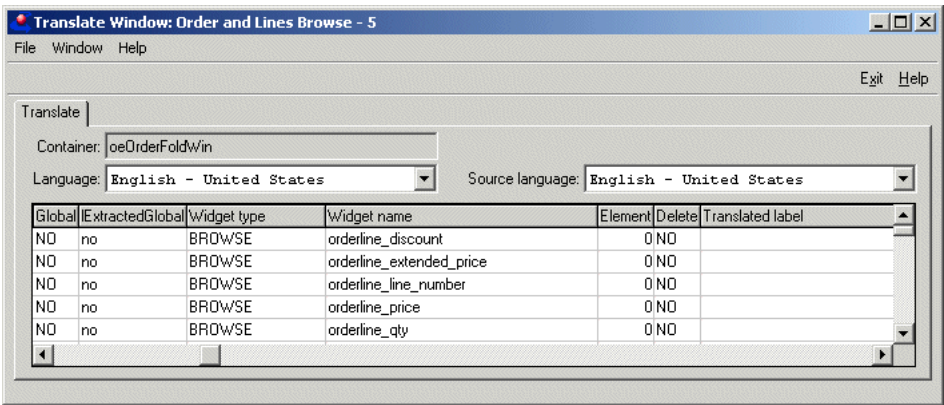
NOTE: Remember to reconnect the DynSports database if you have restarted your session since last running the application.

- 2 ♦ Choose **Order Entry→Orders**.
- 3 ♦ Double-click on an order to launch your Order Browse.

- 4 ♦ Select the **Order lines** tab, as shown:



- 5 ♦ Choose **File→Translate**. The Translate Window dialog box appears. The Localization Manager builds the browse with all the translatable strings in the current window, as shown:



- 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 the [Progress Dynamics Developer's Guide](#).
- 7 ♦ Select the **orderid_price** fill-in widget.
- 8 ♦ Enter **Prix ligne de commande:** in the **Translated label** column.
- 9 ♦ Enter the widget label translations shown in the following table:

Widget type	Widget name	Translated label
Fill-in	orderid_discount	Rabais ligne de commande:
Fill-in	orderid_extended_price	Prix prolonge ligne de commande:
Fill-in	orderid_qty	Quantite:
Tab	Order	Ordre
Tab	Order lines	Ligne de commandes

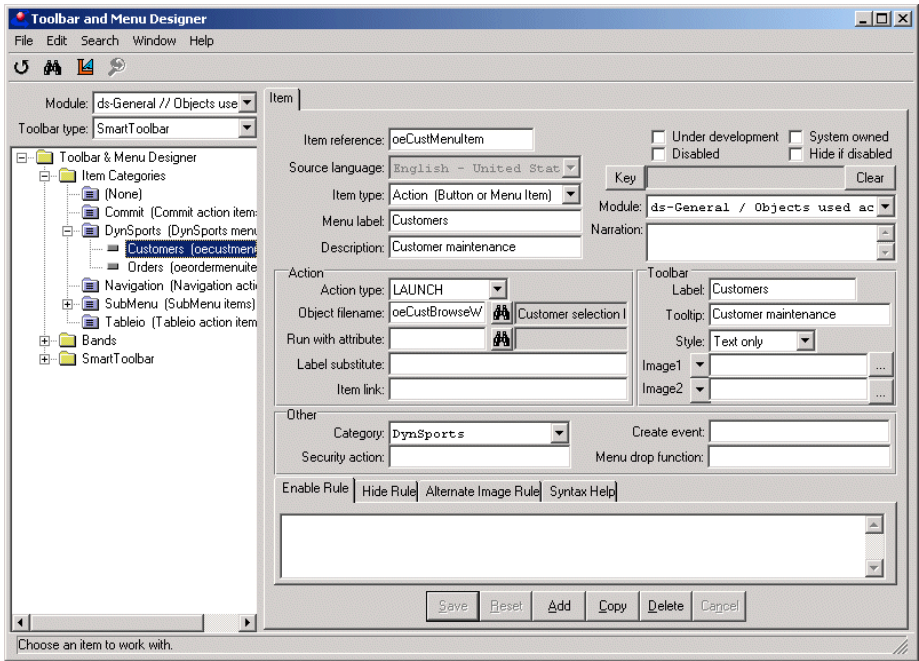
- 10 ♦ Choose **OK**, and close your application windows.

Translating menus

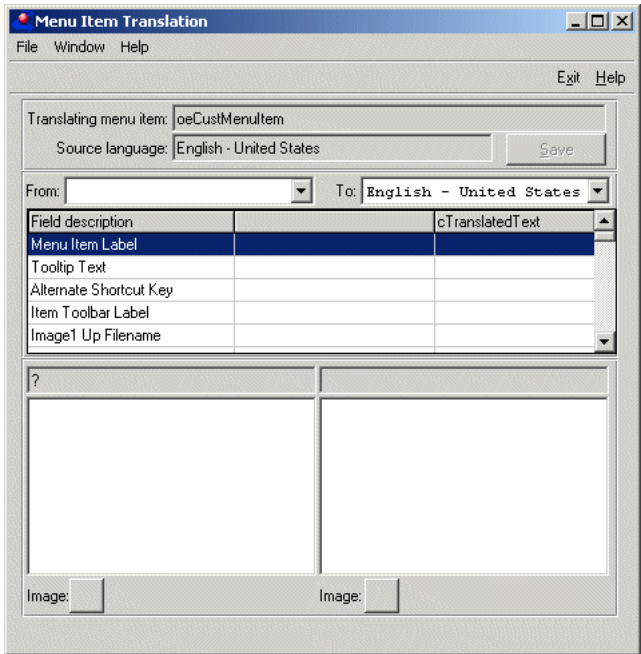
To translate a menu into French:

- 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 ♦ Choose the **Translate Menu Item** button . 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 **Clients** in the **French** column and press **TAB**.
- 8 ♦ Choose **OK**.
- 9 ♦ Add the translations shown in the following table:

Menu element	Translation
Item Categories→DynSports→Orders	Commandes
Item Categories→SubMenu→Order entry	Saisie de commandes

- 10 ♦ Exit the Toolbar and Menu Designer.

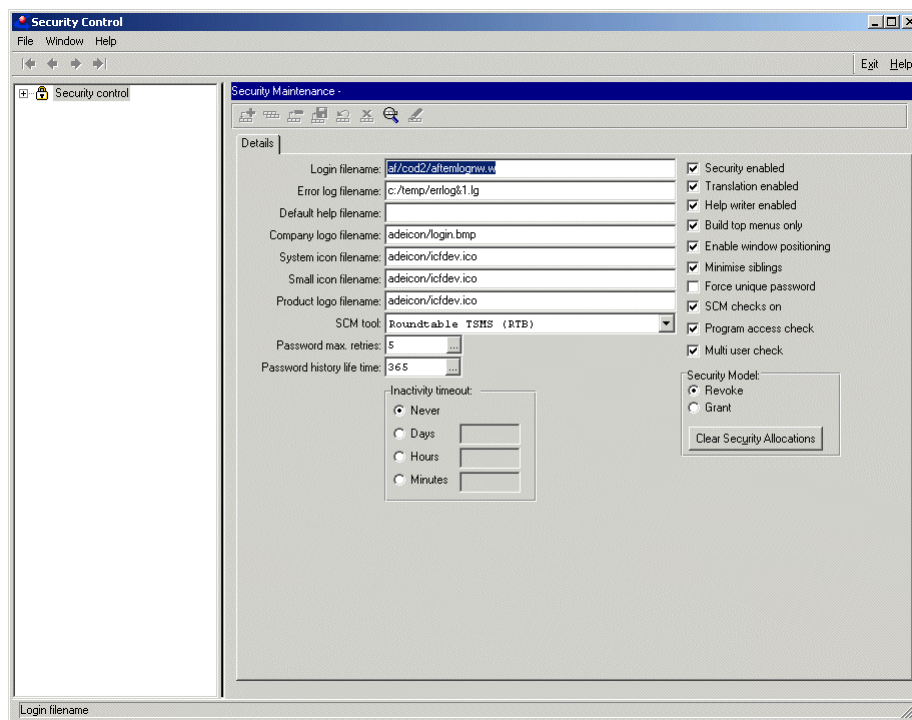
Before you can see the effects of your translations, you need to log on as a French user. So, the next step is to create a French user.

5.1.2 Creating a new user

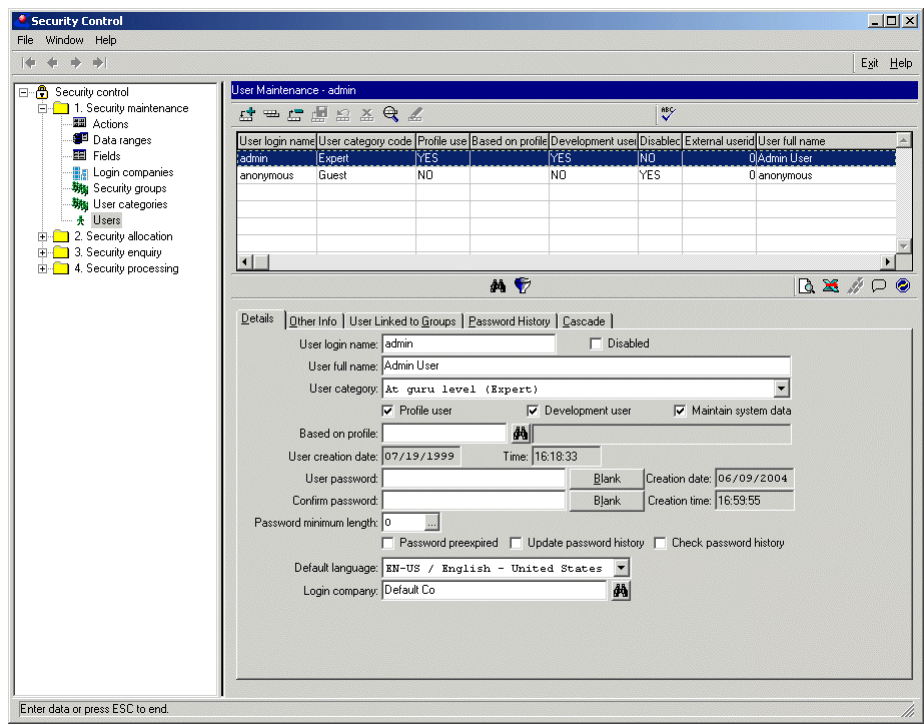
You create users in the Security Control TreeView, which uses data maintained by the Profile Manager.


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:

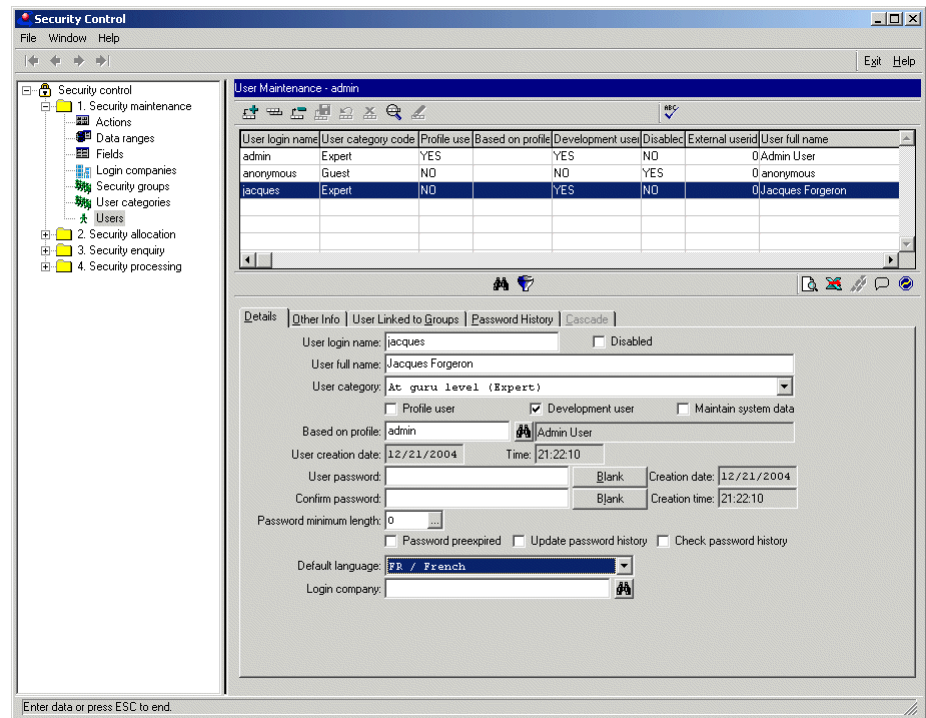


3 ♦ Choose the **Add** button  from the User Maintenance toolbar.

4 ♦ Set the values shown in the following table for your new user:

Field	Value
User login name	jacques
Full name	Jacques Forgeron
User category	At guru level (Expert)
Development User	selected
Based on profile	admin
Default language	FR / French

5 ♦ Choose **Save**, then exit the Security Control TreeView:



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 ♦ Enter **jacques** as the login name in the Application login window.
- 4 ♦ Reconnect the DynSports database.
- 5 ♦ Launch **oeMenuWin** from the Dynamic Launcher.

NOTE: If you receive the “ADM super procedures in use” message, you need to 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:

The screenshot shows the 'Order and Lines Browse - 5' window. It has a menu bar (File, Navigation, Window, Help) and a toolbar. The 'Ligne de commandes' tab is selected. Below the toolbar is a table with the following data:

Line number	Price	Quantity	Discount	Extended price
1	3.69	31	45.00%	62.91
2	13.95	82	45.00%	629.15
3	2.55	69	45.00%	96.77
4	34.00	83	45.00%	1,552.10
5	225.00	17	45.00%	2,103.75
6	81.00	12	45.00%	534.60
7	225.00	82	45.00%	10,147.50

Below the table is a summary section with the following fields:

- Order: 5
- Status: in_Ship / Shipped
- Line number: 1
- Item: 52 / Croquet ball
- Prix ligne de commande: 3.69
- Quantite: 31
- Rabais ligne de commande: 45.00%
- Prix prolonge ligne de commande: 62.91

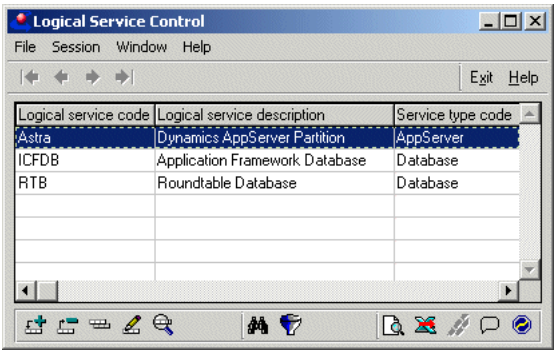
- 9 ♦ Close everything and restart your session as the **admin** user.

5.1.3 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:

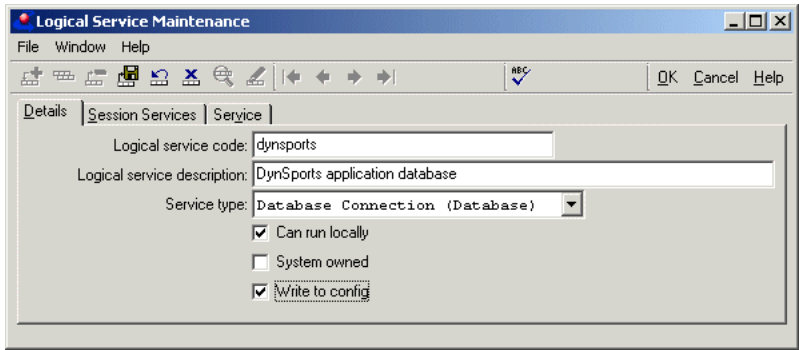


- 2 ♦ Choose **Add** to add a new service for the DynSports database. The Logical Service Maintenance window appears.
- 3 ♦ Set the values shown in the following table for the new service:

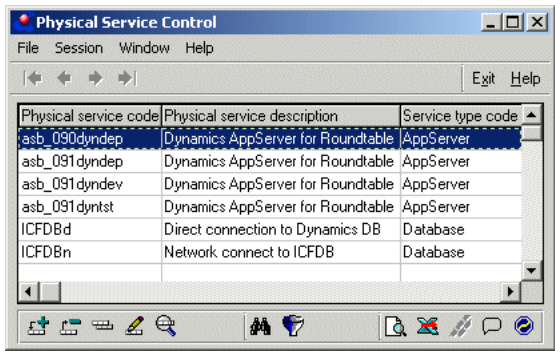
Field	Value
Logical service code	dynsports
Logical service description	DynSports application database
Service type	Database Connection (Database)
Can run locally	Selected
System owned	Not selected
Write to config	Selected

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 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:



- 4 ♦ Choose **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:

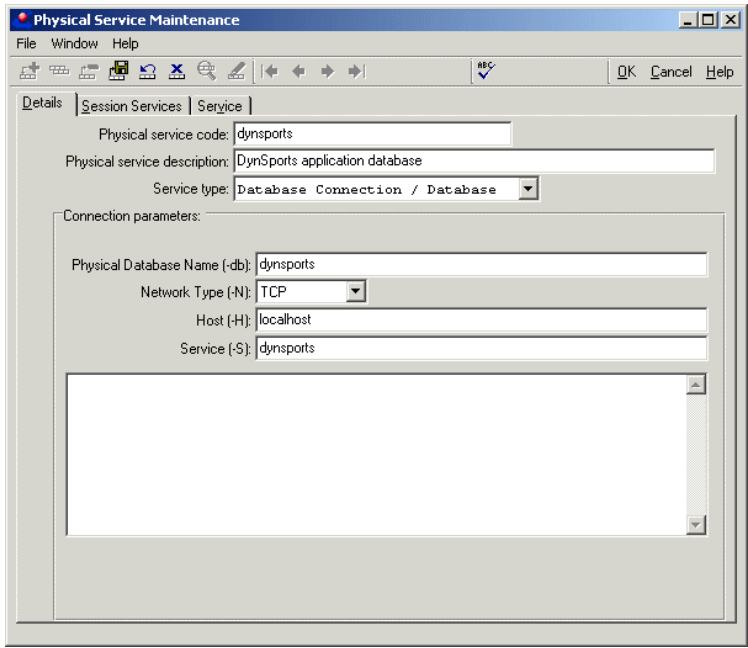


- 6 ♦ Choose **Add** to add a new service for the local DynSports database connection. The **Physical Service Maintenance** dialog box appears.

7 ♦ Set the values shown in the following table for the new service:

Field	Value
Physical service code	dynsports
Physical service description	DynSports application database
Service type	Database Connection / Database
Physical Database Name (-db)	dynsports
Network Type (-N)	TCP
Host (-H)	localhost
Service (-S)	dynsports

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



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 in [Chapter 2, “Setting Up a Tutorial Environment”](#).

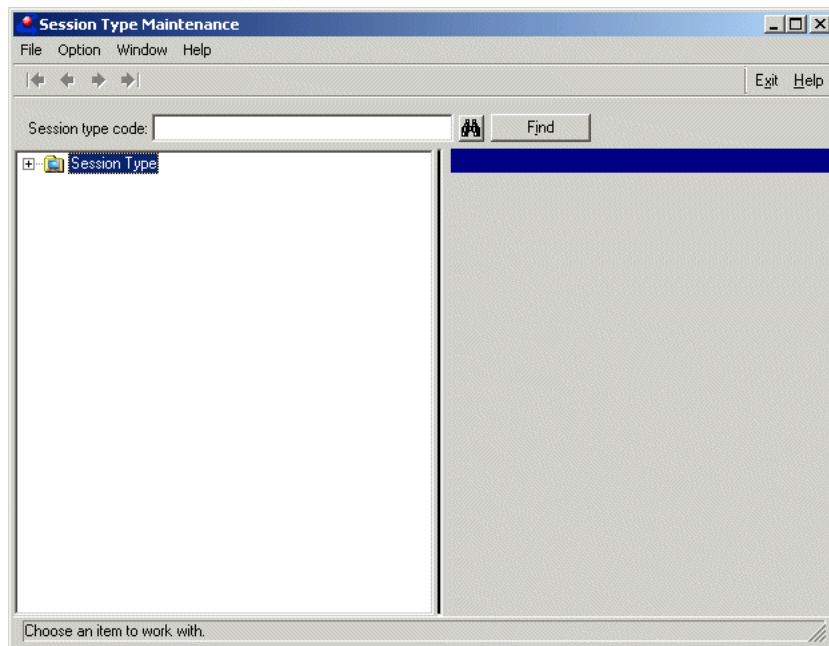
- 8 ♦ Choose **Save**, then exit the maintenance and control dialog boxes.

5.1.4 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.

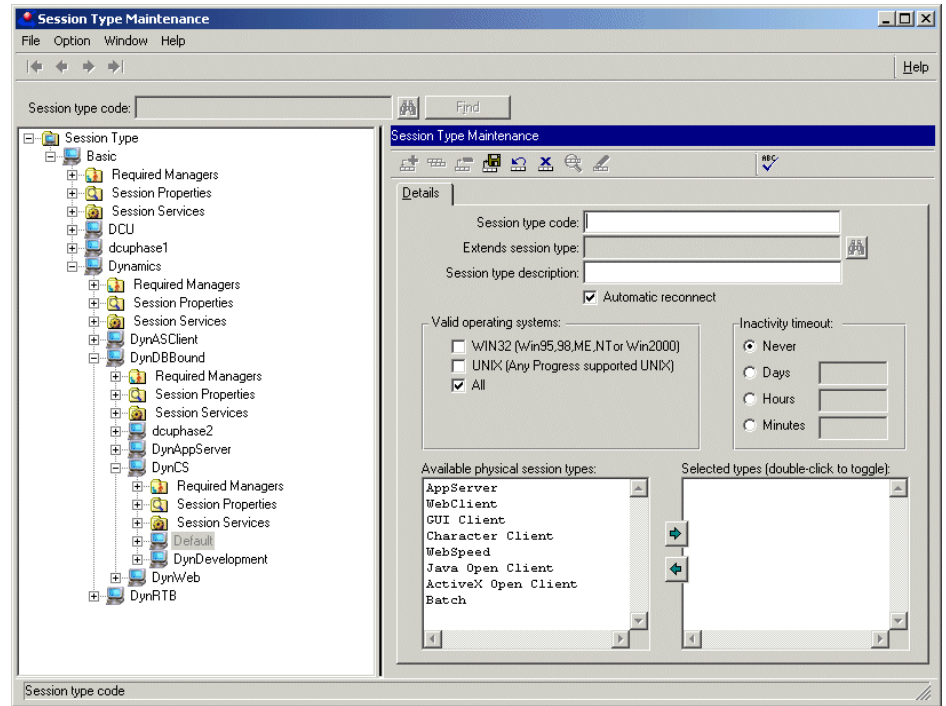
To edit the Default session type:


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



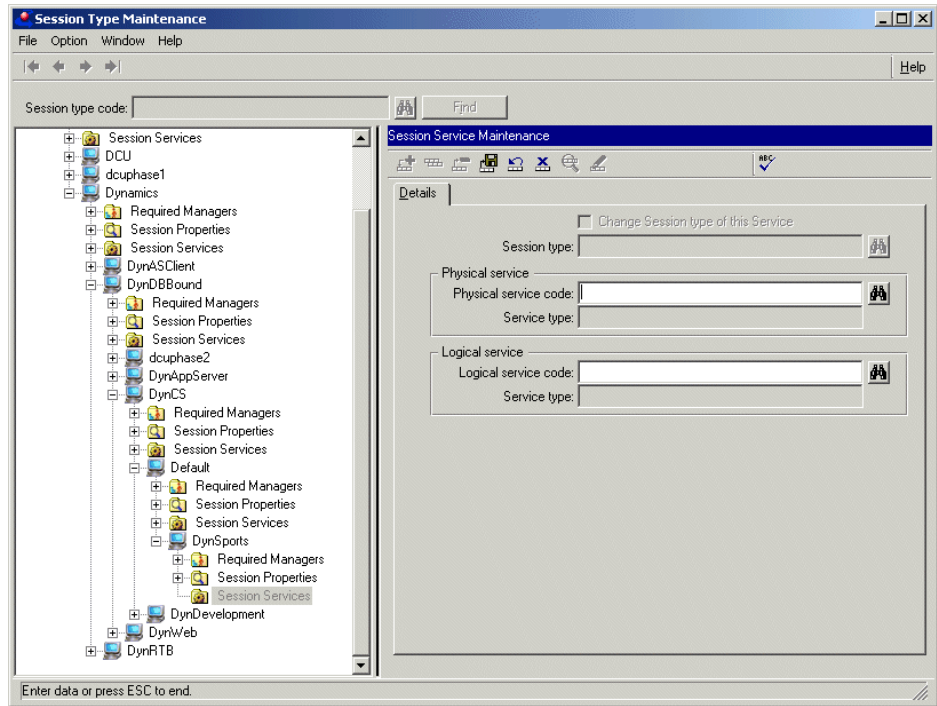
- 2 ♦ Choose the **Session Type**→**Basic**→**Dynamics**→**DynDBBound**→**DynCS** node.

- 3 ♦ Right-click the **Default** node and choose **Add Session Type** from the pop-up menu. A Details update page appears:



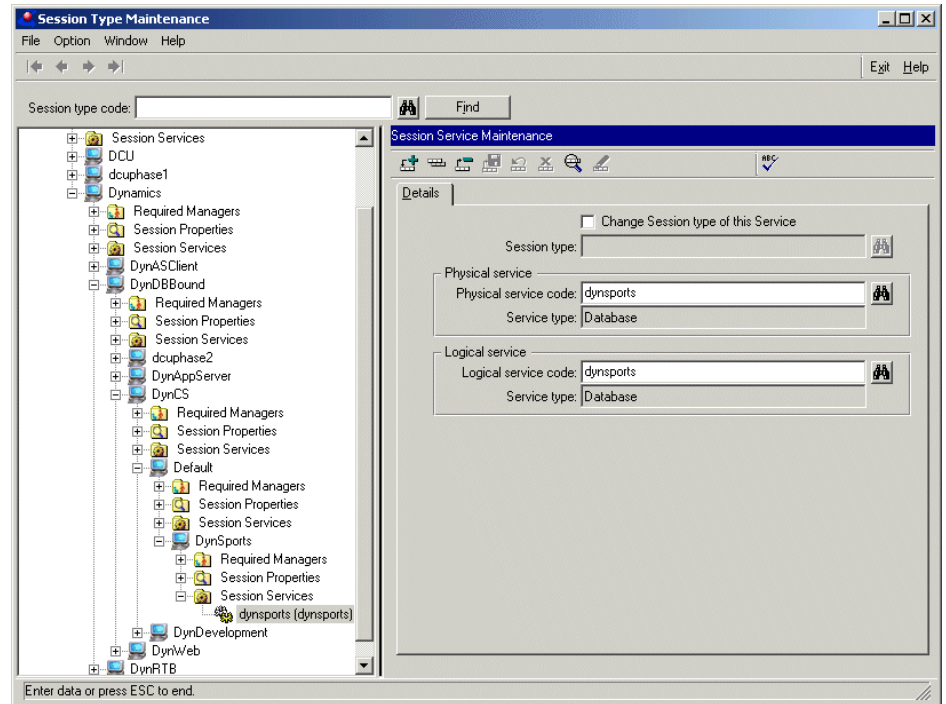
- 4 ♦ Type **DynSports** for the **Session type code** and **DynSports sample application** for the **Session type description**.
- 5 ♦ Select the **WIN32** toggle box in the **Valid Operating Systems** group.
- 6 ♦ Select **GUI Client** from the **Available physical session types** list.
- 7 ♦ Choose the **Add to selected fields** button  to move it to the **Selected types** list.
- 8 ♦ Choose **Save**.
- 9 ♦ Expand the **DynSports** node.

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

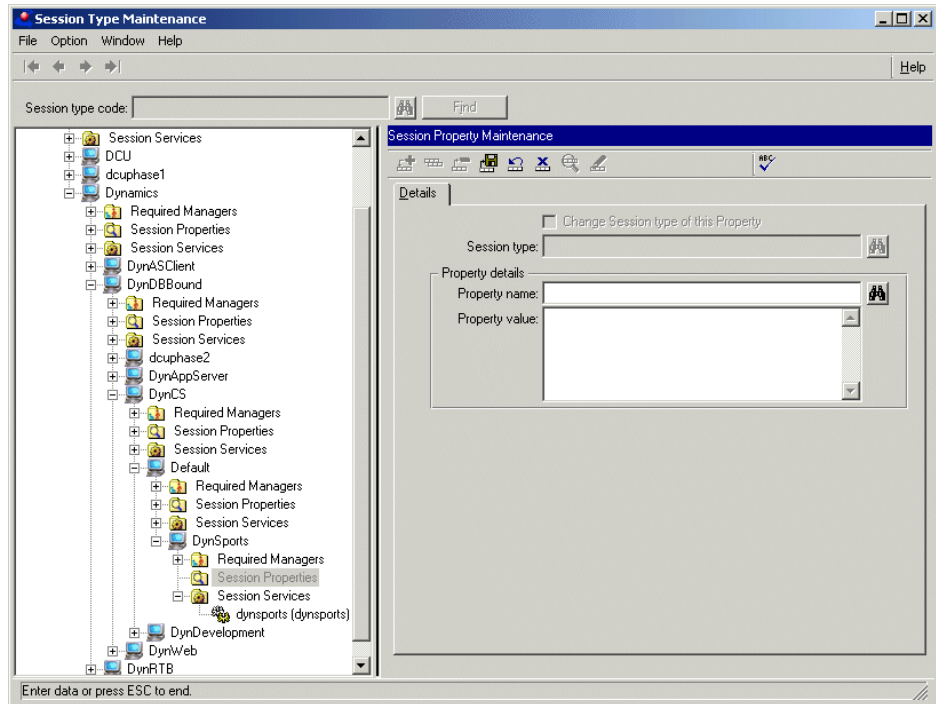


- 11 ♦ Type **dynsports** for the **Physical service code** and press the **TAB** key.
- 12 ♦ Type **dynsports** for the **Logical service code** and press the **TAB** key.

- 13 ♦ Choose **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:

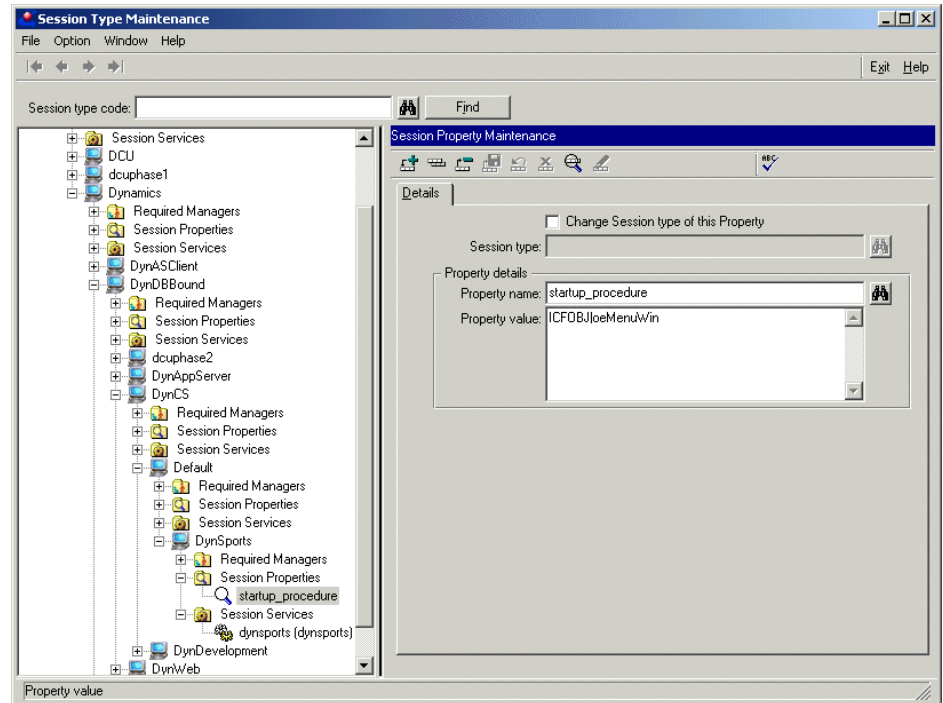


This page shows the logical properties that are defined for this session type. However, because your new DynSports session type extends the Default session type, it inherits properties from the Default session type and its parent session types.

The `run_local` property indicates that the session can run stand-alone, as you want it to do. The `startup_procedure` property defines which Progress 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 **ICFOBJ|oeMenuWin** as the **Property value**, and choose **Save**:



NOTE: The delimiter character is a pipe (|).

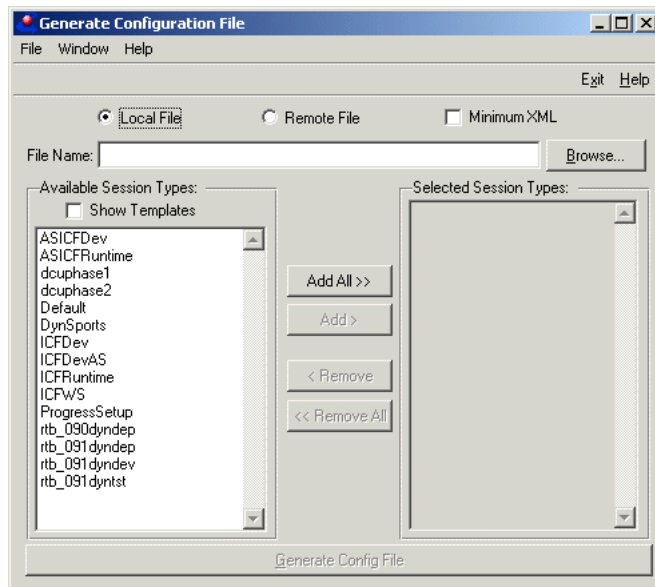
5.2 Creating a configuration XML file for your application

In the “[Creating configuration files](#)” section in [Chapter 2, “Setting Up a Tutorial Environment,”](#) 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.

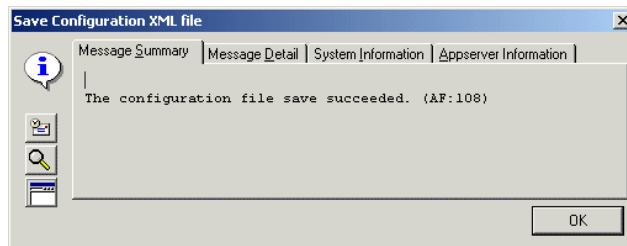
The DynSports session type that you created currently exists only in the Repository. To use it, you need to export this information to a configuration file like `icfconfig.xml`. Once the session type is listed in the configuration file, you can create a Desktop shortcut that uses the information to launch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

To regenerate the configuration file:

- 1 ♦ Choose **Option**→**Generate Configuration File** from the Session Type Maintenance window's menu. The Generate Configuration XML File dialog box appears:



- 2 ♦ Type `<wrk>\Tutorial\dynsports.xml` for the **File Name**, where `<wrk>` is your Progress Dynamics working directory.
- 3 ♦ Select **DynSports** from the **Available Session Types** list and choose **Add>** to move the DynSports session type into the **Selected Session Types** list. The generation process adds only the selected session types into the file, allowing you to tailor what can be accessed from a deployed application. If the configuration file does not have the session types to support the development environment, it cannot be started using that configuration file.
- 4 ♦ Choose the **Generate Config File** button to create your custom configuration file. A message dialog box appears when the file is completed:



- 5 ♦ Close **OK**, and exit the Generate Configuration File and Session Type Maintenance windows.
- 6 ♦ Open the `dynsports.xml` file in a Web browser, and search for the DynSports session type to see the results of your changes, as shown:

```

- <session>
- <sessionSessionType="DynSports">
- <properties>
  <auto_dump_entity_cache>YES</auto_dump_entity_cache>
  <bound_icfdb>yes</bound_icfdb>
  <DynamicsVersion>2.1B</DynamicsVersion>
  <ICFCM_AppServer>AppServerConnectionManager</ICFCM_AppServer>
  <ICFCM_Database>DatabaseConnectionManager</ICFCM_Database>
  <login_procedure>af/cod2/afitemlognw.w</login_procedure>
  <physical_session_list>GUI</physical_session_list>
  <print_preview_preference>
  <print_preview_stylesheet>af/rep/xmlreport.xml</print_preview_stylesheet>
  <root_directory></root_directory>
  <run_local>YES</run_local>
  <session_date_format>mdy</session_date_format>
  <session_year_offset>1950</session_year_offset>
  <startup_procedure>ICFOBJJoeMenuWin</startup_procedure>
  <UseThinRendering>YES</UseThinRendering>
  <valid_os_list>WIN32</valid_os_list>
</properties>
- <services>
+<service></service>
+<service></service>
- <service>
  <cServiceType>Database</cServiceType>
  <cServiceName>dynsports</cServiceName>
  <cPhysicalService>dynsports</cPhysicalService>
  <cConnectParams>-db dynsports -N TCP -H localhost -S dynsports</cConnectParams>
  <IDefaultService>no</IDefaultService>
  <ICanRunLocally>yes</ICanRunLocally>
</service>
</services>
+<managers></managers>
</session>
</session>

```

5.3 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 the shortcut:

- 1 ♦ Open the **Progress Dynamics Tutorial** folder, where you created the shortcuts in the “Creating startup scripts and shortcuts” section in [Chapter 2, “Setting Up a Tutorial Environment.”](#)
- 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**.
- 8 ♦ Choose **OK** to save your changes.

5.4 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 Progress Dynamics AppBuilder, but do not stop the database servers.

This lets you see that the application runs without the Progress Dynamics Development environment. 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**.

5.5 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, Progress WebSpeed®, the new Managers, and using CSS files and JavaScript, see the [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.

5.6 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 the *Progress Dynamics Developer's Guide*, 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 (1 of 2)

For information about . . .	Go to . . .
Installing and viewing the Progress Dynamics online documentation.	<i>Progress Dynamics Installation Guide</i>
The books that make up the documentation set.	<i>Welcome to Progress OpenEdge Studio</i>
The differences between the Version 2.1A release and earlier releases.	<i>Progress Dynamics Product Update Bulletin</i>
Advanced programming topics (including examples).	<i>Progress Dynamics Programming Handbook</i>
Progress SmartObject procedures, functions, and properties.	<i>Progress Dynamics ADM2 API Reference</i>
Developing applications using the Progress Dynamics framework.	<i>Progress Dynamics Developer's Guide</i>

Table 5–1: Where to go from here

(2 of 2)

For information about . . .	Go to . . .
Developing and deploying Web browser-based Progress Dynamics applications.	<i>Progress Dynamics Web Development Guide</i>
Configuring and managing Progress Dynamics.	<i>Progress Dynamics Administration Guide</i>
Common administrative procedures that you might need to perform to support all of the applications that you develop using Progress Dynamics.	<i>Progress Dynamics Administration Guide</i>
Integration with optional third-party tools, such as ERwin.	<i>Progress Dynamics Administration Guide</i>
Progress Dynamics managers.	<i>Progress Dynamics Managers API Reference</i>
The structure of the Progress Dynamics Repository database	<i>Progress Dynamics Repository Reference</i>

Index

A

- Action Type option 4–77
- ActiveX 4–72
- ADE. *See also* Application Development Environment (ADE) 1–5
- Administration window 3–6
- AppBuilder 1–5
 - Progress Dynamics 1–5
- Application
 - running 5–24
 - standard and reusable components 1–6
- Application Development Environment (ADE) 1–5
- Application objects
 - generating 3–20
- Architecture 1–2

B

- Band
 - defined 4–80
- Base Query String 4–8, 4–14

- Bottom of grid 4–60
- Browse sequence
 - setting 4–15
- Browse windows
 - creating 4–30
- Browsers 3–20
- Built-in
 - framework managers 1–7

C

- Cache
 - clearing 4–40
- Calculator 4–59
- Can Run Locally option 5–13, 5–15
- Categories 4–72
- Clear cache option 4–40
- Company
 - login 2–18
- Completed application 4–2
- Configuration Manager 5–12
- Configuration XML file 5–21

Connection Manager 5–12

Container Builder 4–11

Containers 4–32

Control objects 1–7

csunfoldwin 4–43

Custom super procedure 3–28

customerfullb 4–33

D

Data logic procedure
defined 1–6

Data objects 1–6
defined 1–6

Databases
application 3–4
connecting 3–4
DynSports 2–12
repository 1–4

Deleting objects 4–46

Dependent windows 4–43

Design window
Browse 3–28
SDO 3–26

Desktop shortcut
creating 5–24

Destroy ADM Super Procedures option
4–86

Dynamic Combo 1–6

Dynamic combo
adding to a Viewer 4–6
defined 1–7

Dynamic Launcher 4–39

Dynamic Lookup 1–6
adding 4–11

Dynamic lookup
defined 1–7

Dynamic menu controller 4–71

Dynamic objects
defined 1–4

Dynamic Properties window 4–4

Dynamic windows
creating 4–30

Dynamics ADE. *See also* Application
Development Environment (ADE) 1–5

Dynamics AppBuilder 1–5

Dynamics Repository 1–4
defined 1–4

Dynamics. *See* Progress Dynamics

DynField object type 4–43

DynObj type 4–32

DynSports
application database 2–12
connecting 3–4

E

Editing objects 4–3

EMP Product Module
creating 3–9

Entities
importing 3–9

Entity Import dialog box 3–9

Environment Managers
list of 1–7
using 5–2

ERwin 1–5

Excel 4–43

F

Field name separators 3–11

Field objects 1–7
 defined 1–7

Files
 configuration 5–21
 icfconfig.xml 5–21

Filter Records button 4–41

Finished application 4–2

Folder Window to Launch option 3–28

Follow joins option 3–21

Foreign Fields 4–52

Framework managers 1–7

fullb suffix 3–20

fullo suffix 3–20

G

General Product Module
 creating 3–9

Generating objects 3–20

Grid
 bottom section 4–60

I

icfconfig.xml 5–21

icfdb database 1–4

Importing entities 3–9

Independent Windows 4–32

Index information 4–41

Instance Properties window 4–8, 4–14

Internal joins
 following 3–21

Item Type option 4–77

Items 4–72

J

Joins 3–21

K

Key values 4–52

L

Launching containers 4–39

Layout
 changing 4–60

Link field option 4–15

Linked widget column 4–15

Links
 Customer browse window 4–37
 Customer orders browse 4–56
 Order and lines browse 4–66, 4–68

Logical Services Control window 5–12

Login company 2–18

Login language 5–9

Login window 2–17, 3–3

Lookup
 adding 4–11

M

Main menu window
creating 4-71

Maintain System Data option 5-9

Maintenance window
creating 4-43
Order 4-63

Managers 1-7
list of 1-7
using 5-2

Manuals
where to go from here 5-26

Menu controllers 4-71

Menu designer 4-72

Migrating
static objects 1-5

Modules
creating 3-6

N

Naming conventions
object suffixes 3-20

New user
creating 5-9

Nonvisual objects
showing in grid 4-33

O

Object Controllers 4-32

Object Generator tool 3-20
Query 3-26

Object repository 1-4

Object types
DynField 4-43

Objects
deleting from grid 4-46
dynamic 1-4
dynamic physical and logical 1-4
editing 4-3
generating 3-20
migrating 1-5
static 1-4
viewing 3-25

Order Entry Menu window 4-71

Order Entry module
creating 3-9

Order Selection window 4-39

Order/Lines page 4-62

P

Page labels
editing 4-46

Pages
adding 4-48

Physical Service Control window 5-14

Prefix length
tables 3-11

Product Control window 3-6

Product Maintenance window 3-7

Product Modules
defining 3-6

Products
defining 3-6

Profile Manager 5-9

Progress Dynamics

- application components 1–6
- data objects 1–6
- defined 1–1
- managers 1–7
- repository function
- tool set 1–5
- UI objects 1–7

Properties

- Session Type 5–20

Properties window 4–3**Property sheet**

- Browse 3–28

Q**Query string**

- instance level 4–8, 4–14

R**Records**

- filtering 4–41

Relative layout 4–60**Relative path directory**

- Product Module 3–9

Repository. *See* Dynamics Repository**Run Locally option 5–13, 5–15****Run Persistent option 4–40****Running containers 4–39****Running the application 5–24****S****SBO. *See* SmartBusinessObject (SBO)****SDO. *See* SmartDataObject (SDO)****SDV. *See* SmartDataViewer (SDV)****Separators**

- field names 3–11

Sequence

- browse 4–15

Session Manager 5–16**Session types**

- editing 5–16

Shortcut

- creating 5–24

Show nonvisual objects button 4–33**Show visual objects button 4–33****SmartBusinessObject (SBO) 1–6**

- defined 1–6

SmartDataField icon 4–7**SmartDataObject (SDO) 1–6, 3–20**

- defined 1–6, 1–7

SmartDataViewer (SDV) 4–3

- defined 1–7

Source code management (SCM)

- RVDB 1–7

Static objects

- defined 1–4

Suffixes 3–20**Super procedures**

- custom 3–28
- destroy option 4–86

System Owned option 4–77

T

- Templates 4–32
- Toolbar and Menu Designer 4–72
- Toolset 1–5
- TreeView window layout 4–72
- Type property 4–3

U

- UI objects 1–7
 - defined 1–6
- User interface objects 1–6
- Users
 - adding 5–9

V

- Viewers 3–20
- Viewing objects 3–25
- viewv suffix 3–20
- Visual objects
 - showing in grid 4–33

W

- Window container objects
 - defined 1–7
- Window Name attribute 4–37
- Window object. *See* Window container objects
- Write to Config option 5–13, 5–15

X

- XML configuration file 5–21