



Progress Dynamics® Administration Guide

© 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, 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: 3858
Item Number: 103615;V2.1B

Contents

Preface	xi
Purpose	xi
Audience	xi
Organization of this manual	xi
Typographical conventions	xiii
1. Overview of Progress Dynamics Administration	1-1
1.1 Tasks for Progress Dynamics administrators	1-2
1.2 Tools for Progress Dynamics administration	1-2
1.2.1 Application menu	1-3
1.2.2 Deployment menu	1-5
1.2.3 Security menu	1-6
1.2.4 Session menu.	1-7
1.2.5 System menu	1-9
1.2.6 Transaction menu.	1-10
2. Defining and Managing Sessions	2-1
2.1 What is a Progress Dynamics session?	2-2
2.1.1 Predefined session types	2-3
2.2 Starting a Progress Dynamics session	2-5
2.2.1 How sessions start	2-7
2.3 Configuring a Progress Dynamics session	2-8
2.3.1 Overview of session configuration tasks	2-8
2.3.2 Defining session services.	2-9
2.3.3 Creating session properties	2-14
2.3.4 Defining managers	2-16
2.3.5 Creating a new session type	2-17
2.3.6 Generating the configuration file	2-25
2.3.7 Using minimum XML files.	2-27

2.3.8	Creating a shortcut icon to start a session	2-28
2.3.9	Importing a configuration file	2-31
2.4	Defining user profile data for a session	2-32
2.4.1	What is user profile data?	2-32
2.4.2	How Progress Dynamics uses profile data in a session	2-32
2.4.3	Defining user profiles	2-33
3.	Setting Up Basic Security Options	3-1
3.1	Setting global security and management options	3-2
3.2	Choosing a security model: grant versus revoke	3-4
3.2.1	Revoke model	3-4
3.2.2	Grant model	3-5
3.3	Setting up user authentication	3-7
3.3.1	Defining user categories	3-7
3.4	Defining login companies	3-9
3.5	Defining security groups	3-11
3.6	Creating and maintaining users	3-19
3.6.1	Setting a user's password	3-22
3.6.2	Defining a user as a profile user	3-23
3.6.3	Linking users and security groups	3-26
4.	Supporting Built-in Application Features	4-1
4.1	Specifying gapless sequences	4-2
4.1.1	Sequence Control window	4-2
4.1.2	Sequence Maintenance window	4-3
4.2	Defining languages, countries, nationalities, and translations	4-4
4.2.1	Language Control window	4-4
4.2.2	Language Maintenance window	4-5
4.2.3	Country Control window	4-5
4.2.4	Country Maintenance window	4-6
4.2.5	Nationality Control window	4-7
4.2.6	Nationality Maintenance window	4-7
4.2.7	Setting translation preferences	4-8
4.2.8	Translation Control window	4-9
4.2.9	Translation Maintenance window	4-9
4.2.10	Testing the new translation	4-10
4.2.11	User run-time translations with the Translate window	4-11
4.2.12	Menu translations	4-12
4.2.13	Entity translations	4-14
4.3	Adding generic database comments and auditing	4-15
4.3.1	Generic comments	4-15
4.3.2	Generic auditing	4-19

4.4	Specifying multi-media types	4-23
4.4.1	Multi Media Type Control window	4-23
4.4.2	Multi Media Type Maintenance window	4-24
4.5	Defining application categories	4-25
4.5.1	Category definition structure	4-25
4.5.2	Category Control window	4-26
4.5.3	Category Maintenance window	4-27
4.6	Maintaining user-defined status information	4-27
4.6.1	How Progress Dynamics sets up the status mechanism	4-27
4.6.2	Information on adding status tracking	4-28
4.6.3	Status Control window	4-28
4.6.4	Status Maintenance window	4-29
4.7	Exporting data through Print Preview	4-30
4.7.1	Setting session properties	4-30
4.7.2	Using Print Preview	4-32
5.	Extending the Progress Dynamics Configuration Utility	5-1
5.1	Planning ahead	5-2
5.2	Progress Dynamics Configuration Utility overview	5-3
5.2.1	Advantages of the DCU	5-4
5.2.2	How the DCU differs from a Progress Dynamics session	5-5
5.2.3	DCU interface	5-6
5.3	Deployment stages	5-9
5.3.1	Phase 1 and Phase 2 processing stages	5-10
5.3.2	Logging for Phase 1 and Phase 2	5-11
5.3.3	DCU processing sequence	5-12
5.3.4	Picking the correct processing stage	5-13
5.4	DCU XML files	5-13
5.4.1	DCU driver files	5-15
5.4.2	Upgrade files	5-22
5.5	DCU API	5-27
5.5.1	analyzeCase	5-28
5.5.2	analyzef	5-29
5.5.3	btnChoose	5-29
5.5.4	checkForDB	5-30
5.5.5	connectDatabase	5-30
5.5.6	eventProc	5-31
5.5.7	getDBFile	5-31
5.5.8	getDirectory	5-31
5.5.9	gotoPage	5-32
5.5.10	obtainICFSeqVals	5-32
5.5.11	obtainPatchList	5-32
5.5.12	processParams	5-33
5.5.13	restoreProperties	5-33

5.5.14	screenScrape	5-33
5.5.15	startUpgradeProcess	5-34
5.5.16	validateDirectory	5-34
5.5.17	validateSiteNumber	5-34
5.5.18	verifyDBVersion	5-35
5.6	Creating a customized DCU deployment	5-35
5.6.1	Prepare schema and data files	5-35
5.6.2	Designing upgrade programs	5-36
5.6.3	Upgrade program attributes	5-38
5.6.4	Setting up new databases with the DCU	5-38
5.6.5	Upgrading existing databases with the DCU	5-43
5.6.6	Tips for customized DCU sessions	5-45
5.6.7	Starting the DCU	5-46
5.7	Running the DCU in batch mode	5-47
5.7.1	Generating the batch-mode data	5-48
5.7.2	Creating a batch-mode DCU script	5-49
5.8	Dumping and loading site-specific data	5-52
5.8.1	Site Data Dump and Load utilities architecture	5-53
5.8.2	Setting up the utilities	5-54
5.8.3	Writing a dump program	5-56
5.8.4	Writing a load program	5-58
5.8.5	Creating a dumpconfig.txt file	5-61
5.8.6	Using the GUI	5-63
5.8.7	Calling the Utility from the DCU	5-64
A.	Using Progress DataServers with Progress Dynamics	A-1
A.1	Overview	A-2
A.1.1	Progress Dynamics support for DataServers	A-2
A.1.2	When to use DataServers with Progress Dynamics	A-2
A.1.3	Requirements for DataServers in Progress Dynamics	A-2
A.2	Connecting the database	A-3
A.2.1	Configuring Progress DataServer for MS SQL Server	A-3
A.2.2	Configuring Progress DataServer for Oracle	A-5
A.2.3	Schema holder database	A-6
A.2.4	Progress Dynamics configurations with DataServers	A-6
A.3	Programming considerations	A-6
A.3.1	Indexes	A-6
A.3.2	Two-phase commit	A-7
A.4	Other useful documentation	A-7

B.	Progress Dynamics Session Properties	B-1
C.	Inside the Progress Dynamics Configuration File	C-1
C.1	Session nodes	C-2
C.1.1	Properties node	C-2
C.1.2	Services node	C-4
C.1.3	Managers node	C-5
D.	Deployment Notes	D-1
D.1	Deploying multi-transaction sequences	D-2
D.2	Generating static-4GL equivalents of dynamic objects	D-2
D.2.1	Mechanisms for static object generation	D-2
D.2.2	Saving dynamic objects as static objects.	D-4
D.2.3	Generating 4GL for dynamic objects as static objects.	D-7
D.2.4	Generated-4GL object execution and file naming	D-12
D.2.5	Deploying generated-4GL static objects	D-14
D.2.6	Development impact using generated-4GL objects.	D-16
D.3	Deploying Repository objects between Progress Dynamics versions ..	D-18
D.3.1	Cross-version deployment situations	D-19
D.3.2	Progress Dynamics support for cross-version deployments ..	D-19
D.3.3	Recommended procedure for deploying across versions	D-20
E.	Performance Notes	E-1
E.1	Configuring server and client startup options	E-3
E.2	Using static-4GL equivalents of dynamic objects	E-5
E.3	Keeping SmartDataObjects alive on the server	E-5
E.3.1	Configuring SDOs to remain alive	E-5
E.3.2	Using SDOs kept alive on the server.	E-6
E.4	Progress Dynamics lookup/combo (SmartDataField) cache	E-6
E.4.1	Cache operation.	E-7
E.4.2	When Dynamics does not use the SmartDataField cache.	E-8
E.4.3	Enabling and disabling the SmartDataField cache	E-9
E.4.4	Clearing the SmartDataField cache	E-10
E.4.5	Disabling SmartDataField (SDF) cache usage for a specified SDF	E-11
E.5	Dynamic lookup mapped fields	E-12
E.5.1	Benefits and constraints on mapping fields	E-12
E.5.2	Using mapped fields	E-13
E.5.3	Linking lookup fields directly to SDO fields on the viewer	E-14
E.5.4	Mapping lookup fields to SDO fields through viewer widgets .	E-17
E.5.5	When Dynamics does not use mapped fields	E-21

E.6	Class and entity cache	E-21
E.6.1	Benefits and constraints on the class and entity cache	E-21
E.6.2	Cache architecture	E-22
E.6.3	How the Progress Dynamics framework uses the cache	E-22
E.6.4	Generating the cache	E-24
E.6.5	Configuring the class and entity cache	E-26
E.6.6	Using the class and entity cache	E-26
E.7	Toolbar Image Optimization Using PicClip Images	E-27
E.7.1	Benefits and constraints on using PicClip images	E-27
E.7.2	Using PicClip images for custom toolbars	E-28
E.8	Caching toolbars and container menus at session startup	E-29
E.8.1	Benefits and constraints on toolbar and menu pre-caching ...	E-29
E.8.2	Configuring toolbar and menu pre-caching	E-30
E.8.3	Using toolbar and container menu pre-caching	E-32
E.9	Thin SmartObject rendering	E-32
E.9.1	Configuring thin SmartObject rendering	E-34
E.9.2	Using thin rendering	E-35
E.10	Dynamic TreeView optimizations	E-35
E.10.1	Keeping dynamic frames alive on the client	E-36
E.10.2	Configuring the node batching feature	E-37
E.10.3	Using dynamic TreeViews	E-40
E.11	SmartDataObject data definition and schema location	E-40
E.11.1	Configuring the location for field definitions	E-41
E.11.2	Using the SchemaLocation attribute	E-41
E.12	SuperProcedureMode attribute	E-42
E.12.1	Using the attribute	E-42
E.13	Dynamic call wrapper(dynlaunch.i)	E-42
E.13.1	Using dynlaunch.i	E-43
E.14	Creating customized login windows	E-43
E.14.1	Using Dynamics support for customized login windows	E-43
Index		Index-1

Tables

Table 2–1:	Predefined session types	2–4
Table 2–2:	Logical service properties	2–12
Table 2–3:	Qualifiers for the -icfparam startup parameter	2–28
Table 4–1:	Print Preview session properties	4–30
Table 5–1:	DCU standard pages and their objects	5–8
Table 5–2:	DCU deployment stages	5–10
Table 5–3:	Operators for conditional expressions	5–19
Table 5–4:	DCU procedure files	5–27
Table 5–5:	DCU session properties	5–36
Table 5–6:	Standard program properties	5–38
Table 5–7:	Site Data Dump and Load installed files	5–54
Table 5–8:	Field options for creating the dumpconfig.txt file	5–61
Table B–1:	Progress Dynamics session properties	B–1
Table C–1:	Physical session types	C–3
Table C–2:	Manager static handle codes	C–5
Table D–1:	Determining the generated filename for a generated-4GL object	D–13
Table D–2:	Minimum files to deploy for generated-4GL static objects	D–15
Table D–3:	APIs for running generated-4GL static objects	D–17
Table E–1:	Optimal settings for server or client startup parameters	E–3
Table E–2:	Column values for lookup field mapping directly to SDO	E–15

Figures

Figure 1-1:	Administration main window	1-2
Figure 1-2:	Application menu	1-3
Figure 1-3:	Deployment menu	1-5
Figure 1-4:	Session menu options	1-7
Figure 1-5:	System menu	1-9
Figure 2-1:	Session type inheritance	2-3
Figure 2-2:	Required Manager inheritance	2-19
Figure 2-3:	Session property inheritance	2-21
Figure 2-4:	Session services inheritance	2-23
Figure 3-1:	Security Control window	3-2
Figure 3-2:	Revoke model example	3-13
Figure 3-3:	Grant model example	3-14
Figure 4-1:	Sequence Control window	4-2
Figure 4-2:	Sequence Maintenance window	4-3
Figure 4-3:	Language Control window	4-4
Figure 4-4:	Language Maintenance — New window	4-5
Figure 4-5:	Country Control window	4-5
Figure 4-6:	Country Maintenance window	4-6
Figure 4-7:	Nationality Control window	4-7
Figure 4-8:	Nationality Maintenance window	4-7
Figure 4-9:	Dynamics Preferences dialog box	4-8
Figure 4-10:	Translation Control window	4-9
Figure 4-11:	Translation Maintenance window	4-9
Figure 4-12:	Translate window for user run-time translations	4-11
Figure 4-13:	Translate window with new translation	4-11
Figure 4-14:	Administration window with run-time translation	4-12
Figure 4-15:	Multi Media Type Control window	4-23
Figure 4-16:	Multi Media Type Maintenance window	4-24
Figure 4-17:	Multi Media Typed Control window with new media type	4-24
Figure 4-18:	Category Control window	4-26
Figure 4-19:	Category Maintenance window	4-27
Figure 4-20:	Status Control window	4-28
Figure 4-21:	Status Maintenance window	4-29
Figure 4-22:	Default Print Preview settings	4-31
Figure 4-23:	Overriding Print Preview settings	4-32
Figure 5-1:	Application design cycle	5-3
Figure 5-2:	DCU interface	5-6
Figure 5-3:	Standalone GUI for Site Data Dump and Load utilities	5-63
Figure E-1:	Disabling the SmartDataField cache for a specified SmartDataField . .	E-11
Figure E-2:	PicClip file, toolclip.bmp	E-27

Preface

Purpose

This manual describes how to configure and manage Progress Dynamics®. It also describes a common set of administrative procedures that you might need to perform to support all of the applications that you develop using Dynamics.

Audience

This manual is designed for Progress Dynamics programmers who have some familiarity with the Progress® ADM and who are already familiar with the *[Progress Dynamics Installation Guide](#)*.

Organization of this manual

This guide is organized in the following manner:

[Chapter 1, “Overview of Progress Dynamics Administration”](#)

Describes the basic administrative tasks and tools.

[Chapter 2, “Defining and Managing Sessions”](#)

Describes how to create and configure Progress Dynamics sessions, which are runtime and development environments for applications.

[Chapter 3, “Setting Up Basic Security Options”](#)

Shows the requirements and how to install Progress Dynamics on your system and configure it for basic operation.

Chapter 4, “Supporting Built-in Application Features”

Describes how to prepare several application features that are partly or completely built into Progress Dynamics that you can set up to work in your own applications.

Chapter 5, “Extending the Progress Dynamics Configuration Utility”

Describes how to extend the Dynamics Configuration Utility (DCU) in order to deploy your own Dynamics applications, including information on running the DCU in batch mode and saving site-specific data during an application installation or upgrade.

Appendix A, “Using Progress DataServers with Progress Dynamics”

Describes some of the factors you should consider when using a Progress DataServer in the Progress Dynamics environment.

Appendix B, “Progress Dynamics Session Properties”

Describes describes the Progress Dynamics session properties.

Appendix C, “Inside the Progress Dynamics Configuration File”

Describes the contents of the default configuration file to help you gain an understanding of what this XML file does.

Appendix D, “Deployment Notes”

Describes deployment features that supplement the features described in the deployment white paper posted on the Progress Software Developers Network® (PSDN) Web site (<http://psdn.progress.com>).

Appendix E, “Performance Notes”

Describes features that enable you to improve Progress Dynamics application performance, depending on your application design and deployment requirements.

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 file names and path names

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

Overview of Progress Dynamics Administration

This chapter provides an overview of Progress Dynamics® administration and includes descriptions of:

- [Tasks for Progress Dynamics administrators](#)
- [Tools for Progress Dynamics administration](#)

1.1 Tasks for Progress Dynamics administrators

The tasks described in this manual cover the following topics:

- **Defining and managing Progress Dynamics sessions** — How to use the flexible mechanism for configuring Progress Dynamics environments to run applications in different application client, application server, database, and hardware configurations with little or no change to application data and code.
- **Setting up basic management and security options** — How to use Progress Dynamics tools to set up an environment that is structurally sound and secure.
- **Supporting application features** — How to include information on using a variety of application functions that are partly or completely built into the Progress Dynamics framework, such as generic database comments and auditing, localization, and access to multi-media files.

For information about source control management, application deployment, and upgrading Progress Dynamics, go to:

http://psdn.progress.com/library/progress_dynamics/index.ssp

The *Progress Dynamics Installation Guide* also contains information about using the Dynamics Configuration Utility to upgrade Progress Dynamics from one release to another, while ensuring database integrity.

1.2 Tools for Progress Dynamics administration

[Figure 1–1](#) shows the Administration main window that you use for most Progress Dynamics administration tasks. You can start the Administration main window from the AppBuilder Tools menu. You can also open it from the Links menu on the Dynamics Development window.

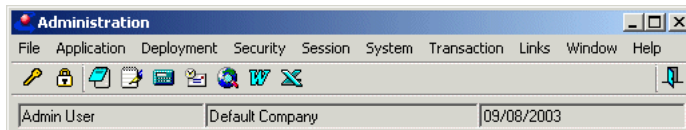


Figure 1–1: Administration main window

The following sections describe the tools available from the Administration main window, and where to find more information about them.

1.2.1 Application menu

Figure 1–2 shows the Application menu. This manual covers only some of the functions available, namely, those that have the most impact on supporting general application development functions.



Figure 1–2: Application menu

The Application menu contains the following functions:

- **Product Maintenance** — Defines details about the installed products and product modules.
- **Instance Attribute Control** — Defines the unique, instance-specific characteristics of generic objects, customized to suit a particular application of it.
- **Toolbar and Menu Designer** — Allows for the creation of items, bands, band structures, toolbar/menu structures, and object/band association structures.
- **Country Control** — Defines countries for application localization.
- **Nationality Control** — Defines nationalities for application localization.
- **Language Control** — Defines languages for application localization.
- **Translation Control** — Defines translations for application text objects for any defined language. This includes information on the run-time Translate Window function that allows users to translate text objects in any application window they are using while they use it.

- **Category Control** — Defines criteria for data categorization that can be applied to any table, using a one to three-level category typing scheme. This facility is critical to implementing several other Progress Dynamics-supported application features, including generic database comments, progressive status tracking, among others. You can use it wherever you need a way to add a degree of logical structure to the flat data of a table, or even to define valid lists of values.
- **Custom Procedure Control** — Defines logical names and descriptions for custom procedures that you want to access in the framework.
- **Multi Media Type Control** — Defines the file types and instantiating program for virtually any type of file format, including sound, audio-visual, video, or word-processing files. This allows such files to be linked to and referenced from Progress Dynamics entities.
- **Multi Media Image Control** — Maintains images being added to the Multi Media table. This was mainly added to allow the addition of images for the Dynamic TreeView nodes but does not limit it to that.
- **Status Control** — Defines progressive status codes that you can use to maintain states of change and thresholds in your application data, including status histories of any extent. This is to support the internal workings of a status tracking system that you design and implement as part of your application.
- **Currency Control** — Defines currencies for application localization.

For more information on these controls, see [Chapter 4, “Supporting Built-in Application Features.”](#) This also includes a description of how to use the Generic Database Commenting and Auditing features that are built into the Progress Dynamics UI and data management infrastructure.

1.2.2 Deployment menu

Figure 1–3 shows the Deployment menu, which provides the tools for deploying Progress Dynamics applications from one Progress Dynamics site to another.

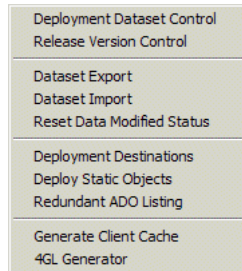


Figure 1–3: Deployment menu

The following tools are on the Deployment menu:

- **Deployment Dataset Control** — Defines the contents of datasets that you can use to deploy related tables from the Progress Dynamics Repository or application databases. The Site Control on the System menu is also essential to this process (see the “[System menu](#)” section).
- **Release Version Control** — Allows you to specify release milestones that you can use to specify objects for deployment that have changed since a prior release milestone.
- **Dataset Export** — Filters the records to include in a deployment dataset and exports the dataset in a format ready for import at another Progress Dynamics site.
- **Dataset Import** — Imports a deployment dataset generated with the Dataset Export tool at another Progress Dynamics site.
- **Reset Data Modified Status** — Resets the Data Modified Status of all the data in the Repository so that the data is listed as Not Modified.
- **Deployment Destinations** — Specifies deployment destinations for one or more static objects.
- **Deploy Static Objects** — Creates deployment packages containing pre-compiled static objects for an individual deployment destination.
- **Redundant ADO Listing** — Generates a list of the ADO files that no longer exist in the Repository. You can then use this list to remove the redundant ADOs from the Repository.

- **Generate Client Cache** — Creates a cache of application objects and entities used by the client. Caching can improve performance by reducing the number of calls to the Repository.
- **4GL Generator** — Generates static 4GL equivalents of dynamic objects that are optimized for performance. You can then deploy these static 4GL equivalents as the run-time versions of the dynamic objects that you otherwise deploy and maintain in the design-time Repository.

For more information on the Generate Client Cache tool, see the information on the class and entity cache in [Appendix E, “Performance Notes.”](#) For more information on the 4GL Generator tool, see the sections on generating 4GL equivalents of dynamic objects in [Appendix D, “Deployment Notes.”](#)

For more information on deploying Progress Dynamics applications and Dynamics deployment tools, see the product release notes and deployment white papers posted on the following Progress Software Developers Network® (PSDN) Web site:

<http://psdn.progress.com>

1.2.3 Security menu

The Security menu contains a single function, **Security Control**. You use this function for adding basic security, and for creating security restrictions on virtually any object that requires security in an application.

This manual covers only basic security, including the following features in the Security Control:

- **Security Maintenance** — Sets global security and functional options for your Progress Dynamics installation.
- **Login Companies** — Defines organizations (login companies) that you can use to group users and application functionality, perhaps based on a company product line or a group expertise.
- **Groups** — Defines users, their login names, passwords, privileges, and all the other criteria used to authenticate a user so they can login and work with the appropriate application function.

- **User Categories** — Defines categories of users irrespective of location or affiliation, perhaps as a means to distinguish, for example, novice from expert users.
- **Users** — Defines users, their login names, passwords, privileges, and all the other criteria used to authenticate a user so they can login and work with the appropriate application functions.

For more information on using these security functions, see [Chapter 3, “Setting Up Basic Security Options.”](#) For more information on advanced security options, including the remaining functions on the Security Control window, see the *[Progress Dynamics Developer’s Guide](#)*.

1.2.4 Session menu

[Figure 1–4](#) shows the options in the Session menu. You use this menu to configure Progress Dynamics sessions with Progress and system resources, and to set up options for session operations that are independent of the application that is running.

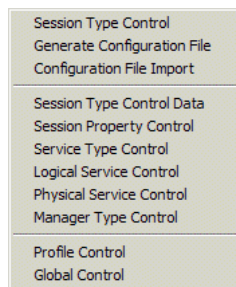


Figure 1–4: Session menu options

The Session menu contains the following functions:

- **Session Type Control** — Defines the different types of sessions in which applications can run, for example, sessions configured for development, application run time, WebClient™ n-tier operation, and so on.
- **Generate Configuration File** — Generates a configuration XML file that contains the startup configuration for any selection of session types that you have defined in the Repository.
- **Configuration File Import** — Allows you to import a configuration XML file that you might have modified manually, allowing you to synchronize it with the Repository.

- **Session Type Control Data** — Specifies how to handle session types that are connected by a client and are not registered in the Repository or that are configured without a clear indication of where, between the Repository and the configuration XML file, to resolve their settings.
- **Session Property Control** — Defines the run-time options that traditionally apply to the SESSION handle, but also additional options that Progress Dynamics requires or you find helpful in defining the run-time environment for a session.
- **Service Type Control** — Defines the basic types of services that make up a session type and thus become session services, such as database and Progress® AppServer™ services.
- **Logical Service Control** — Defines a component of session services that specifies the more abstract characteristics of a service type, providing greater flexibility in service definition to handle various local and remote configurations of the same physical services.
- **Physical Service Control** — Defines the connection parameters that establish connections between a client and a specific physical database, AppServer, or other service.
- **Manager Type Control** — Specifies standard or custom session managers, the persistent procedures that, along with the Repository, provide the core of Progress Dynamics functionality.
- **Profile Control** — Defines operational options that a user can enable or disable for a current session or to apply more permanently across multiple sessions, such as maintaining window sizes and positions, default SmartDataObject™ preferences, and so on.
- **Global Control** — Defines system control defaults. These are parameters that determine the system operation, such as date format and currency.

For more information on using these session management functions, see [Chapter 2, “Defining and Managing Sessions”](#) and session management white paper posted on the the following PSDN Web site:

<http://psdn.progress.com>

1.2.5 System menu

[Figure 1–5](#) shows the System menu. This manual covers only some of the functions available, namely, those that have the most impact on installation configuration and administration.

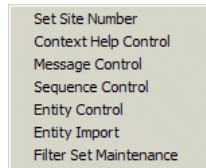


Figure 1–5: System menu

The System menu contains the following functions:

- **Set Site Number** — Sets a unique site identification number for a registered Progress Dynamics site in the Progress Dynamics Repository database (ICFDB).
- **Context Help Control** — Displays the online help context number mappings for application user interface containers, objects, and fields.
- **Message Control** — Displays the messages you can use in your application.
- **Sequence Control** — Displays the available sequences.
- **Entity Control** — Displays the entities in all or one of the connected databases from which you have already imported entities.
- **Entity Import** — Imports entity mnemonic records for one or more application database tables.
- **Filter Set Maintenance** — Defines and names groups of rules for including or excluding data from queries during a particular session.

For more information on sequences and generic database auditing, see [Chapter 4, “Supporting Built-in Application Features.”](#)

The *Progress Dynamics Developer’s Guide* describes how to use the Entity Import and Entity Control tools.

1.2.6 Transaction menu

The Transaction menu contains a single function, **Audit Control**. This function provides a means to access all audit records that have been generated from tables that have Generic Auditing turned on. It works very much like the Audit button that allows users to access audit records generated on the data with which they are working. However, this control allows you to review the audit records for an entire Progress Dynamics installation. For more information on Generic Auditing and reviewing audit records, see [Chapter 4, “Supporting Built-in Application Features.”](#)

Defining and Managing Sessions

This chapter provides the information you need to work with Progress Dynamics sessions. It includes the following sections:

- [What is a Progress Dynamics session?](#)
- [Starting a Progress Dynamics session](#)
- [Configuring a Progress Dynamics session](#)
- [Defining user profile data for a session](#)

2.1 What is a Progress Dynamics session?

A session is the context in which Progress Dynamics itself, and Progress Dynamics applications run. This context includes:

- **Required Managers** — Each session type requires a set of persistent procedures referred to as managers. The managers implement the core functionality of the Progress Dynamics framework. Each session type has different requirements regarding which managers it needs to run. For example, the session type used to run applications in Web browsers (ICFWS), requires a User Interface Manager to receive data from a Web stream. In an application development session (ICFDev), the User Interface Manager is not required.
- **Session Properties** — Session properties are a list of startup parameters and environment variables for the session. Sessions can be defined with a wide range of properties including PROPATH settings, startup procedures, date/time formats, user login requirements, and so on.
- **Session Services** — Session services are the physical and logical resources that the session type must connect to. For example, the session type that you use to run an application would include the Repository (ICFDB) and an application database as session services.

You can view the managers, properties, and services defined for each session by starting the Session Type Maintenance tool. To start the Session Type Maintenance tool, choose **Session→Session Type Control** from the Administration tool menu bar.

When you start a Progress Dynamics session, the Configuration File Manager reads the definition of the session from an XML configuration file. You can generate configuration files for a single session type or that contain several session types. You can access the Generate Configuration File tool from either the Administration window's Session menu or from the Session Type Maintenance tool's Option menu.

The session information in the configuration file allows Progress Dynamics applications to run in different session types with little or no manual reconfiguration. You simply include the session type and location in the XML configuration file as arguments to the -icfparm startup parameter for the application. For example, the following fragment of a startup command specifies ICFDev as the session type defined in the configuration file located in the DynamicsWRK folder:

```
. . . -icfparm ICFSESSTYPE=ICFDev, ICFCONFIG="C:\DynamicsWRK\icfconfig.xml"
```

A session begins when you start Progress Dynamics or a Progress Dynamics application. The session ends when you stop Progress Dynamics or an application. You can also define the session to stop when it times out due to inactivity.

2.1.1 Predefined session types

Session types are defined in a hierarchy where basic session types are extended to create more complex session types. A session type that extends another session type inherits the properties of that other session type. You can then override existing properties with new values and add additional properties to create a different environment.

Several session types are predefined and shipped with the Progress Dynamics software to give you a place to start. These session types represent guidelines for creating your own session types. [Figure 2–1](#) shows the inheritance between the standard session types.

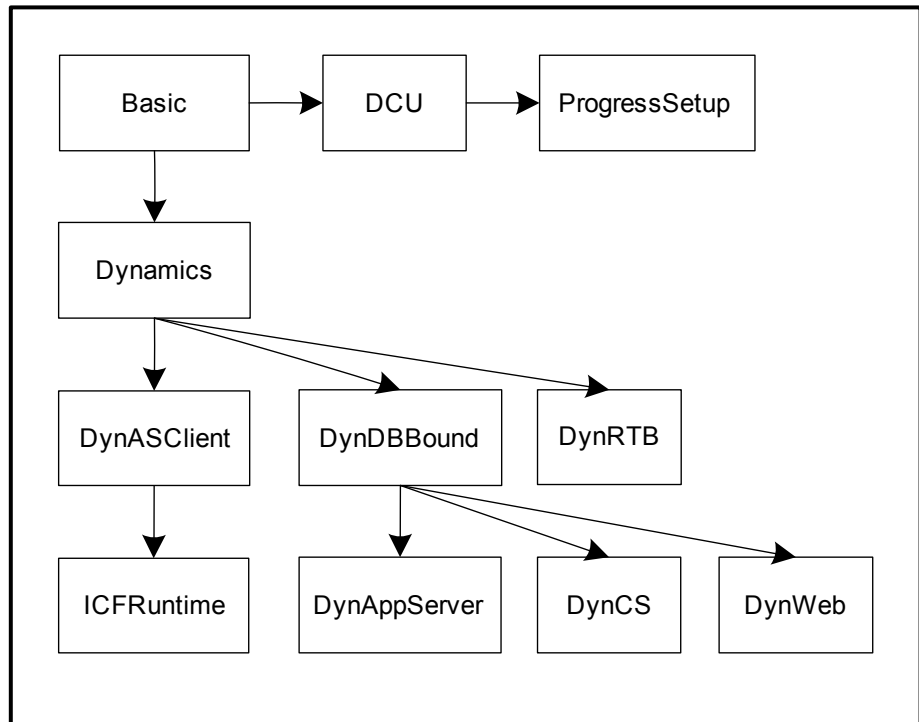


Figure 2–1: Session type inheritance

[Table 2–1](#) describes the session types that ship with Progress Dynamics.

Table 2–1: Predefined session types

Session type	Description
Basic	The starting point for all session types. It supplies common connection information for starting the framework.
DCU	The basic Progress Dynamics Configuration Utility (DCU) session type. The DCU runs in a specialized session that does not include all the managers of a Dynamics session. This session type supplies common information for all DCU sessions.
ProgressSetup	The standard DCU session type. You can extend this session type to create a session type for your custom DCU deployments.
Dynamics	The basic session type for session that use the full framework.
DynASClient	The basic session type for Progress Dynamics applications running across the AppServer.
ICFRuntime	The standard AppServer runtime client session type.
DynRTB	The basic session type for sessions that use Roundtable. You would extend this session type for each particular Roundtable environment.
DynDBBound	The basic session type for sessions that run with a direct connection to the Repository.
DynAppServer	The standard session type for database-bound sessions that run with the AppServer. This session is extended to create development and runtime session types.
DynCS	The standard session type for database-bound sessions that run client/server. This session is extended to create development and runtime (default) session types.
DynWeb	The standard session type for database-bound sessions that run over the Web. This session is extended to create the default Web session (ICFWS).

2.2 Starting a Progress Dynamics session

Before you start a Progress Dynamics session, you must start the Dynamics Repository database, ICFDB. You **must** connect to ICFDB to access the Dynamics tools in the AppBuilder. You also must start the ICFDB database in multi-user mode so that you can make client, AppServer, and/or Web Server connections as needed.

When you installed Progress Dynamics, the Dynamics Configuration Utility (DCU) created shortcuts to batch files that start (in multi-user mode) and stop the ICFDB Repository.

To start a Progress Dynamics session:

- 1 ♦ To start the Dynamics Repository database, select **Start→Progress Dynamics→Start Dynamics DB Servers**.

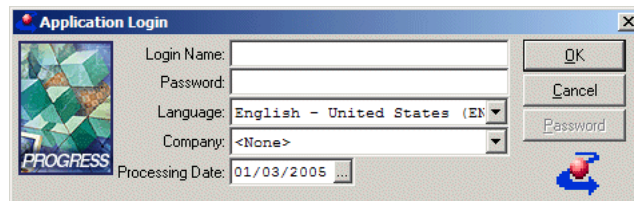
You might see the batch file run briefly in a command window. The command window will close once the batch file completes running.

- 2 ♦ Start your application databases.

You can use a database shortcut and create a new startup icon that starts your application databases for you. See the [Getting Started with Progress Dynamics](#) tutorial for more information on creating database startup shortcuts.

- 3 ♦ To start a Progress Dynamics development session, choose **Start→Progress Dynamics→Dynamics Development**.

By default, this starts the client/server development (ICFDev) session. After you start the framework, the first screen you see is an Application Login dialog box, as shown:



NOTE: The processing date on the login screen comes from the startup parameters in the configuration file. For more information on the configuration file, see the [“Generating the configuration file”](#) section.

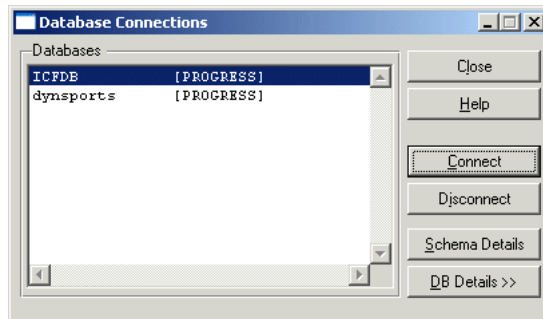
The login program provides for application security and language control. For more information about setting up and using basic security features, see [Chapter 3, “Setting Up Basic Security Options.”](#)

- 4 ♦ Type a Login Name value (the default is **admin**). The AppBuilder, along with the PRO*Tools and Object Palette, appear.

For more information about setting up you own login name, see the [“Defining session services”](#) section.

- 5 ♦ Verify that your Repository and application databases are connected. Select **Tools→Database Connections** from the AppBuilder main menu.

The following shows that the ICFDB Repository and the DynSports sample application database are connected to the session:



NOTE: The application database is not connected unless you modified the default ICFDev session type to include it as a service. (See the [“Configuring a Progress Dynamics session”](#) section.) However, if the application database is running, you can add it at this point by choosing the **Connect** button and connecting in multi-user mode.

- 6 ♦ To stop the Dynamics Repository database at the end of a session, select **Start→Progress Dynamics→Stop Dynamics DB Servers**.

2.2.1 How sessions start

Here is what happens when you start a Progress Dynamics session:

1. The Configuration File Manager is launched and initializes the session. The Configuration File Manager is a persistent procedure that determines and establishes the startup environment. It gets its session startup information from the Progress Dynamics configuration file, `icfconfig.xml`.

The `icfconfig.xml` file defines the environment in which a Progress Dynamics can run. The definition of each session type includes the following startup information:

- Its logical and physical resources (session services).
- Its startup parameters (session properties).
- The management procedures (Progress Dynamics managers) with which it runs.

NOTE: The Progress Dynamics configuration file can reside at a central source, anywhere on a network that can host an XML file, on behalf of all Progress Dynamics installations that rely on it. You can also import any manual changes that you make to the configuration file so all of its startup information is synchronized with and secure in the Repository. For more information, see the [“Importing a configuration file”](#) section.

For more information about the `icfconfig.xml` file, see [Appendix C, “Inside the Progress Dynamics Configuration File”](#)

2. The Connection Manager manages the connections to required services, including databases, AppServer, or JMS Server (for SoniqMQ connections only).

The Connection Manager calls a Service Type Manager that manages the physical connections for all session services of the same service type. Each service type has its own Service Type Manager. All Service Type Managers are built with a common starting API to the Connection Manager, allowing it the flexibility to handle any device or service that can be manipulated through that common API.

3. The Session Manager controls the operation of the application.
4. The Dynamics session runs until you shut it down, or until it times out (if the session has a time-out setting specified for it.) The Session Manager handles the termination of the session. First, it shuts down all persistent procedures that are running, then it shuts itself down.

2.3 Configuring a Progress Dynamics session

This section describes how to configure different kinds of sessions, change your existing session configurations, and set up a new configuration.

The following sections provide an overview of the tasks, plus step-by-step instructions:

- [Overview of session configuration tasks](#)
- [Defining session services](#)
- [Creating session properties](#)
- [Defining managers](#)
- [Creating a new session type](#)
- [Generating the configuration file](#)
- [Using minimum XML files](#)
- [Creating a shortcut icon to start a session](#)
- [Importing a configuration file](#)

2.3.1 Overview of session configuration tasks

When you create a new session type or extend an existing session type, you specify the following:

- The Progress Dynamics managers that need to run for the session type.
- Any session properties for the session type.
- Any required session services. (Session services for running applications typically include a Repository and an application databases referenced as physical and logical services.)

The following section describes, in general, the process for configuring a new session.

To configure a new session:

- 1 ♦ Create logical and physical service records in the Repository for any new session services, such as databases or Web services.
- 2 ♦ Create records in the Repository for any new session properties.
- 3 ♦ Create records in the Repository for any new managers.
- 4 ♦ Create a new session type in the Repository.
- 5 ♦ Generate or regenerate your configuration XML file.
- 6 ♦ Create a shortcut icon that uses your configuration XML file and that points to the configuration information you specified for the session type.
- 7 ♦ You might later import your generated configuration XML file after making any manual changes to it that you want synchronized with the Repository.

2.3.2 Defining session services

A *session service* is made up of a logical and a physical service, both of which have the same service type. A *service type* is a named object with associated management objects that control all services of a particular type. For example, a session service for database access would have the logical and the physical services defined with the database service type. Each service type has its own Service Type Manager that the Connection Manager calls for services of that type.

A *logical service* lets you define the more abstract characteristics of a service type, which provides greater flexibility in service definition to handle various local and remote configurations of the same physical services. A logical service is a separate process that runs locally or remotely and requires connection parameters to establish communication between a session and a service. Logical service names must be unique so that connection to the service can be completely abstracted from the developer. The logical service and session type combination determines which physical service will be used.

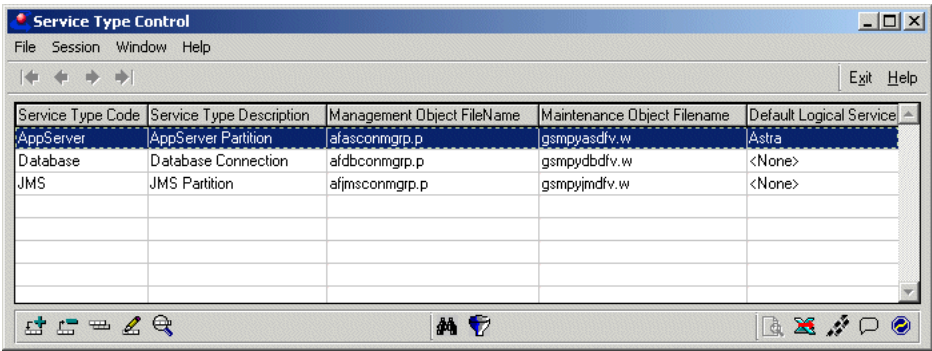
The *physical service* determines the actual connection parameters. For example, consider an application deployed to several sites. The application has a single logical service for its application database. But, there is a different physical service for each site that describes how to connect to the application database on that site's network. To configure the session type on each site, you only need to change which physical service is listed in the session type's description.

Service types

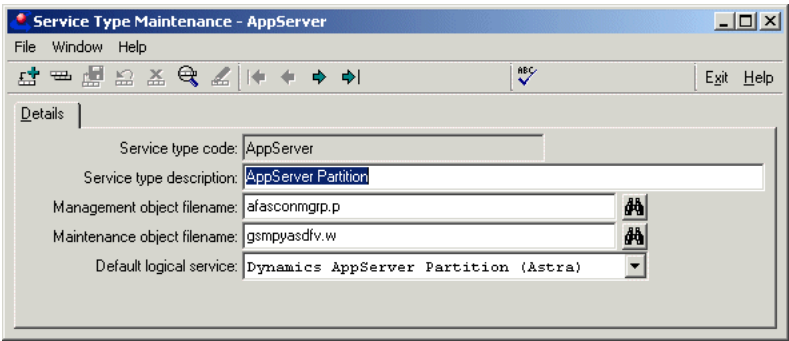
You can view the service types that are registered in the Repository from the Service Type Control window.

To view and edit service types:

- 1 ♦ From the Administration window, choose **Session→Service Type Control**. The **Service Type Control** window appears. You can see the pre-defined service types (AppServer partition, Database, Web service, and JMS Partition) and their associated manager and maintenance files:



- 2 ♦ To edit a service type, double-click it in the browse. The **Service Type Maintenance** window appears:



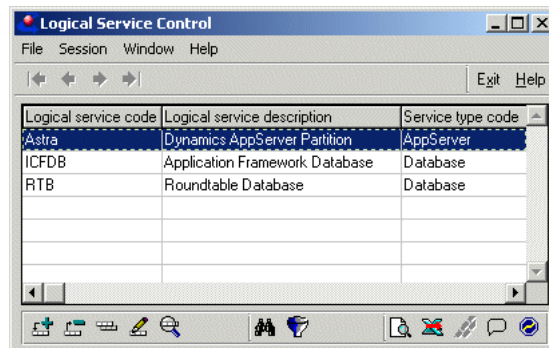
You can build and define your own service types by creating Service Type Mangers that adhere to Progress Dynamics requirements. See the [Progress Dynamics Programming Handbook](#) for more information on creating managers.

Creating a logical service

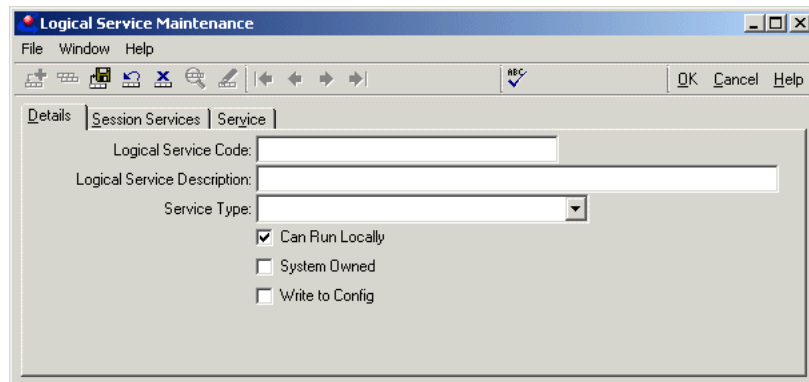
By defining a logical service, you can change the physical service when you deploy an application without changing any code. For example, your code can reference the logical service called DynSports. The physical service name can be dynsportsn for a network service. If you change the physical service to dynsportsl for a local connection to the database, you do not have to change your code because the logical service name remains DynSports.

To create a logical service:

- 1 ♦ From the Administration window, choose **Session→Logical Service Control**. The **Logical Service Control** window appears:



- 2 ♦ Choose **Add record** . The **Logical Service Maintenance** window appears:



- 3 ♦ Type a logical service code, for example, **DynSports**.

NOTE: In the case of the AppServer, this code replaces the partition name that is used by ADM code written or generated outside of Progress Dynamics. If an application makes use of ADM code that has the partition name coded into the container objects, you must specify these partition names as logical service code values. Also note that if you alter the logical service code after you create it, you might render the earlier ADM code unusable.

- 4 ♦ Type a description of the logical service, for example, **DynSports application database**.
- 5 ♦ Select a service type. In this example, it would be **Database**.
- 6 ♦ Specify the appropriate values for the remaining properties by selecting or unselecting the toggle boxes. [Table 2–2](#) describes the remaining properties.

Table 2–2: Logical service properties

Property	Description
Can run locally	Specifies if the service can run locally. If it cannot run locally (unselected), you must have an entry in the session service table to use the partition.
System owned	Specifies if the service is system owned. System owned services cannot be modified unless you have the appropriate permissions. NOTE: The framework does not enable this feature automatically. To fully implement this feature, you must provide code in the trigger for the table, or the SDO linked to the table, containing the system owned field.
Write to config	Specifies if the service, and its associated physical service, should be included in generated configuration files.

- 7 ♦ Choose **Save** to save the new logical service to the database.

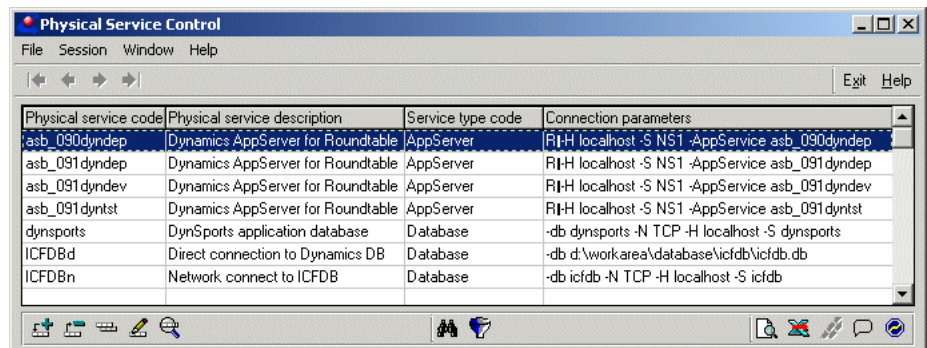
NOTE: The **Logical Service Maintenance** window contains two additional tab folders: Session Services and Service. The Session Services tab displays the session services for the selected logical service, after you associate the new logical service with one or more session services. The Service tab lets you define a session service for the selected logical service.

Creating a physical service

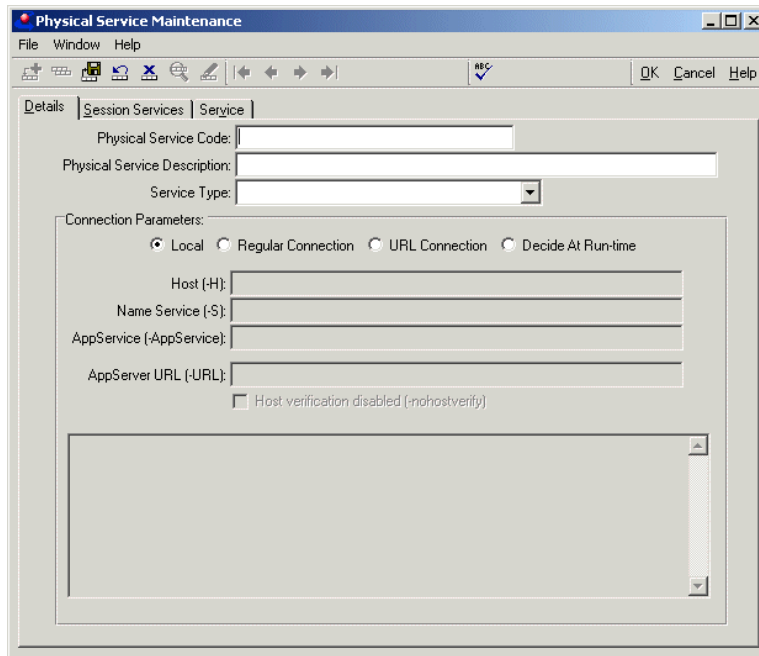
A physical service provides the specific connection parameters that are required to connect a physical device or service to a session.

To create a physical service:

- 1 ♦ From the Administration window, choose **Session→Physical Service Control**. The **Physical Service Control** window appears:



- 2 ♦ Choose **Add record** . The **Physical Service Maintenance** window appears:



The **Physical Service Maintenance** window is a standard Windows-style application window. It has a title bar with the text "Physical Service Maintenance" and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with "File", "Window", and "Help". Under "File", there are icons for adding, deleting, and saving records, along with navigation arrows. The main area of the window is divided into three tabs: "Details", "Session Services", and "Service". The "Service" tab is currently selected. Within the "Service" tab, there are several input fields: "Physical Service Code:" (a text box), "Physical Service Description:" (a larger text box), and "Service Type:" (a dropdown menu). Below these is a section titled "Connection Parameters:" which contains four radio buttons: "Local" (selected), "Regular Connection", "URL Connection", and "Decide At Run-time". Under the "Local" radio button, there are four text boxes labeled "Host (-H):", "Name Service (-S):", "AppService (-AppService):", and "AppServer URL (-URL):". Below these text boxes is a checkbox labeled "Host verification disabled (-nohostverify)". At the bottom of the window is a large, empty rectangular area, likely for a list of services or a log.

- 3 ♦ Type a physical service code, for example, **dynsports**.
- 4 ♦ Type a description, for example, **DynSports application database**.
- 5 ♦ Select a service type. In this example, it would be **Database**.
- 6 ♦ Specify the appropriate connection parameters.

Depending on the service type you selected, a different set of connection parameters are displayed. For more information on connection parameters, see the online help and/or the *Progress Startup Command and Parameter Reference*.

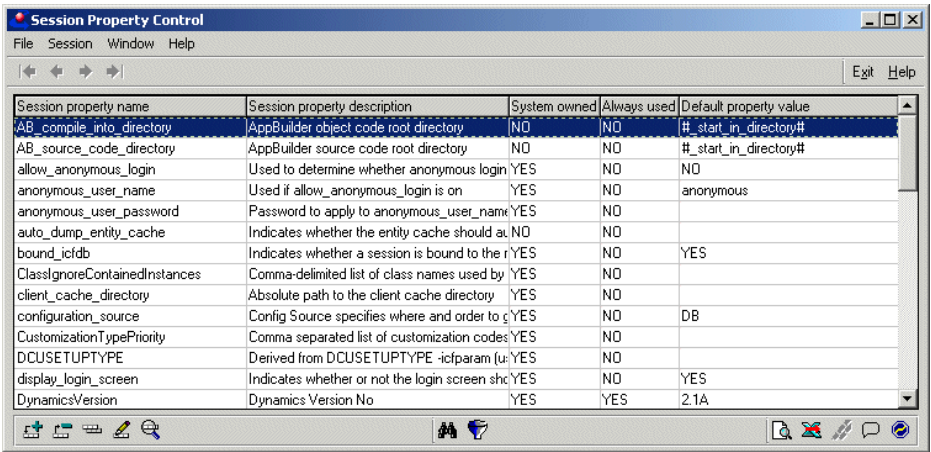
- 7 ♦ Choose **Save**.

2.3.3 Creating session properties

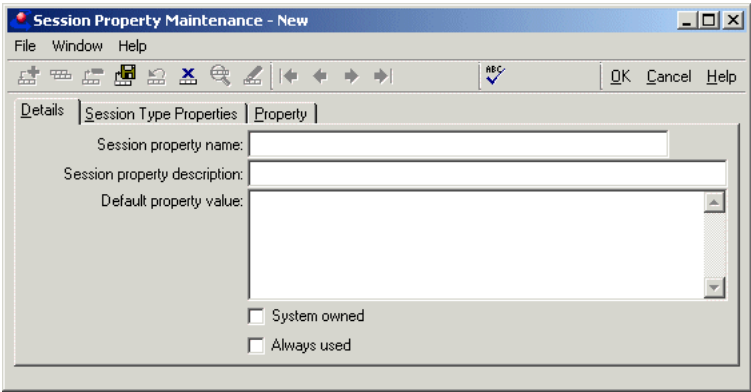
For each session type, you can set multiple session properties. For example, there is a session property that specifies the startup procedure for Progress Dynamics. For a list of all the default session properties, see [Appendix B, "Progress Dynamics Session Properties."](#)

To view and create session properties:

- 1 ♦ From the Administration window, choose **Session→Session Property Control**. The **Session Property Control** window appears:



- 2 ♦ Choose **Add record** . The **Physical Service Maintenance** window appears:

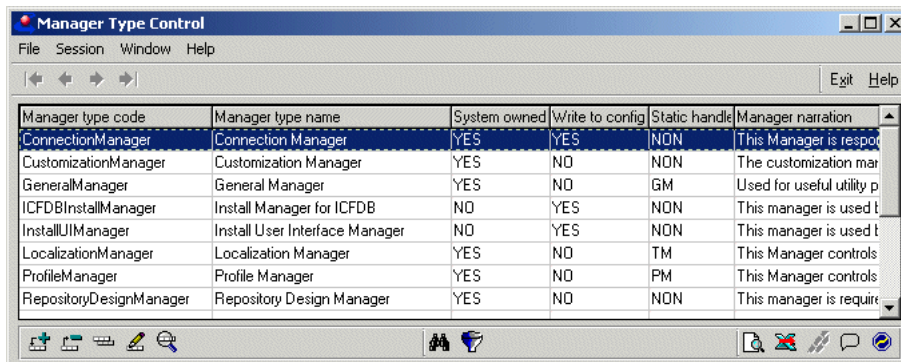


- 3 ♦ Type a **Session property name**.
- 4 ♦ Type a **description**.
- 5 ♦ Specify the **Default property value**.
- 6 ♦ Select the **System owned** toggle box to restrict who can alter the session property.
- 7 ♦ Select the **Always used** toggle box to have the session property written to the configuration file whether or not there is a session type that uses it.
- 8 ♦ Choose **Save**.

2.3.4 Defining managers

In addition to session properties and session services, you need to specify any managers that are used by a session type. A predefined set of managers are included with the Progress Dynamics installation. A manager is a persistent procedure that directly affects the core functionality of the framework. For example, the Session Manager lets you start other code, the Security Manager lets you define user authentication, and so on. If you create any custom managers that similarly affect the framework, you must register them in the Repository to use them with the framework.

To view the managers registered in the Repository, select **Session→Manager Type Control**. The **Manager Type Control** window appears:



Manager type code	Manager type name	System owned	Write to config	Static handle	Manager narration
ConnectionManager	Connection Manager	YES	YES	NON	This Manager is respon
CustomizationManager	Customization Manager	YES	NO	NON	The customization man
GeneralManager	General Manager	YES	NO	GM	Used for useful utility p
ICFDBInstallManager	Install Manager for ICFDB	NO	YES	NON	This manager is used t
InstallUIManager	Install User Interface Manager	NO	YES	NON	This manager is used t
LocalizationManager	Localization Manager	YES	NO	TM	This Manager controls
ProfileManager	Profile Manager	YES	NO	PM	This Manager controls
RepositoryDesignManager	Repository Design Manager	YES	NO	NON	This manager is requir

NOTE: The Configuration File Manager does not appear in the list of configurable manager types in the Manager Type Control window. This management procedure is at the core of session management, but because it is the manager responsible for initiating the process of starting the managers on this list, its configuration is built into the framework and you should never change it.

For details about how to use the existing managers as well as how to create a new manager, see the *Progress Dynamics Programming Handbook*. For more information about the application programming interfaces (APIs) of these managers, see the *Progress Dynamics Managers API Reference*.

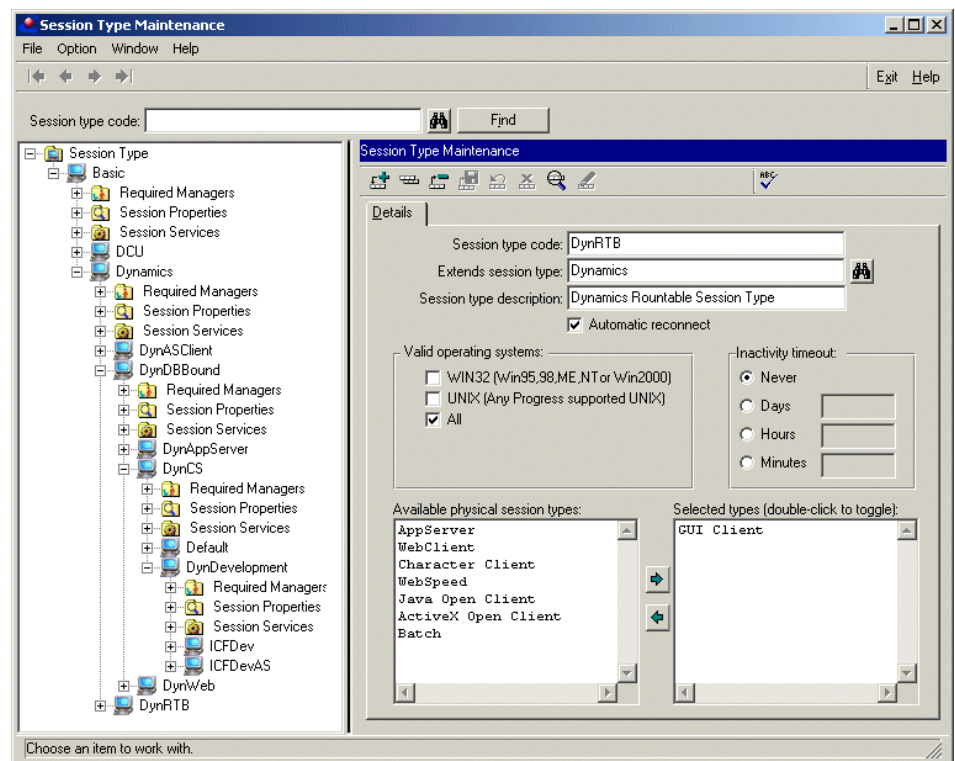
2.3.5 Creating a new session type

Because session types are designed to inherit settings from other session types, the preferred method for creating new session types is to extend an existing session type. When you create a new session type in this way, you only have to specify the new items or changed values rather than everything that the session needs to run.

As an example, we will create a session type for the DynSports sample application by extending the Default session type.

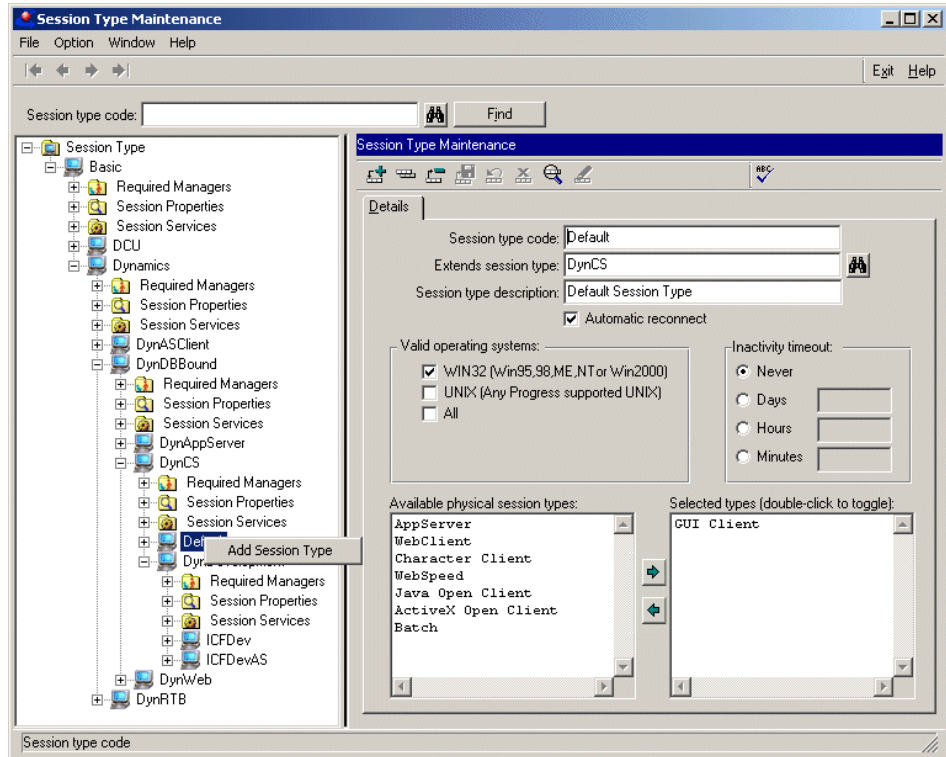
To create a new session type:

- 1 ♦ From the Administration window, choose **Session→Session Type Control**. The **Session Type Maintenance** window appears:



Note the hierarchy of the session types. The standard development session (ICFDev) inherits from DynDevelopment, which inherits from DynCS, which inherits from DynDBBound, which inherits from Dynamics, which inherits from the Basic session type.

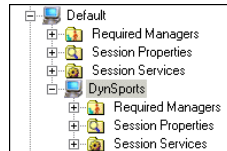
- 2 ♦ Right-click on the **Basic→Dynamics→DynDBBound→DynCS→Default node**, and choose the **Add Session Type** pop-up option:



Session Type Maintenance appears in the right frame.

- 3 ♦ Type **DynSports** in the Session type code field.
- 4 ♦ Type **DynSports sample application session** in the Session type description field.
- 5 ♦ Unselect the **Automatic reconnect** toggle box, if necessary. You set this option when you want the framework to automatically attempt to reconnect a dropped AppServer connection and restore the session context.
- 6 ♦ Select the **WIN32** option in the Valid operating systems choices.

- 7 ♦ Choose **Never** for the Inactivity timeout options. The session will keep running until the user actively shuts it down.
- 8 ♦ Leave the physical session type selections as they are. You do not need to add any for this session type.
- 9 ♦ Choose **Save**. The new session type displays under the **Default** node:



Defining managers for a session type

Now that you have a new session type, you need to modify it to suit your needs. Under each session type, there are three nodes that enable you to configure the session. The first node is the Required Managers node. If your new session type requires a manager that is not defined in one of the session types from which it inherits, you need to add it through the Required Managers node.

Figure 2–2 shows the managers that our example session type, DynSports, inherits.

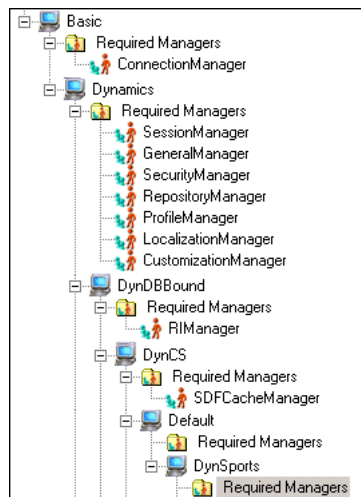
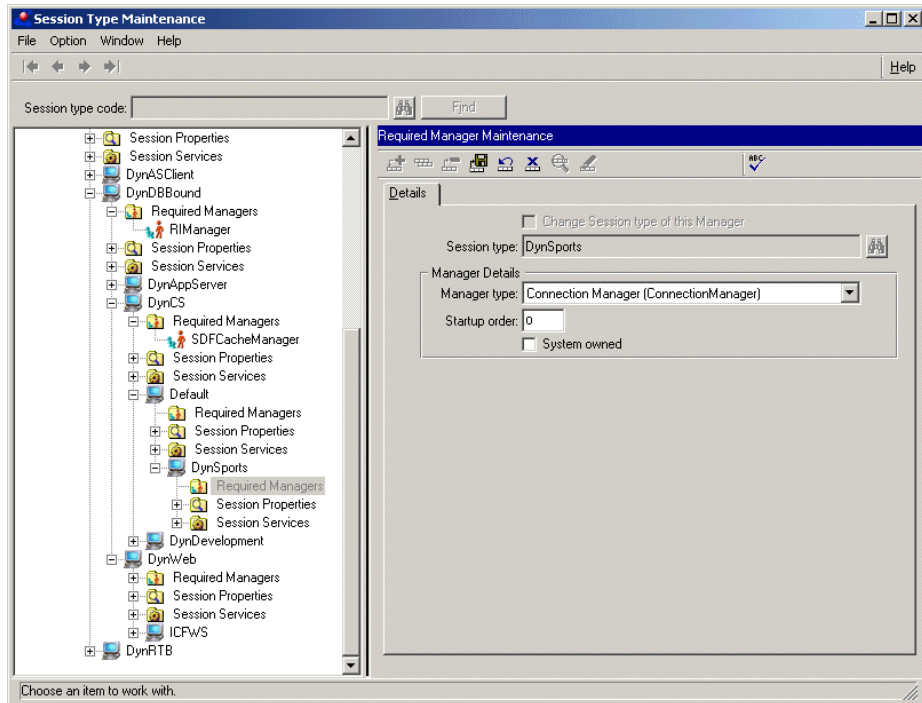


Figure 2–2: Required Manager inheritance

Suppose you wanted to turn the DynSports session type into a development session. You would need to add the Repository Design Manager to it.

To define a manager for a session type:

- 1 ♦ Expand the **DynSports** session type node.
- 2 ♦ Right-click on the **Required Managers** node, then choose the **Add Required Manager** pop-up option. **Required Manager Maintenance** appears in the right frame:



- 3 ♦ Select **Repository Design Manager** from the Manager type list.
- 4 ♦ Type **11** for the Startup order. This is the order in which the Connection Manager starts the necessary managers.
- 5 ♦ Unselect the **System owned** toggle box, if necessary. You set this option when the manager can only be modified by users with appropriate privileges.
- 6 ♦ Choose **Save**.

For more information on Progress Dynamics managers, see the [“Defining managers”](#) section.

Defining properties for a session type

The second node that enables you to configure a session type is the Session properties node. If your new session type requires properties that are not defined in one of the session types from which it inherits, you need to add it through the Session properties node. You might also want to override the value of an inherited property. You do this by adding the property to the new session type and assigning it a new value.

Figure 2–3 shows the session properties that our example session type, DynSports, inherits.

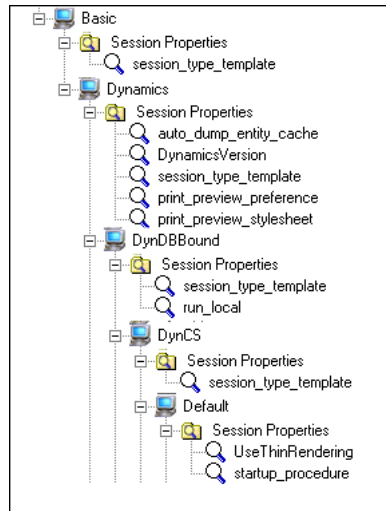
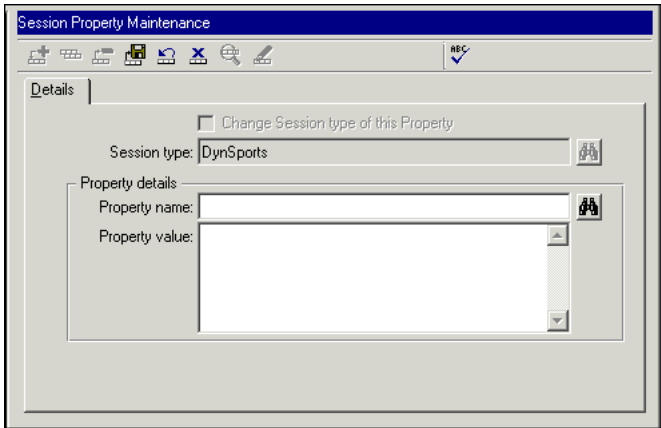


Figure 2–3: Session property inheritance

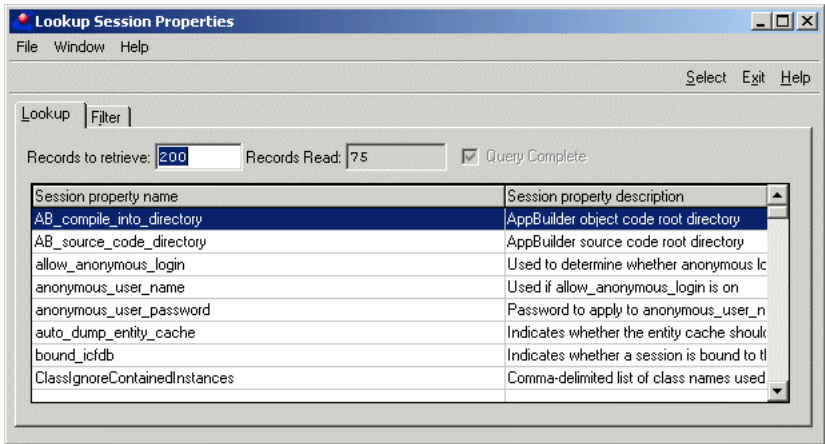
Let us continue with the example of turning the DynSports session type into a development session. There are several session properties that are set for development sessions. You can look at the session properties for the DynDevelopment session type to see them all. We will set one of them for DynSports.

To define session properties for a session type:

- 1 ♦ Expand the **DynSports** session type node.
- 2 ♦ Right-click on **Session Properties**, then choose the **Add Session Property** pop-up option. The **Session Property Maintenance** window appears in the right frame:



- 3 ♦ Choose the lookup button for **Property name**. The **Lookup Session Property** dialog box appears:



- 4 ♦ Double-click on **IDEPalette**.
- 5 ♦ Type **PaletteDynamics** for the **Property value** field.
- 6 ♦ Choose **Save**.

For more information on Progress Dynamics session properties, see the [“Creating session properties”](#) section.

Defining services for a session type

The third node that enables you to configure a session type is the Session services node. If your new session type requires a session service that is not defined in one of the session types from which it inherits, you need to add it through the Session properties node. You might also want to override the value of an inherited service. You do this by adding the service to the new session type and assigning it a new value.

[Figure 2–4](#) shows the session services that our example session type, DynSports, inherits.

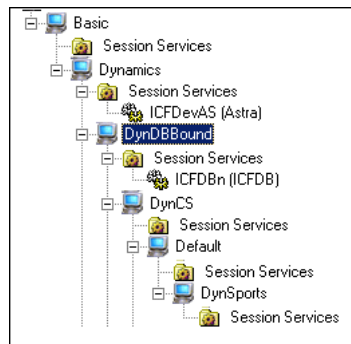
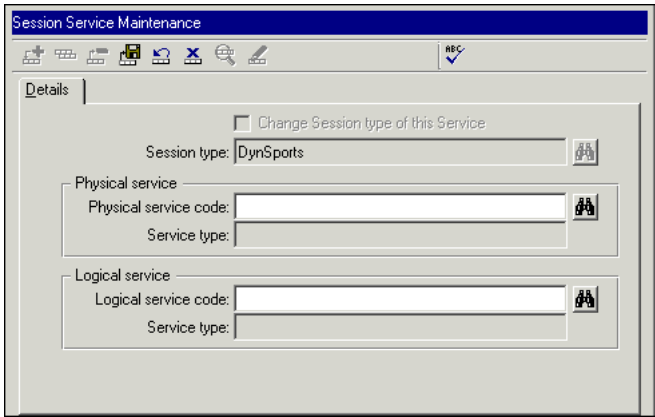


Figure 2–4: Session services inheritance

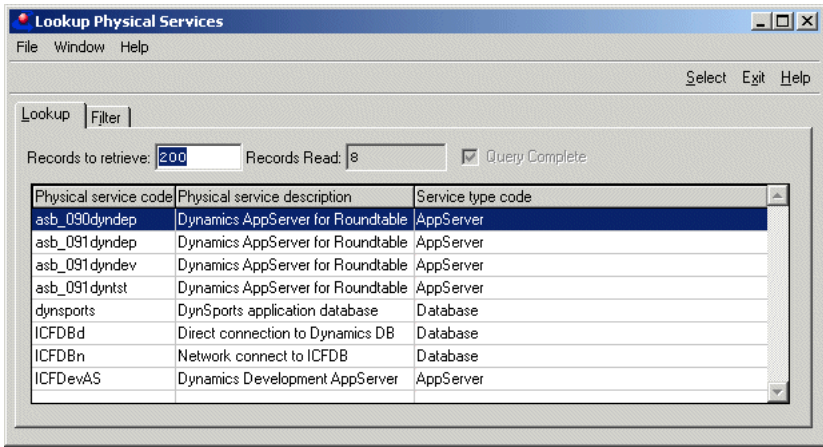
As shown in [Figure 2–4](#), our DynSports session type inherits only two services, the default AppServer partition (ICFDevAS) and a network Repository connection (ICFDBn). To run the sample application, the DynSports session type needs a connection to the DynSports database. You can create that connection using the logical and physical services that you created in the [“Creating a logical service”](#) section and the [“Creating a physical service”](#) section.

To define session services for a session type:

- 1 ♦ Expand the **DynSports** session type node.
- 2 ♦ Right-click on the **Session Services** node, then choose the **Add Session Service** pop-up option. The **Session Service Maintenance** window appears in the right frame:

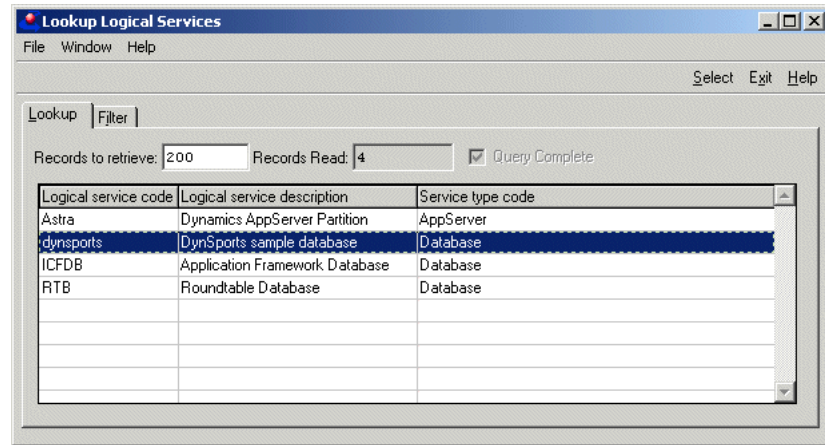


- 3 ♦ Choose the lookup button for **Physical service code**. The **Lookup Physical Services** dialog box appears:



- 4 ♦ Double-click on **dynsports**.

- 5 ♦ Choose the lookup button for **Logical service code**. The **Lookup Logical Services** dialog box appears:



- 6 ♦ Double-click on **dynsports**.

- 7 ♦ Choose **Save**.

For more information on Progress Dynamics services, see the [“Defining session services”](#) section.

2.3.6 Generating the configuration file

The configuration file defines the environment in which a Progress Dynamics installation can run. The definition of each session type includes the following startup information:

- All of its required startup parameters (session properties).
- Its logical and physical resources (session services).
- The management procedures (Progress Dynamics managers) with which it runs.

The configuration file is used solely by the Configuration File Manager. The main purpose of the configuration file is to allow a centralized point where you can place various different sessions’ configuration parameters so that they are accessible anywhere on the network. Multiple clients can use the same configuration file. The configuration file is an XML file that can be read across the network because of built-in features of the XML Document Object Model (DOM).

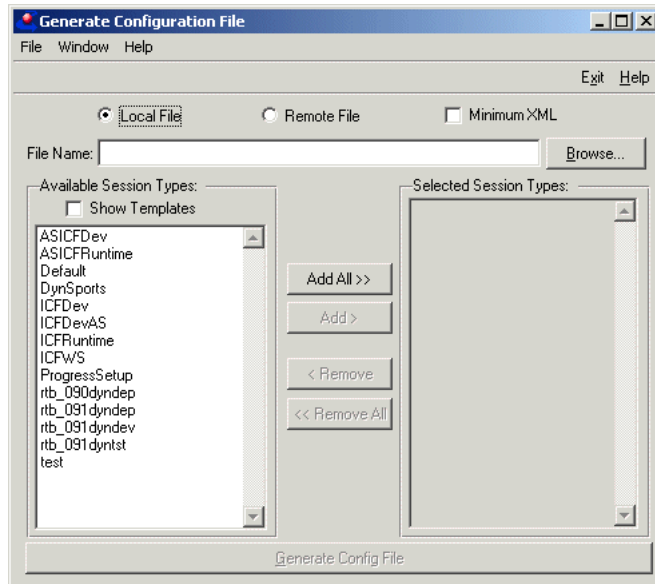
When you install Progress Dynamics, the Progress Dynamics Configuration Utility (DCU) places a default configuration file called `icfconfig.xml` in `Dynamics-Install-Dir/src/icf`. (For a detailed description of the contents of the default `icfconfig.xml` see [Appendix C, “Inside the Progress Dynamics Configuration File.”](#)) This default configuration file lets you initially start Progress Dynamics so you can access the Progress Dynamics tools to define your own sessions and create your own configuration file.

When you start Progress Dynamics for the first time, you should create a new configuration file in your working directory. In general, you should not edit your configuration XML file manually (in an XML or text editor). You should instead use the framework’s administration tools to generate the files. The tools allow you to include only the session types you want in a given configuration file.

When you save the information that you add or change in the Session Type Control window, that information is saved into the Repository. To place those settings into the configuration file, you need to generate (or regenerate) the XML file.

To complete the example, you can generate a configuration file for the DynSports session type. To generate (or regenerate) your configuration XML file:

- 1 ♦ From the Administration window, choose **Session→Generate Configuration file**. The **Generate Configuration** file dialog box appears:



- 2 ♦ Choose **Local File**. Because the configuration file is in XML format, you can deploy it on a local disk or remotely, on a Universal Naming Convention (UNC) share or on a Web server. Generating it for remote use allows you to centralize the administration and set up of your application environment and reduces deployment issues.
- 3 ♦ Type **DynSportsConfig.xml** in the **File Name** field. If you want the file to be stored somewhere other than in the *Dynamics-Install-Dir/src* directory, use the **Browse** button to set the appropriate path. The directory should be listed in your **PROPATH**.
- 4 ♦ Unselect the **Minimum XML** toggle box, if necessary. The **Minimum XML** setting is a security measure. For more information, see the [“Using minimum XML files”](#) section.
- 5 ♦ Select **DynSports** in the **Available Session Types** list, then choose **ADD>** to move it to the **Selected Session Types** list.
- 6 ♦ Choose the **Generate** button. Progress Dynamics only selects the session types in the **Selected Session Types** list to include in the generated configuration file.
- 7 ♦ When the message stating that the configuration file was saved successfully appears, choose **OK** to dismiss the message dialog box.

2.3.7 Using minimum XML files

A *minimum XML file* is a generated configuration XML file that contains the minimum data required to make the connection to the Repository. The rest of the configuration data for the session is read from the Repository when the session starts. You can generate these minimum XML files by selecting the Minimum XML toggle box in the Generate Configuration File tool (see the [“Generating the configuration file”](#) section).

By requiring the session to get most of its configuration data from a secure Repository, unauthorized users cannot easily start up and gain access to or make unauthorized modifications to the session by modifying the configuration file alone. Note that generating a minimum XML file requires the session data to be configured so it *can* be retrieved from the Repository. In other words, you must configure each of the session types that have a minimum XML file using one of the following configuration settings:

- A configuration_source session property setting that allows retrieval of the session data from the Repository.
- Appropriate settings for the Session Type Control Data window accessed from the Session menu of the Administration tool.

Specifying a minimum XML file significantly reduces the amount of information that is written to a generated configuration XML file. The information generated to a minimum XML file includes:

- **Session properties** — A list of essential session properties that is hard-coded into the configuration file generation procedure.
- **Required managers** — Only those managers where the Write to config toggle box in the Manager Type Control window has been selected.
- **Session services** — Only those services where the Write to config toggle box in the Logical Service Control window has been selected.

The default session types installed with Progress Dynamics only generate a manager entry for the ConnectionManager and only generate services for ICFDB and Astra.

For more information on specifying and configuring sessions types to use minimum XML files, see the session management white papers posted on the the following PSDN Web site:

<http://psdn.progress.com>

2.3.8 **Creating a shortcut icon to start a session**

To create a shortcut icon to start Progress Dynamics, you need to specify startup parameters and properties that:

- Specify the session type.
- Specify the location of your configuration XML file.

Use the -icfparam startup parameter, with the qualifiers described in [Table 2–3](#).

Table 2–3: Qualifiers for the -icfparam startup parameter

Qualifier	Description	Default value
ICFCONFIG	Specifies the XML file to use for startup.	icfconfig.xml in your PROPATH.
ICFSESSTYPE	Specifies the session type.	Default session type.

Copying a shortcut

Use the Progress Dynamics shortcut icons as a starting point.

To copy shortcut icons:

- 1 ♦ Right-click on the **Dynamics Runtime** icon from the Windows Start menu, then choose **Copy** from the pop-up menu.
- 2 ♦ Paste the copy of the icon on your desktop or wherever you want to create your new shortcut.
- 3 ♦ Right-click on the icon and choose the **Rename** pop-up menu option.

Setting up shortcuts that use a local configuration file

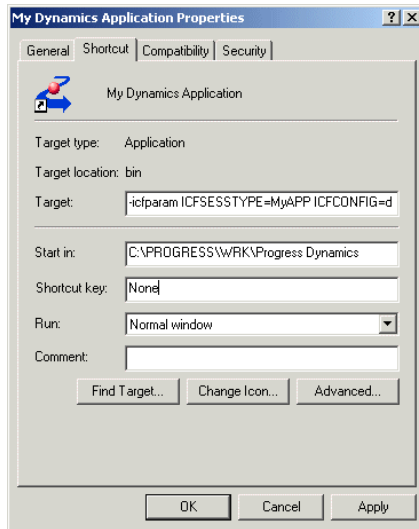
Once you have made a copy of a Progress Dynamics shortcut, you need to edit the properties to specify the appropriate session type and configuration file. This is an example using a configuration file named `dynconfif.xml`, which was locally generated in your working directory.

To edit the properties of the shortcut:

- 1 ♦ Right-click and choose the **Properties** pop-up menu option.
- 2 ♦ In the Properties dialog box's Target field, edit the values of the `-icfparam` startup parameter qualifiers. For example, if the session type you want to use is called `MyAPP` and your configuration file is called `dynconfig.xml`, use the following values:

`-icfparam ICFSESSTYPE=MyAPP ICFCONFIG=dynconfig.xml`

- 3 ♦ Edit the **Start in** field as shown, if needed:



- 4 ♦ Choose **OK**.
- 5 ♦ To test to make sure your session configuration works, start the Repository database and your application database.
- 6 ♦ Double-click on your new shortcut.

Setting up shortcuts that use a remote configuration file

You can have shortcuts that access configuration files on remote machines as long as it is possible to use the Universal Naming Convention (UNC) to access resources on the remote machine.

To create a shortcut that uses a remote configuration file:

- 1 ♦ Copy your configuration file to the location on the network where you want to make it available.
- 2 ♦ Right-click and choose the **Properties** pop-up menu option for your shortcut.

- 3 ♦ In the **Properties** dialog box for the shortcut, edit the -icfparam startup parameter setting for dynconfig.xml to the full UNC path to the file on the network.

For example, if you copy the ICFCONFIG to a server called myserver on a share called configurations in a directory called dynamics, the UNC share name is:

```
//myserver/configurations/dynamics/dynconfig.xml
```

You set the ICFCONFIG qualifier of the -icfparam startup parameter to:

```
ICFCONFIG=//myserver/configurations/dynamics/dynconfig.xml
```

If you deploy the icfconfig.xml on the same server while the server is acting as the Web sever, set the ICFCONFIG qualifier to:

```
ICFCONFIG=http://myserver/configurations/dynamics/dynconfig.xml
```

2.3.9 Importing a configuration file

You can import the Progress Dynamics configuration XML (icfconfig.xml) file into the Dynamics Repository using the Configuration File Import tool (**Session→Configuration File Import** from the Administration tool). This enables the Repository to completely define the session startup environment without the need to maintain a separate configuration file. If you make manual changes to your configuration file, you can easily synchronize them with the Repository by importing the file using this tool. For more information on using the Configuration File Import tool to import your session startup information, see the session management white paper that is posted on the the following PSDN Web site:

```
http://psdn.progress.com
```

CAUTION: Using the Configuration File Import tool can alter the data for a session in the Repository so that the session data is unusable. Before using this tool, be sure to review the documentation in the Session Management white paper for this release.

2.4 Defining user profile data for a session

While session management is primarily concerned with the definition and configuration of Progress Dynamics sessions and the resources that they use, this section describes session data that affect how sessions run based on user run-time settings and actions. These settings depend on profile data that you define using Progress Dynamics and handle in your application code, as well as on profile data that is predefined and that Progress Dynamics can handle automatically for all applications that you build with it. Settings that Progress Dynamics handles automatically from predefined data include common application functions that users might want to change, such as window sizes and positions. In all cases, these settings are based on user profile data that you can modify using the Administration tool.

2.4.1 What is user profile data?

User profile data consists of a two-level definition that includes profile types and profile codes that indicate the exact application function that you allow users to affect. A *profile type* is a large category of profile data that you treat as a group, such as Window. A *profile code* is a subcategory of profile data, such as SaveSizPos, which is a subcategory of Window. You can also specify options in your profile definitions that determine if a profile type applies to client sessions only or also to server sessions, and whether the profile type is enabled (active) or disabled for an entire Progress Dynamics installation.

NOTE: Other than generally distinguishing between client and server sessions, you cannot restrict or assign profile types to the specific session types that you define. That is, there is no relation between Progress Dynamics profile types and session types. The primary relation with Progress Dynamics profile types is a Progress Dynamics registered user. Progress Dynamics thus saves profile data for a Progress Dynamics user across all session types in which the user runs an application on the same Progress Dynamics installation.

2.4.2 How Progress Dynamics uses profile data in a session

The Profile Manager provides the program interface for profile data between the Repository and your application. It provides you with the option to store selected profile data for the duration of a single session or permanently between sessions, which you can have an application user decide. The Profile Manager stores values, by user, for each affected profile code in the Repository. At initial application startup, a default value is used for each setting until the user performs an application function that changes the value, such as changing the size of a window. You can also provide options that let the user decide whether certain profile types and codes affect application behavior.

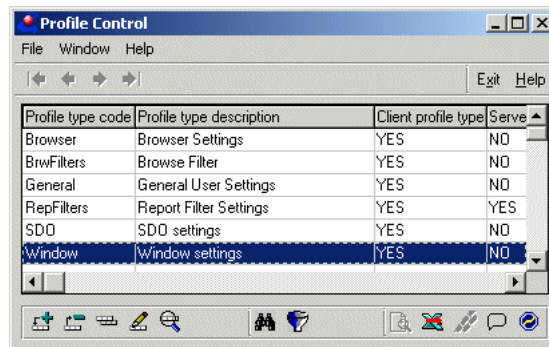
For an example of user profile data in an application, review the Preferences function that is, by default, built into the Administration and Development windows. You can see how it works for the user by changing these settings in the Administration tool or the Development tool. You can provide additional user profile options in your own applications by defining the profile types and codes, as described in this section, and by referencing and updating their values at run time using the Profile Manager API.

NOTE: User profiles are not directly linked to Progress Dynamics security. Instead, Dynamics uses a group-based security model. For more information on how to setup security in Dynamics, see [Chapter 3, “Setting Up Basic Security Options.”](#)

2.4.3 Defining user profiles

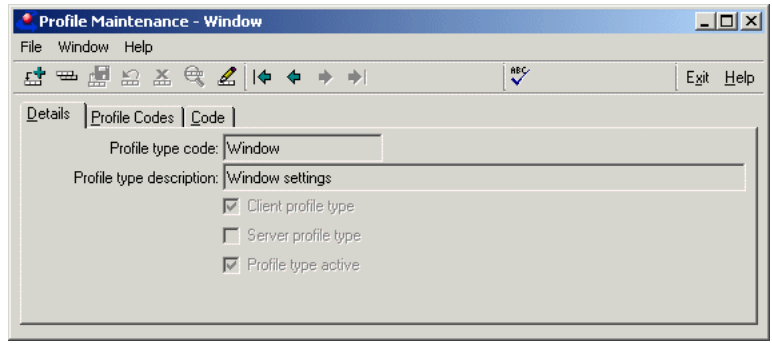
To define user profile types and codes, from the Progress Dynamics Administration window:

- 1 ♦ Choose **Session→Profile Control**. The **Profile Control** window appears:



This window displays information about the existing profile types.

- 2 ♦ To view the details of a particular profile type, double-click on it. The **Profile Maintenance** window appears, displaying the **Profile type code** in the window title:



For information about each of the tab folders and the objects in them, see the online help.

- 3 ♦ To create a new profile, choose the **Add** button from the toolbar.

For information on creating users, see [Chapter 3, "Setting Up Basic Security Options."](#)

Setting Up Basic Security Options

The Progress Dynamics security feature is powerful and flexible. It lets you set security on many types of objects and levels, such as objects, menus, fields, data ranges, actions, and data. You can set up security for specific users or all users, specific companies or all companies, and any combination of the above. In addition to this, you can define restrictions globally or for a specific product module, for a specific object or for an instance of an object. Your application's security policy can be as simple or as complex as you choose.

This chapter provides an overview of security and describes the basic options for managing and securing your Progress Dynamics installation. It includes the following sections:

- [Setting global security and management options](#)
- [Choosing a security model: grant versus revoke](#)
- [Setting up user authentication](#)
- [Defining login companies](#)
- [Defining security groups](#)
- [Creating and maintaining users](#)

For information on security allocations, action security, field security, and data range security, see the chapter on defining security in the *Progress Dynamics Developer's Guide*.

NOTE: Progress Dynamics only secures objects in the application. If you want to secure database access or files in the operating system, see the *Version 9 Progress System Administration Guide* and the *Progress System Administration Reference*.

3.1 Setting global security and management options

When you first start up Progress Dynamics, you might want to set some general options that govern the operation of sessions that run there. A table in the Progress Dynamics Repository (gsc_security_control, entity mnemonic GSCSC) maintains settings in a single record used to control an entire session. These settings include basic options for managing the user interface and security control.

NOTE: All session data, including data used for session startup and security, is typically stored in the Progress Dynamics configuration XML file (icfconfig.xml). You can make this startup and security data more secure, especially if you maintain it outside the Repository, by importing any changes to the configuration file into the secured Repository. For more information, see the sections on importing the configuration file in [Chapter 2, “Defining and Managing Sessions.”](#)

To access the global options for Progress Dynamics session security and management, choose **Security**→**Security Control** from the Progress Dynamics Administration window. Progress Dynamics displays the **Security Control** window, as shown in [Figure 3–1](#).

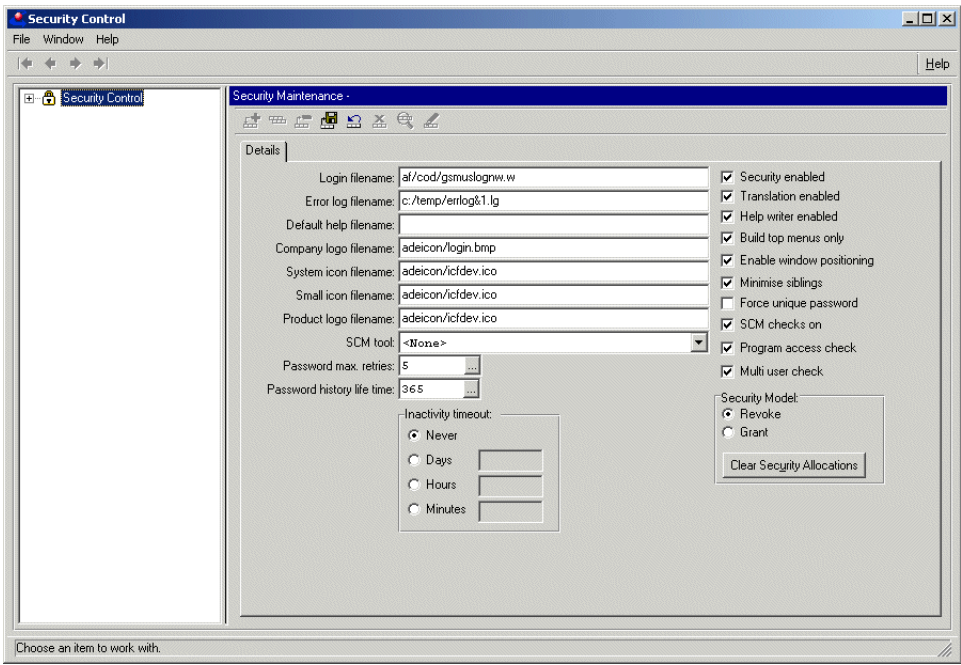
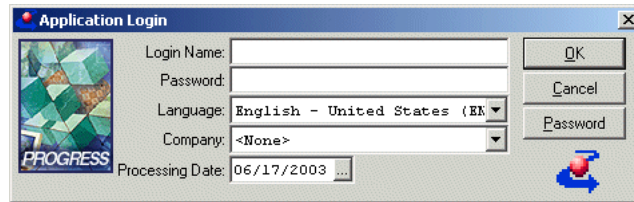


Figure 3–1: Security Control window

NOTE: You can save the changes you make in this window, but because Progress Dynamics allows only one record in the gsc_security_control table, the **Add record** button is disabled in the toolbar. However, you can modify the existing record.

For security, the important settings include:

- **Security model** — Lets you define whether you want to use a revoke or a grant security model for your entire application. For more information on this option, see the [“Choosing a security model: grant versus revoke”](#) section.
- **Login Filename** — The name of the login program to run. By default, this is `af/cod/gsmuslognw.w` and it displays the default login window:



- **Password Max. Retries** — The maximum number of times a user can enter their password incorrectly before the user’s account is disabled.
- **Password History Life Time** — The number of days to keep and check the password history. The default value is 365 (one year). The password history is a log of when and who changed a user’s password. To see the password history for a user, select the **Security Maintenance**→**Users node**, then choose the **Password History** tab.
- **Security Enabled** — Specifies for Dynamics to perform security checks.
- **Force Unique Password** — Specifies that the password for every user must be different from the passwords used by all other users. This option allows Dynamics to locate a user’s record with just a password.
- **Program access check** — Specifies that the system checks whether a program is already running for a user. This restricts the user to running only a single session of a program.
- **Multi user check** — Specifies that the system should ensure a user does not log in more than once. A user can only log in once per user id.

3.2 Choosing a security model: grant versus revoke

Your first step in setting up security for your application is to decide whether to implement a grant or revoke model. As a general rule, if you want to hide a large part of your application system from most of your users, use a grant model. Conversely, if most of your users need to see most of your application, use a revoke model (the default). However, as each application differs, you need to decide what makes sense for your particular application. Consider the following implications of your decision:

- Once you have set up security against a model, you cannot change the model unless you first delete the existing security setup (by clearing all security allocations in your application). Make sure you decide on the correct model before you start to implement security.
- The security model you choose applies to your entire application. As administrator, you can implement either a grant strategy or a revoke strategy for your application, but you cannot combine both in the same application.

The following sections provide more details about each type of security model.

3.2.1 Revoke model

The default security model is the revoke model. Users have access to all functions by default, and you revoke access as necessary.

In a revoke model, you need to revoke access in every security group to which the users are linked. If you do not revoke access in all security groups to which the users are linked, they will have access. If you revoke access differently in different groups, the users get the least restrictive access.

For an example of how the revoke model relates to security groups, see the [“Security groups and the revoke security model”](#) section.

3.2.2 Grant model

By default, in a grant model users have no access, unless you define specifically how they have access.

In a grant model, the users gain access if you have granted them access to any of the groups to which they are linked. If you have granted access in more than one group, the users get the least restrictive access.

In applications where only a small part of the application is visible to the user, the grant security model reduces the effort in setting up security for a new user. Instead of revoking security rights to most of the options in the application, it will only be necessary to grant rights to a small part of the application.

Field security and action security have the potential to create huge numbers of records in the database if you use a grant strategy, as you will need to grant access to every single field and action in the system. To prevent field and action security from becoming unusable, Dynamics only checks them if you have set the applicable security field or action in the database. For example, in the grant model, if you have not set security on a particular field in the database, all users will be able to access that field. As soon as you set up security for the field, you will have to grant access to all users who need to access it. The same rule applies to action security.

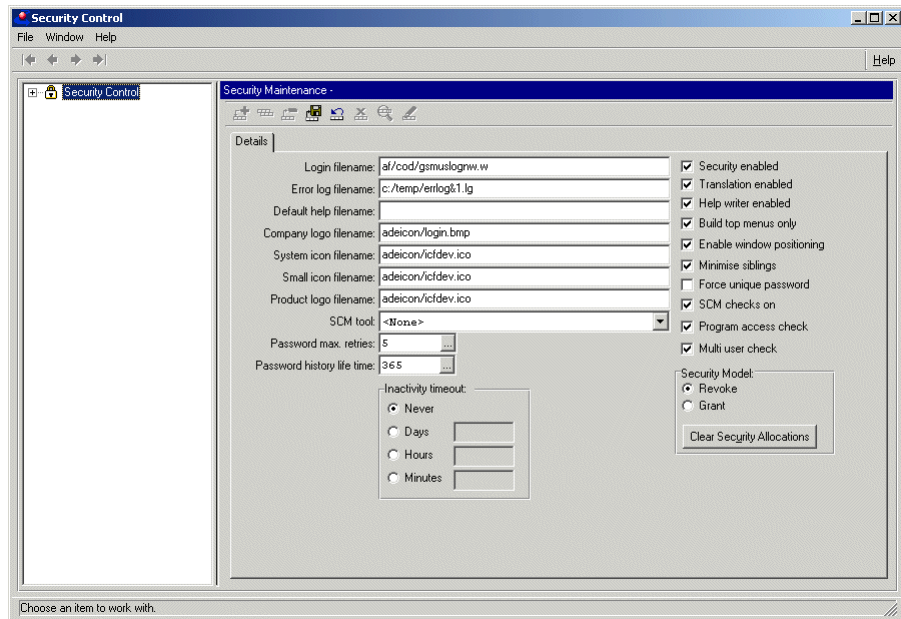
In a grant model, keep in mind that you might have to grant access to more than one object to allow a user to access certain functionality. For instance, for a menu item, the user needs access to the parent menu structure, the menu item itself, and the object that is launched by the menu item. If you do not grant access to all of these objects, the user will not have access to the menu item.

For an example of how the grant model relates to security groups, see the [“Security groups and the grant security model”](#) section.

Specifying a security model

To specify the security model for your application:

- 1 ♦ If you are changing your existing security model, make a backup of your application database.
- 2 ♦ From the Dynamics Administration window, select **Security**→**Security Control**. The **Security Control** window appears:



- 3 ♦ If you are changing your existing security model, choose the **Clear Security Allocations** button. An alert box displays, warning you about the implications of choosing this action.
- 4 ♦ Choose **Yes** in the alert box.
- 5 ♦ In the **Security Allocation** window, specify either **Revoke** or **Grant** as the model you want to use, then choose the **Save** button.

3.3 Setting up user authentication

User authentication establishes the requirements for users to login and run your application. You generally set up user authentication by registering and maintaining users in the `gsm_user` table (entity mnemonic, `GSMUS`). Registering a user includes specifying all the security information that is associated with a user of your application, including such items as login name, password, default login company, and user category.

A user always belongs to some user category (if user categories are defined) and can be a default member of a login company. You must create any user categories and login companies before you use them to register a user. Both of these settings originate from values stored in corresponding Repository tables, as described in this section.

You can also define user profiles by specifying certain types of data to be associated with users of your application in some way. While not strictly a security feature, user profile data are part of the criteria that define a Progress Dynamics user. Thus, this section describes user profile data as part of the user definition process.

To register users of your application:

- 1 ♦ Define user profile data types that your application can use to associate certain session settings with each user.
- 2 ♦ Define any user categories.
- 3 ♦ Define any companies.
- 4 ♦ Define security groups.
- 5 ♦ Create and maintain users.

The rest of this section describes these steps.

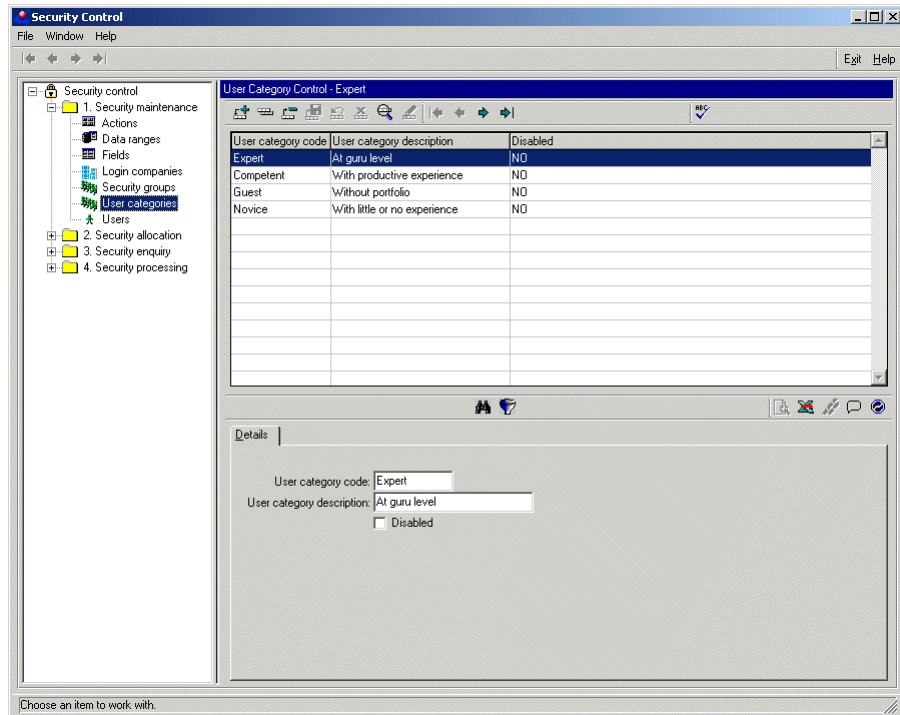
3.3.1 Defining user categories

A *user category* is a value that you define for your application and that you assign to users. If you define any user category values, you must assign one of these values to each user that you register. Your application can use this value for any purpose that it finds useful. Because all users are assigned one of the defined values, your application can respond to all users assigned the same value in the same way. For example, the application might respond in one way for expert users and another way for novice users. You can also have Progress Dynamics restrict all access to users based on their user category.

User categories are stored in the `gsm_user_category` table (entity mnemonic `GSMUC`) and keyed from the `gsm_user` table by `user_category_obj`.

To define or review user categories:

- 1 ♦ In the **Security Control** window, expand the **Security control** node, then expand the **Security maintenance** node.
- 2 ♦ Select the **User categories** node. The **User Category Control** window appears in the right frame:



- 3 ♦ You can select and modify an existing user category or choose the **Add** button from the toolbar to create a new one.
- 4 ♦ Choose the **Save record** button to save your changes.

3.4 Defining login companies

Login companies are organizations in which the user might be a member. When you set up object access restrictions, you can define access restrictions based on the user, the login company, or both in association. Progress Dynamics also uses login company information as part of its automatic reference number (sequence) generation. An example of login company security usage is to manage application access for different corporate clients of an Application Service Provider (ASP). Another example might be to manage users logging into the same corporate financial application on behalf of different subsidiaries.

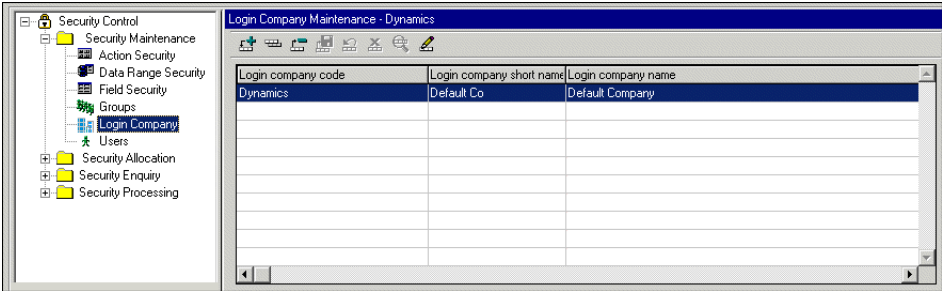
You can register a user to be associated optionally with a specific company by default. Using the standard Progress Dynamics login dialog box, the user can also choose an association from the available companies during log in. Based on the login company field, you can customize your application's appearance and security according to the company the user specifies. Dynamics checks login company security automatically when a user logs into the system. The existence of a security allocation record indicates whether access to the allocated login company is granted.

When you link users to security groups, you can specify the login company to which the security group applies for that user. So even though a security group is valid for all companies, if it is applicable only in a certain company for a certain user. Dynamics only applies security when the user logs into the specified company. If other users are linked to the security group, security is still applied for them for all companies.

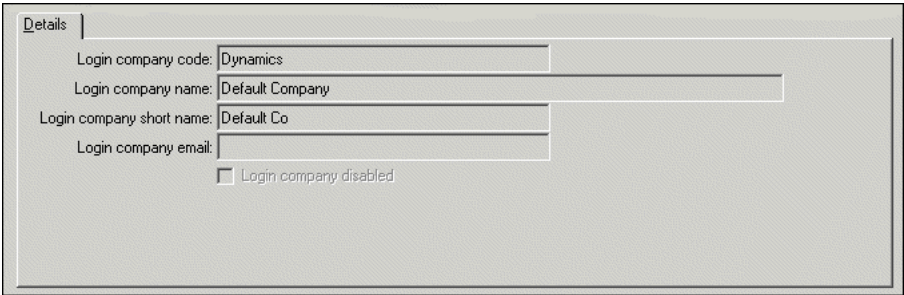
Login companies are stored in the `gsm_login_company` table (entity mnemonic GSMLG) and keyed from the `gsm_user` table by `default_login_company_obj`. The information stored in this table is the minimum to identify a company in the Progress Dynamics environment. Typically, an application stores additional company information, such as address and other contact information, in a database external to the framework. When you integrate such an application with Progress Dynamics, you typically define `gsm_login_company` fields that you can use to key into your application database to access the corresponding company information.

To define or review login companies:

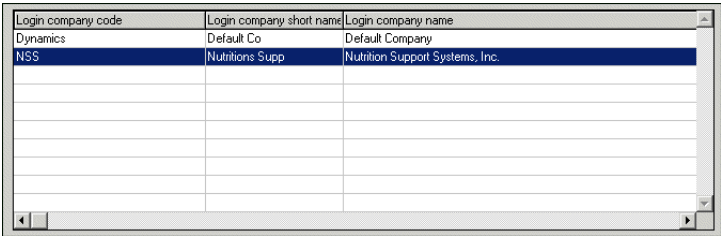
- 1 ♦ In the Security Control window, expand the Security Maintenance node, then choose the **Login Company** node. The **Login Company Maintenance** frame appears:



- 2 ♦ To add a new login company (for example, Nutrition Support Systems, Inc.) choose the **Add record** button. The fields in the **Details** tab become enabled, allowing you to enter the field values for the new login company:



- 3 ♦ Choose the **Save record** button. The new login company immediately displays in the browse of the **Login Company Maintenance** frame:



3.5 Defining security groups

Progress Dynamics implements *security groups*. Security groups allow you great flexibility, permitting you to set up any security structure. You can assign users to multiple security groups, and in turn you can link a security group to one or more other security groups. User permissions are accumulated from all groups a user belongs to, and from all groups to which those groups, in turn, belong.

NOTE: Progress Dynamics does **not** use security hierarchies. In a security hierarchy, you cannot consolidate security groups in different levels in the hierarchy, and you can only assign groups at the bottom of the hierarchy to users. As you must assign each security group to the user individually, a hierarchical approach results in increased administration. A security hierarchy incurs a high performance overhead, as the hierarchy must be navigated every time security rights need to be determined.

Consolidated groups

Consolidated groups are security groups that exist only to group other security groups. You can then assign these consolidated security groups to users. These consolidated groups do not have any security allocations against them directly. Use consolidated groups to simplify security administration. You do not need to flag them in any way. Progress Dynamics assumes that a security group is a consolidated group if it is linked to one or more other security groups but does not have security allocated specifically against it. For example, instead of having to allocate several groups to every user, you can create a consolidated group and link the allocated groups to it. You can then assign just this one consolidated group to every user.

NOTE: Dynamics assumes that security groups with no security allocated against them exist only to consolidate other security groups into one security group. Dynamics does not look at consolidated groups (that is, any group that has no security allocated against it) when it checks security.

Least restrictive rule

Users belonging to multiple security groups need to perform all the actions required in each group. If one group grants less access than another, or revokes access granted in another, the users will not be able to perform all the actions necessary for one of their roles. For this reason, the least restrictive security option always takes precedence, giving the user all access necessary to perform their roles.

If security allocations from different security groups contradict each other, Dynamics uses the least restrictive security allocation. For example, if you have secured a field as Full Access in one security group, and Read Only in another security group, Dynamics will grant Full Access.

In a grant model, the users gain access if they have been granted access in any of the groups they are linked to. If they have been granted access in more than one group, they get the least restrictive access.

In a revoke model, you must revoke access in every group to which the users are linked. If access has not been revoked in all security groups the users are linked to, they will gain access. If access has been revoked differently in different groups, they get the least restrictive access.

Security override rule

Security that you set up at the user-level always overrides security set up at the group-level. This rule lets you set up user-specific exceptions. This rule means that if you allocate any security against the user directly, the system will use that security immediately. The system will not do any further checks against the security groups to which the user belongs.

Where you specify security at the user level, Dynamics ignores the security specified at the security group level.

Security groups and the revoke security model

Figure 3–2 provides an illustration of how security groups work in the Revoke security model.

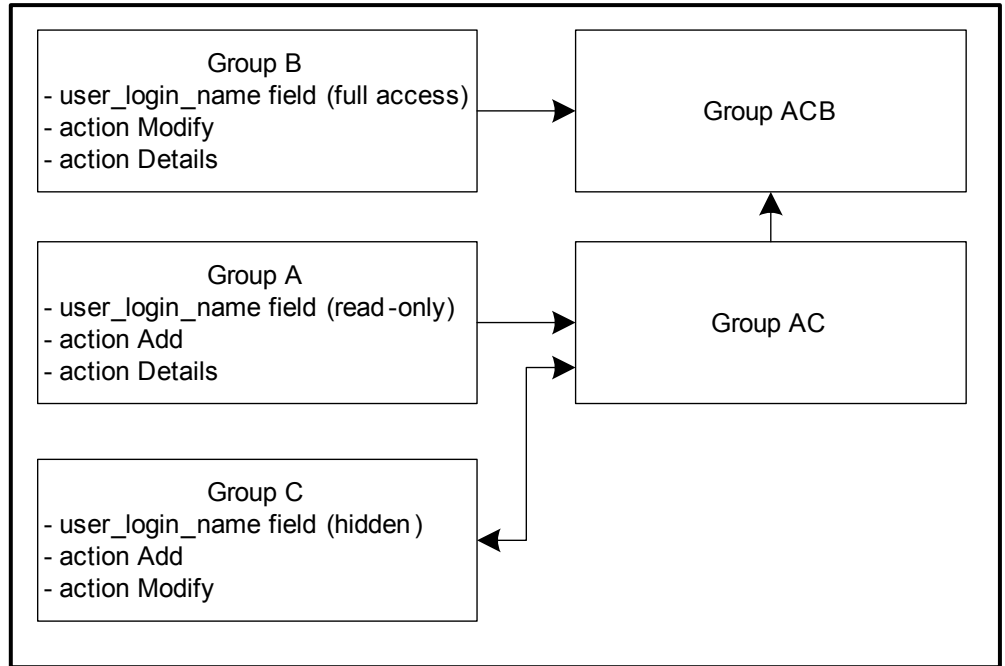


Figure 3–2: Revoke model example

In this revoke model example, Groups A and C are consolidated into Group AC. Groups AC and B are consolidated into Group ACB.

The following scenarios describe how this example revoke security model works:

- A user belonging to Group **B** has full access to field `user_login_name`, but does **not** have access to the `Modify` and `Details` actions.
- A user belonging to Group **AC**, has read-only access to the `user_login_name` field. Because of the least-restrictive rule, the user does **not** have access to the `Add` action, but does have access to the `Details` and `Modify` actions. (Access to an action would have to be revoked in both A and C for it to be revoked in AC.)
- A user belonging to Group **ACB**, has full access to the `user_login_name` field. Because of the least-restrictive rule, the user has access to the `Add`, `Modify`, and `Details` actions.

Security groups and the grant security model

Figure 3–3 provides an illustration of how security groups work in the Grant security model.

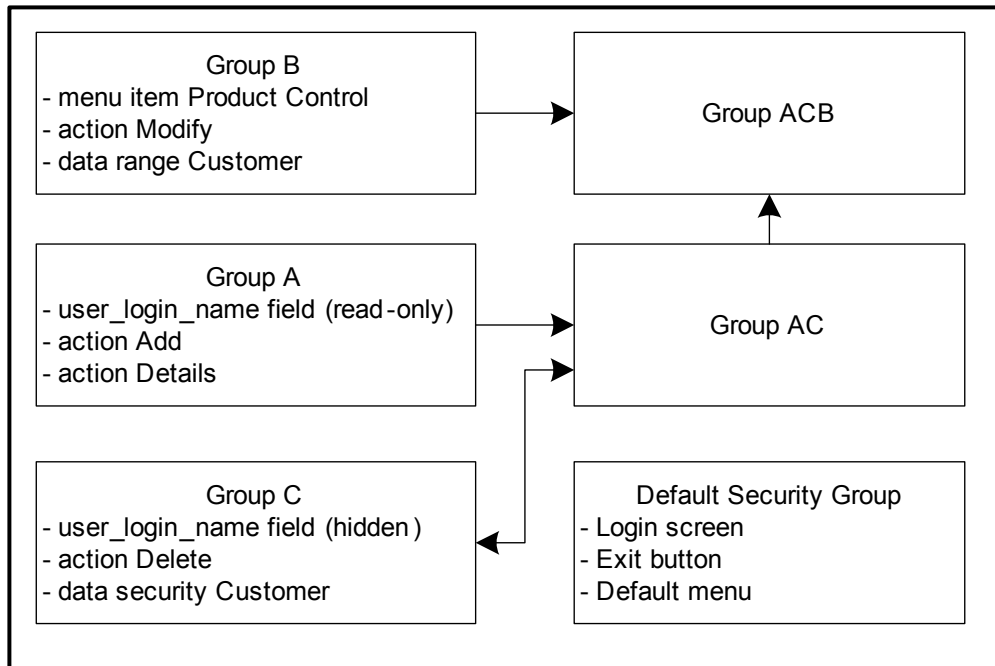


Figure 3–3: Grant model example

In this grant model example, as illustrated in Figure 3–3, Groups A and C are consolidated into Group AC. Groups AC and B are consolidated into Group ACB.

The following scenarios describe how this example grant security model works:

- A user belongs to the **Default Security Group** and **C** security groups has access to:
 - The Login screen, Exit button, Default menu, Field user_login_name (Hidden), Action delete, and data security customer.
 - All actions and fields **not** set up in the General Security Manager (GSM) gsm_token and gsm_field tables.
- A user belonging to the **Default Security Group** and **AC** security groups, has access to:
 - The login screen, Exit button, default menu, user_login_name field (read-only), Action add, Action details, Action delete, and data security customer.
 - All actions and fields not set up in the gsm_token and gsm_field tables.

- A user belonging to only the **ACB** security group has access to:
 - Menu Item Product Control, Action modify, data range customer, user_login_name field (read-only), Action Add, Action Details, Action Delete, and data security customer.
 - All tokens and fields **not** set up in the gsm_token and gsm_field tables.

In this example, this group affiliation would not be of much use, as the user cannot access the login screen. To resolve this problem, you could instead link the user to security group **A** and allocate field security directly against the user record: field user_login_name – Full Access. The user now has access to:

- User_login_name field (full access), Action add, Action details.
- All tokens and fields **not** set up in the gsm_token and gsm_field tables.

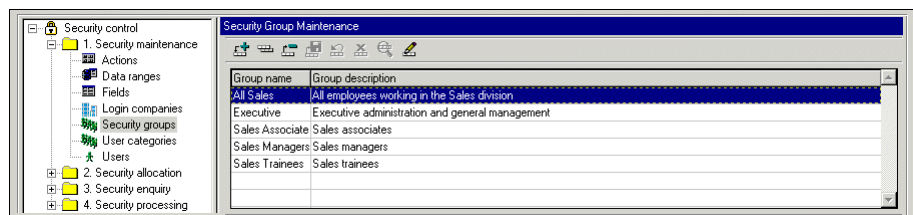
Creating a new security group

To determine how to set up a security group for your application, identify which functions need to be secured routinely for users performing the same function. Then create a security group and secure the applicable objects in that security group. For example, you might create a group for Sales Managers and another group for Sales Representatives.


Avoid trying to set up security based on how you anticipate a security group is going to interact with other security groups. Instead, regard each security group as a totally independent unit, and leave it to the system to accumulate and resolve conflicting security.

To set up a new security group:

- 1 ♦ From the Progress Dynamics Administration window, select **Security**→**Security Control**.
- 2 ♦ In the **Security Control** window, expand the **Security Control** node, then the **Security Maintenance** node.
- 3 ♦ Select the **Security Groups** node. The **Security Groups Maintenance** frame appears:



- 4 ♦ Choose the **Add** button. The fields in the **Details** tab become enabled, as shown:



The screenshot shows a web-based form with a tabbed interface. The 'Details' tab is selected and active. It contains the following elements:

- A tab bar at the top with four tabs: 'Details', 'Users/Groups linked to This Group', 'This Group Linked to Groups', and 'Applies to Company'.
- Below the tabs, there are two text input fields: 'Group name:' and 'Group description:'.
- Below the 'Group description' field, there are two checkboxes:
 - ☐ Disabled
 - ☐ Default security group

- 5 ♦ Enter the name and the description of the group.
- 6 ♦ Choose the **Default security group** toggle box if you want all new users (that is, those you create after you save this toggle box setting) to be linked to this security group.
- 7 ♦ Choose the **Save** button. The new security group displays in the browse at the top of the Security Groups Maintenance frame.

Creating a consolidated group

If you can identify security groups that are all going to be assigned to users performing the same role on a regular basis, consolidate the set of security groups into one security group. Then assign the one consolidated security group to your users.

To set up a consolidated security group:

- 1 ♦ Create a group.
- 2 ♦ Do not specify any allocations against the group.
- 3 ♦ Under the **This group linked to groups** tab, specify the groups you want to consolidate as subgroups of this group.

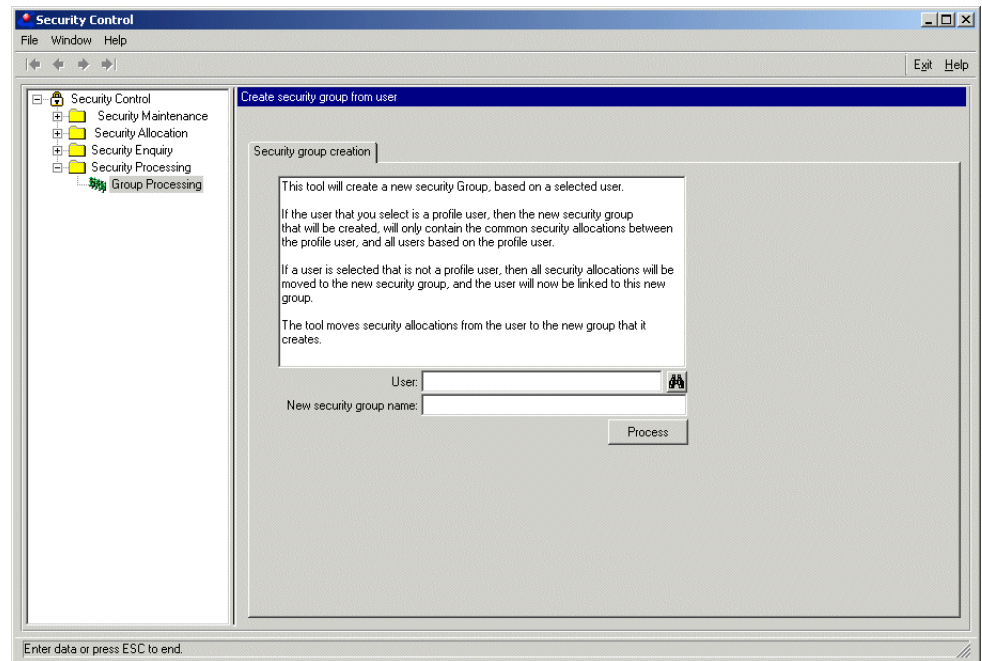
Creating a security group based on an existing user

You can convert an existing user into a security group.

NOTE: A user cannot log in as a security group.

To create a new security group based on a user:

- 1 ♦ From the Progress Dynamics Administration window, select **Security→Security Control**.
- 2 ♦ In the **Security Control** window, expand the **Security Control** node, then the **Security Processing** node.
- 3 ♦ Choose the **Group Processing** node. The **Create security group from user** frame appears:



- 4 ♦ In the **User** field, specify the user on whom you want to base the new security group. The utility will move the security allocations from the user you specify to the new group.

If the user that you specify is a profile user, then the utility moves (to the new group) only those security allocations that the profile user and all users based on the profile user share in common. The utility leaves alone any allocations that are not common. The utility then links the profile user, and all users based on the profile user, to the new group.

If you specify a user who is **not** a profile user, then the utility moves all security allocations from the user to the new security group and then links the user to this new group.

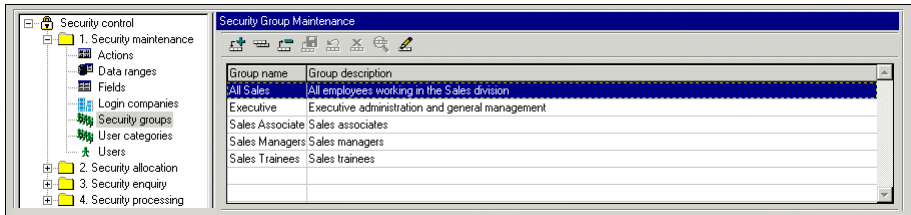
- 5 ♦ Type in the new **security group name**.
- 6 ♦ Choose **Process**.

Specifying default security groups

You can specify default security groups. Determine the default functionality to which every user in your system needs access (for example, the login screen). Allocate this security to a security group, and save it as a default security group. All users in your system will automatically have access to the functionality defined in the default group. Also, when you add a new user to the system, Progress Dynamics automatically links the new user to any default groups.

To create a default security group:

- 1 ♦ In the **Security Control** window, choose the **Security Maintenance→Groups** node. The **Security Group Maintenance** frame appears:



- 2 ♦ Select an existing group or choose the **Add** button to create a new group.
- 3 ♦ In the **Details** tab, check the **Default security group** toggle box, then choose the **Save** button.

Whenever you create a user, Dynamics automatically links the new user to this (and any other) default security group.

Linking a company to a security group

When you create a group, you can link the group to a specific login company. In addition, when you link users to groups, you can specify that a security group only applies when the user logs into a certain company. This functionality lets you specify what functions the user fulfills in different companies. Alternately, you can specify that a security group applies application-wide.

For all security groups you have created, decide if the security group applies to only certain login companies in your application, or all. If only to certain login companies, link the applicable security groups to those companies.

Of the security groups allocated to your users, check if certain security groups only apply to that user when logging into a specific company. Update the security group allocation to indicate which security groups apply to which companies for which user. By default, the security groups allocated to a user will apply for all companies the user logs into.

Linking users to a security group

Decide which security groups apply to which users, then link them. If you want to override security set up in the security group for a single specific user, allocate the security against the user directly. Remember, for security allocated directly to a user, the security set up against the user's security groups is not checked.

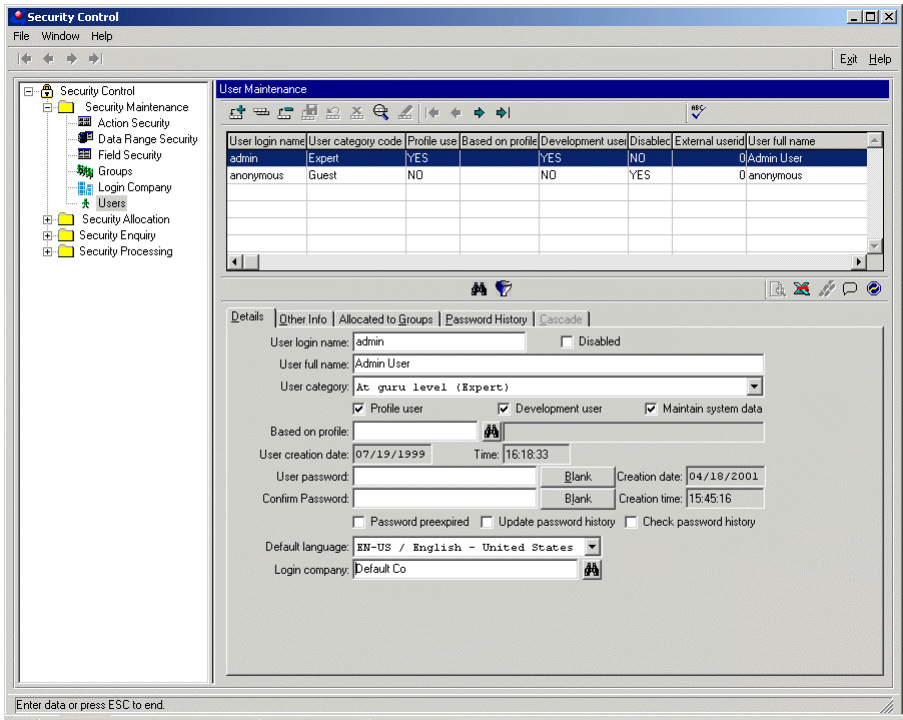
When you link security groups to a profile user, the users based on that profile user will also be linked to the security group.

3.6 Creating and maintaining users

Once you have defined user categories, login companies, and security groups, you can create and maintain users using the Users node of the Security Control window. Through this window, you can set and review various types of profile data and security restrictions that are in effect for a given user. You can also cascade certain profile and security data associated with one user profile to other user profiles that are based on the initial user profile.

To create or review user records:

- 1 ♦ In the **Security Control** window, expand the **Security Maintenance** node, then choose the **Users** node. The **User Maintenance** frame appears:



- 2 ♦ Choose the **Add record** button on the toolbar.
- 3 ♦ For the **User login name**, type the name you want the user to enter when logging into a session. For example, type: **lthomas**. (The login name is not case-sensitive.)

For **User full name**, type the users first and last name. For example, type **Liam Thomas**.
- 4 ♦ Specify a **User category**.

- 5 ♦ Check the one of the toggle boxes:
 - **Profile user** — When selected, you can use this user’s profile as a base for creating other user profiles. If you select this toggle box and then choose the **Save** button, this user’s name will appear in the **Based on profile** lookup dialog box. Any user whose profile is based on this profile can automatically inherit certain profile and security settings. To implement this type of inheritance, you initiate a cascade of the data from the profile user by choosing the **Cascade** button on the **Cascade** tab. See the [“Cascading information from a profile user to individual users”](#) section.
 - **Development user** — When selected, this user can access system development functions.
 - **Maintain system data** — When selected, this user can maintain system control data.
- 6 ♦ If you want to base the new user’s profile and security settings on an existing profile user’s settings, select the **Based on profile** lookup button and select a profile user.
- 7 ♦ Choose a **Language**, such as **EN-US**.
- 8 ♦ Specify a **Login Company**. For example, choose the lookup button to specify **Default Co.**
- 9 ♦ Leave the other fields at their default values or change them as needed. For more information on all the fields in this window, press **F1** to see the online help topic.
- 10 ♦ Choose the **Save** button. The new user appears in the browse at the top of the User Maintenance frame, as shown:

User login name	User category code	Profile use	Based on profile	Development user	Disabled	External userid	User full name
admin	Expert	YES		YES	NO	0	Admin User
anonymous	Guest	NO		NO	YES	0	anonymous
lthomas	Competent	NO		NO	NO	0	Liam Thomas

To make sure that the new user was saved properly:

- 1 ♦ Exit the **Security Control** window.
- 2 ♦ From the **Administration** window, choose **File→Re-Login**.
- 3 ♦ In the **Application Login** dialog box, enter the name of the new user, then choose **OK**.

Verify that you are now logged in as the new user by checking the name in the status bar of the Administration window.

3.6.1 Setting a user's password

To set or change a user's login password:

- 1 ♦ In the **Security Control** window, expand the **Security Maintenance** node, then choose the **Users** node.
- 2 ♦ In the browse of the **User Maintenance** frame, select the user whose password you want to change.
- 3 ♦ In the **Details** tab, choose the **Blank** buttons next to the **User Password** and **Confirm Password** fields to clear them, as shown:

The screenshot shows the 'Details' tab of the User Maintenance window. The 'User login name' is 'lthomas' and 'User full name' is 'Liam Tomas'. The 'User category' is 'With productive experience (Competent)'. The 'Based on profile' is 'admin' and 'Admin User'. The 'User creation date' is '06/19/2003' and 'Time' is '13:25:18'. The 'User password' and 'Confirm Password' fields are empty, with 'Blank' buttons next to them. The 'Creation date' is '06/19/2003' and 'Creation time' is '13:25:18'. The 'Default language' is 'EN-US / English - United States' and 'Login company' is 'Default Co'.

Details Other Info Allocated to Groups Password History Cascade		
User login name:	lthomas	<input type="checkbox"/> Disabled
User full name:	Liam Tomas	
User category:	With productive experience (Competent)	
<input checked="" type="checkbox"/> Profile user	<input type="checkbox"/> Development user	<input checked="" type="checkbox"/> Maintain system data
Based on profile:	admin	Admin User
User creation date:	06/19/2003	Time: 13:25:18
User password:	<input type="text"/>	Blank
Confirm Password:	<input type="text"/>	Blank
	<input type="checkbox"/> Password preexpired	<input type="checkbox"/> Update password history
	<input type="checkbox"/> Check password history	
Default language:	EN-US / English - United States	
Login company:	Default Co	

- 4 ♦ Enter the new password in the **User Password** and **Confirm Password** fields, then choose the **Save** button.

Viewing a user’s password change history

Progress Dynamics maintains a history of password changes in the `gst_password_history` table (entity mnemonic `GSTPH`). You can review this history for each user to see the date and time a password was changed, as well as the user who changed it.

To see information about changes to a user’s password, choose the **Password History** tab in the User Maintenance frame.

3.6.2 Defining a user as a profile user

In addition to security information, each user has associated session profile data that is maintained in the `gsm_profile_data` table (entity mnemonic, `GSMPF`). This includes such information as window sizes and positions, filter criteria for database queries, and any other information that you want maintained for users during application sessions. You define the types of profile data maintained for your application using the Profile Control utility available from the Administration window’s **Session** menu.

For more information on how to maintain profile data types, see [Chapter 2, “Defining and Managing Sessions.”](#) For information on managing profile data in your application, see the *Progress Dynamics Developer’s Guide*.

To define a user as a profile user:

- 1 ♦ In the **Security Control** window, expand the **Security Maintenance** node, then choose the **Users** node.
- 2 ♦ In the **Details** tab, check the **Profile user** toggle box, as shown:

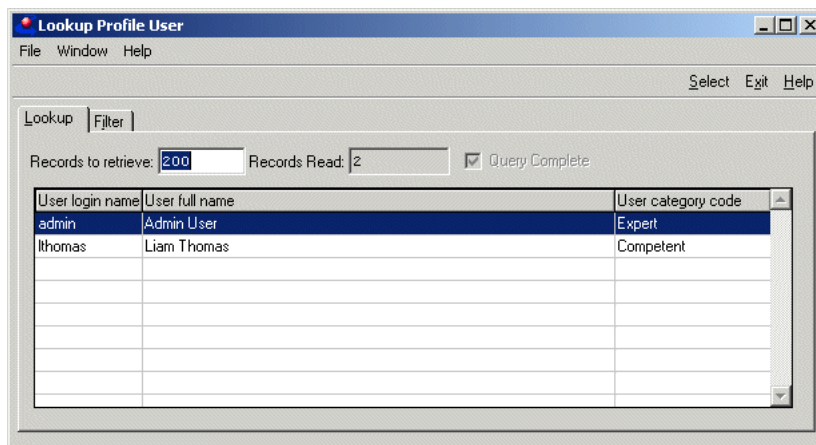
The screenshot shows a user details form with the following fields and values:

- User login name: lthomas
- User full name: Liam Thomas
- User category: With productive experience (Competent)
- Based on profile: admin
- User creation date: 06/18/2003
- User password: Blank
- Confirm Password: Blank
- Default language: EN-US / English - United States
- Login company: Default Co

Additional options and values:

- ☐ Disabled
- ☒ Profile user
- ☐ Development user
- ☒ Maintain system data
- Time: 20:51:53
- Creation date: 06/18/2003
- Creation time: 20:51:53
- ☐ Password preexpired
- ☐ Update password history
- ☐ Check password history

- 3 ♦ Choose the **Save** button in the toolbar.
- 4 ♦ To verify that Liam is now considered to be a profile user, choose the lookup button next to the **Based on profile** field. The **Lookup Profile User** dialog box appears. Notice that Liam Thomas is listed as one of the users in the browse, as shown:



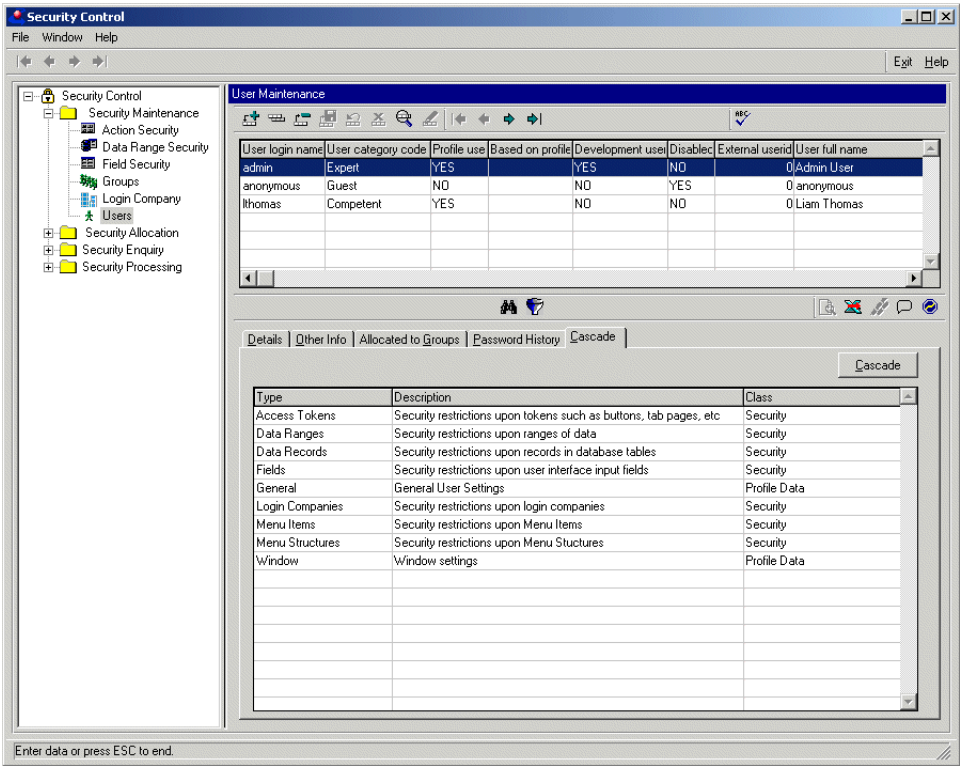
Cascading information from a profile user to individual users

When you update a profile user's security options, you can specify to also update the security options for all users based on that profile user. When you update a profile user, you can specify which security settings you want to cascade down to users based on the profile user.

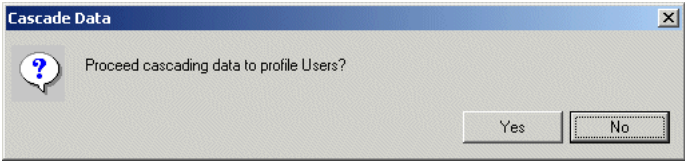
For example, because you have defined Liam's profile to be based on the admin account, the next thing might be to pass certain common user settings to Liam based on the admin account.

To cascade information from a profile user to individual users:

- 1 ♦ In the browse of the **User Maintenance** frame, select the profile user whose security or profile data you want to cascade down to linked users, then choose the **Cascade** tab, as shown:



- 2 ♦ In the **Cascade** tab's browse, select the rows for the settings you want to cascade. (Hold the **CTRL** key to select more than one.)
- 3 ♦ Choose the **Cascade** button to allow all users based on the current profile user to inherit the selected access restrictions and profile data. A confirmation box appears:



- 4 ♦ Choose **Yes** to cascade the settings.

The Cascade operation passes the selected access restrictions and profile data to all users that are based on the current profile user. If a user who inherits the cascaded settings is also a profile user, the settings cascade to all users based on that profile user. This process continues for as many profile users that exist in the cascade tree. The restrictions are thus inherited by all those users as are based on profile users whose profiles are, in turn, directly or indirectly based on the current profile user.

3.6.3 Linking users and security groups

There are two ways to link users and security groups:

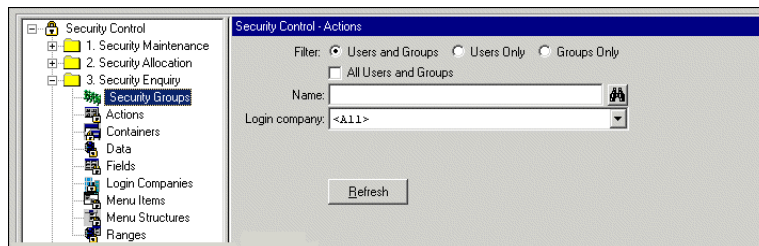
- You can create or update a group and then specify all users that belong to that group.
- You can create or update a user and then specify all the groups to which that user belongs.

Your first step might be to review the security setup as it exists. When setting up user security, you can use the Security Enquiry tool to determine what security is going to be applied to each user.

You can use this tool to determine where user security needs to override group security and only allocate the necessary security allocations to the user. You can query exactly what security would be applicable to a user per login company. Furthermore, you can query all security set up against each object for each security type. For more information on using the Security Enquiry tool, see the [Progress Dynamics Developer's Guide](#).

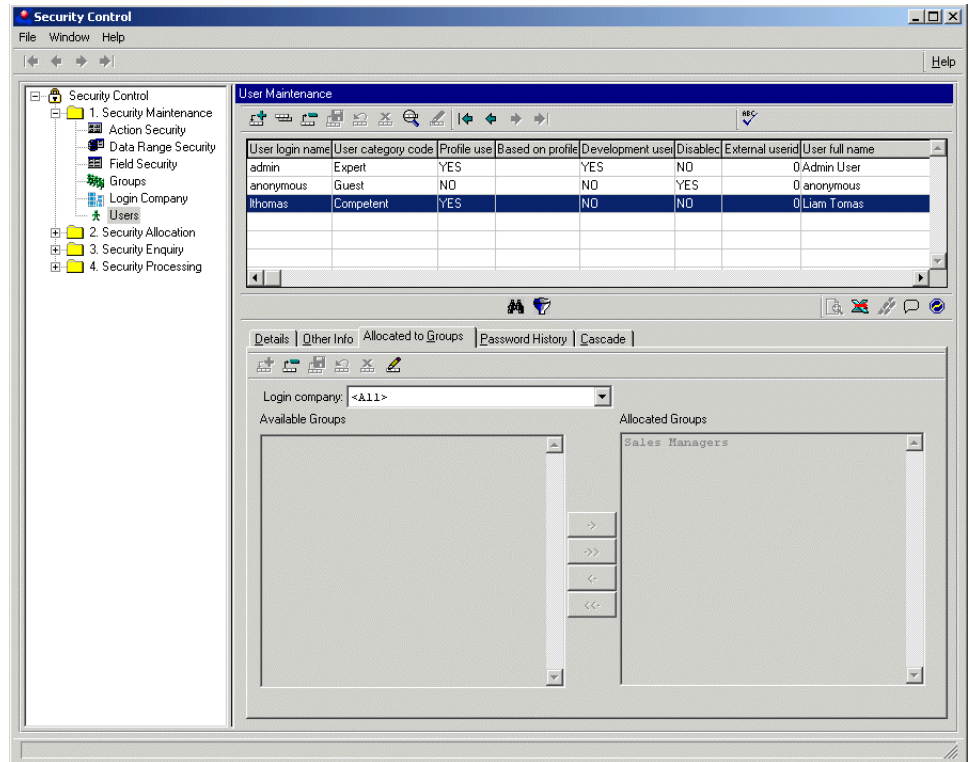
To access the Security Enquiry tool:

- 1 ♦ From the **Security Control** window, expand the **Security Control** node.
- 2 ♦ Expand the **Security Enquiry** node.
- 3 ♦ Choose the **Security Groups** subnode, as shown:



To allocate a specific user to one or more groups:

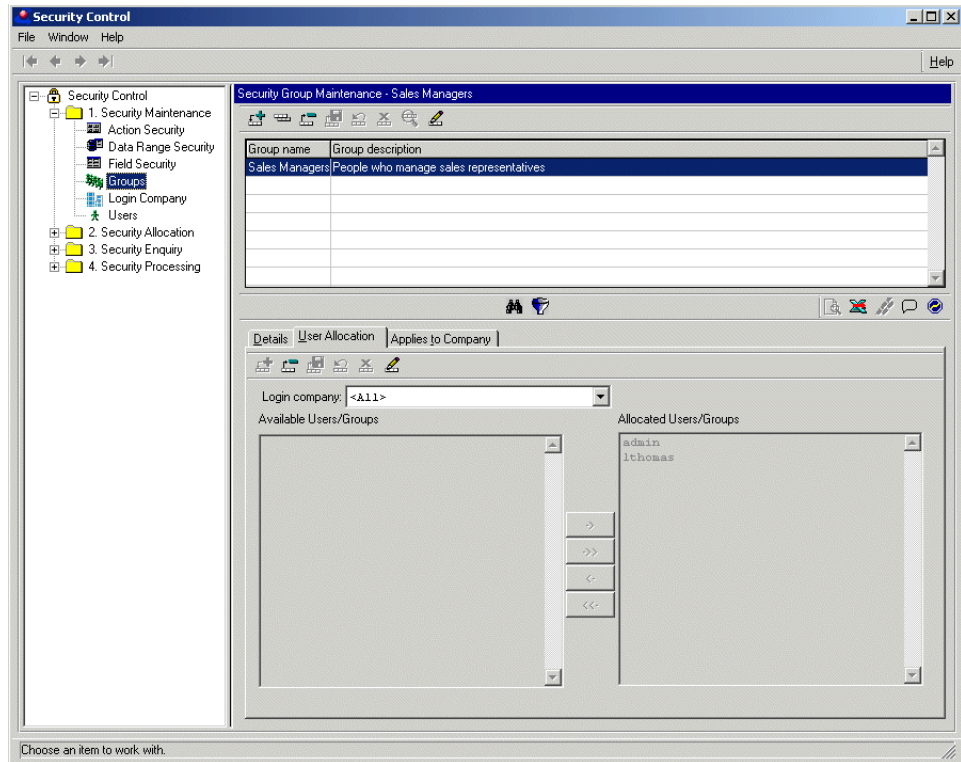
- 1 ♦ In the **Security Control** window, expand the **Security Maintenance** node, then choose the **Users** node.
- 2 ♦ Select a user in the browse of the **User Maintenance** window in the right frame, then choose the **Allocated to Groups** tab, as shown:



- 3 ♦ Use the buttons in the toolbar of the **Allocated to Groups** tab folder to add or modify the groups linked to the current user.
- 4 ♦ Choose the **Save** button to save any changes you make.

To identify all of the users in a specific security group:

- 1 ♦ In the **Security Control** window, expand the **Security Maintenance** node, then choose the **Groups** node.
- 2 ♦ Select a group in the browse of the **User Maintenance** window, then choose the **User Allocation** tab, as shown:



- 3 ♦ Use the buttons in the toolbar of the **User Allocation** tab folder to add or modify the users or groups linked to the current group.
- 4 ♦ Choose the **Save** button to save any changes you make.

Supporting Built-in Application Features

Progress Dynamics provides a number of management functions to support application features that are built into the framework. As with the framework in general, these functions reduce the coding effort required to provide the supported features in an application. In some cases, you do not need to provide any extra coding to make the features available, while in others the management functions described in this chapter allow you to define properties or behavior for the features that you can access from your code. In general, these are generic features that you enable for access in all of your application objects, rather than to implement a particular object or function.

The following sections describe:

- [Specifying gapless sequences](#)
- [Defining languages, countries, nationalities, and translations](#)
- [Adding generic database comments and auditing](#)
- [Specifying multi-media types](#)
- [Defining application categories](#)
- [Maintaining user-defined status information](#)
- [Exporting data through Print Preview](#)

4.1 Specifying gapless sequences

The Progress Dynamics General Manager provides a generic mechanism for generating reference/sequence numbers for application records without holes in the sequence numbers. Progress sequences cannot be used for this purpose since holes can occur with Progress sequence numbers.

NOTE: When a sequence number is required during record creation, it should be created at the end of the update as part of the transaction. Such transactions are potential bottlenecks, and locks on the affected table should be kept at an absolute minimum.

4.1.1 Sequence Control window

You use the Sequence Control window to define any sequences that you reference in your application. As described in the previous section, these sequences are automatically defined in and generated from one of the two Repository databases, depending on your Repository configuration.

To define sequences, choose **System→Sequence Control** from the Progress Dynamics Administration tool’s menu bar. This opens the **Sequence Control** window, as shown in [Figure 4-1](#).

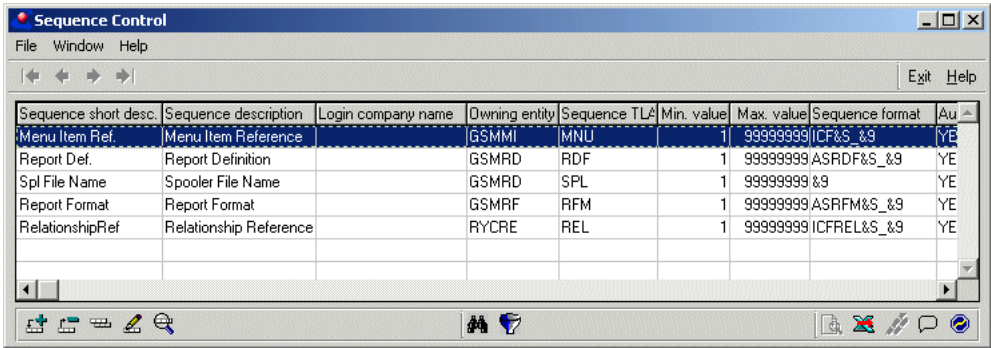


Figure 4-1: Sequence Control window

Like most control windows installed with Progress Dynamics, it contains a single browse that controls access to the records of a query, in this case, records that define gapless sequences and reference numbers for use in your application. You can use the toolbar at the bottom of the control window (a standard Progress Dynamics TableIO toolbar) to add new records or to delete, copy, edit, or view a currently selected record. Any add, edit, or view function opens the Sequence Maintenance window, with fields initialized and displayed according to the selected row and function (default values for an add record).

4.1.2 Sequence Maintenance window

The **Sequence Maintenance** folder window, shown in [Figure 4–2](#), allows for editing of all the fields for a sequence.

Figure 4–2: Sequence Maintenance window

The **Sequence TLA** fields refers to a unique three-letter-acronym which, in conjunction with the **Entity Mnemonic** field, forms the composite key. Thus, each mnemonic can have multiple sequences.

The **Sequence Format** field contains a user-defined format mask for the sequence. The format mask must include a numeric format that allows for the maximum value and that conforms to the underlying database format rules (for example: 99999999 = eight-digit number with 0 prefix). The format can include hard-coded characters (for example **PO** for purchase order).

The format can also include insertion codes to dynamically insert data at run time, as follows:

- %Y = Insert the last digit of the current year.
- %YY = Insert the current two-digit year.
- %YYYY = Insert the current four-digit year.
- %MM = Insert the current two-digit month.
- %MMM = Insert the current three-character month.
- %DD = Insert the current day of the month.

4.2 Defining languages, countries, nationalities, and translations

Progress Dynamics provides both a development and a user run-time translation capability for application localization. That is, you can define codes for languages, countries, and nationalities. You can also provide translations for all of the UI widget labels, ToolTips, and even constant data for an application as you develop it. After you have deployed an application, users can modify or create the translations for the UI of any Progress Dynamics window they happen to be using. Menu translations are easy with a feature exclusive to the Toolbar and Menu Designer. The following sections describe how to use all of these localization options.

4.2.1 Language Control window

To define a language, choose **Application→Language Control** from the Progress Dynamics Administration tool's menu bar. The **Language Control** window, as shown in [Figure 4–3](#), opens displaying a list of the currently defined languages.

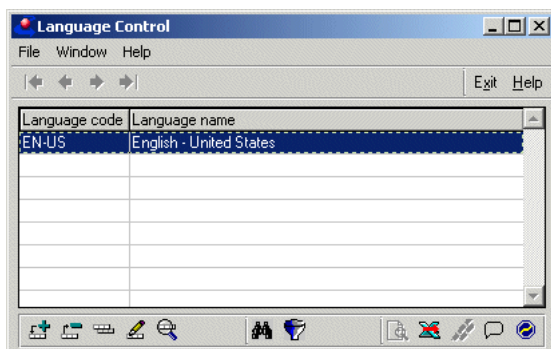


Figure 4–3: Language Control window

To add another language, choose **File→Add record**. The **Language Maintenance** window appears.

4.2.2 Language Maintenance window

The **Language Maintenance** window has two fields that specify a language.

As shown in [Figure 4-4](#), you are adding German to the supported languages, using the English name for the language as the unique **Language Code**, and you are entering the German name (Deutsch) as the **Language Name** for your German users.

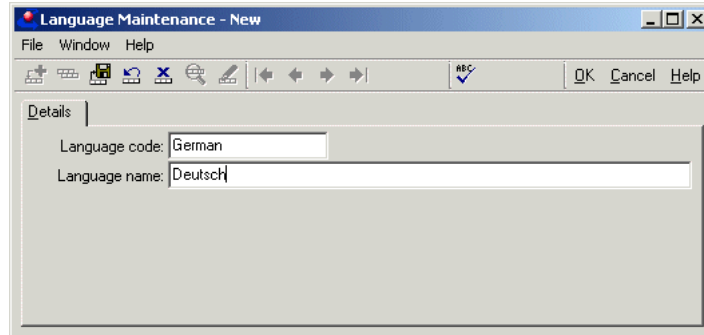


Figure 4-4: Language Maintenance — New window

After you save, the new record appears in the **Language Control** window.

4.2.3 Country Control window

To define a country, choose **Application→Country Control** from the Progress Dynamics Administration tool's menu bar. The **Country Control** window, as shown in [Figure 4-5](#), opens displaying a list of currently defined countries.

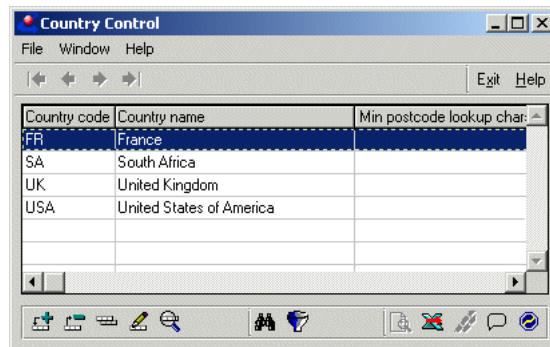


Figure 4-5: Country Control window

To add another country, choose **File→Add record**. The **Country Maintenance** window appears.

4.2.4 Country Maintenance window

The **Country Maintenance** window allows editing of all the fields to specify a country. As shown in [Figure 4–6](#), you are adding German words for the address labels.

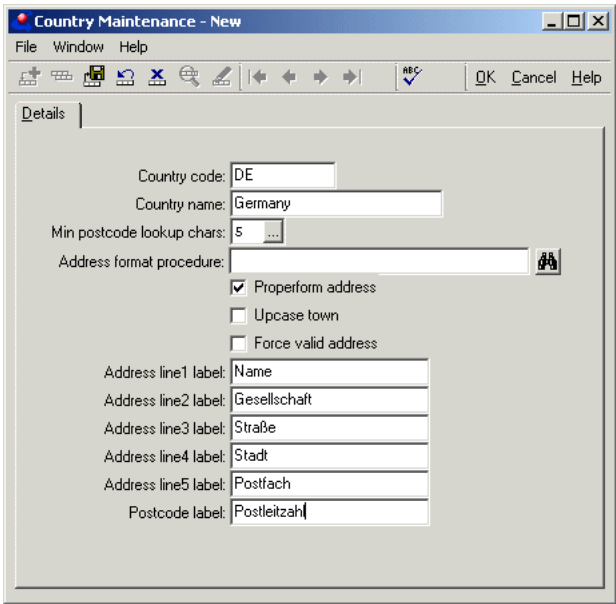


Figure 4–6: Country Maintenance window

After you save, the new record appears in the **Country Control** window.

4.2.5 Nationality Control window

To define a nationality, choose **Application**→**Nationality Control** from the Progress Dynamics Administration tool's menu bar. This opens the **Nationality Control** window, as shown in [Figure 4-7](#).

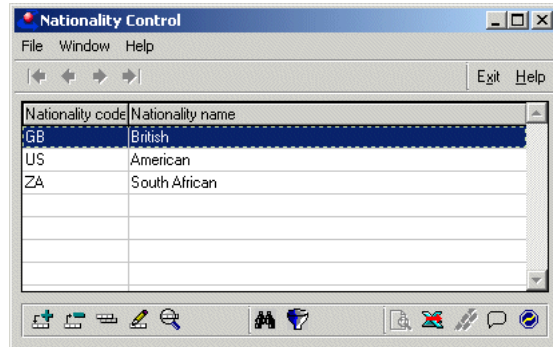


Figure 4-7: Nationality Control window

To add another nationality, choose **File**→**Add record**. The **Nationality Maintenance** window appears.

4.2.6 Nationality Maintenance window

The **Nationality Maintenance** window as shown in [Figure 4-8](#), has two fields that specify a nationality. In [Figure 4-8](#), you are adding German as a supported nationality.

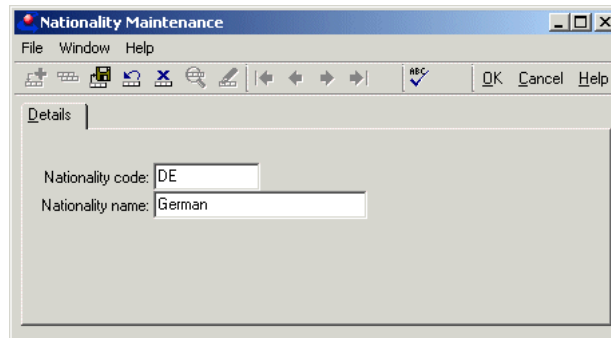


Figure 4-8: Nationality Maintenance window

After you save, the new record appears in the **Nationality Control** window.

4.2.7 Setting translation preferences

If your application uses containers with multiple pages and you want to translate the items in those pages, you should check the **Initialize all pages when translating** option in the **Preferences** dialog box, as shown in [Figure 4-9](#):

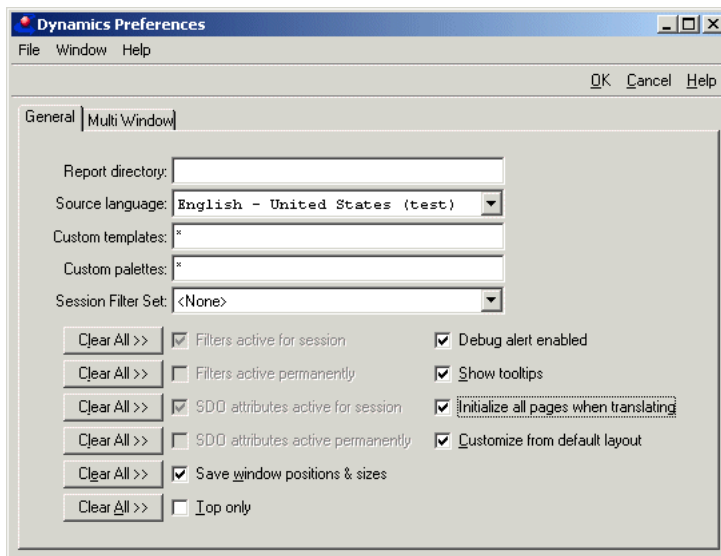


Figure 4-9: Dynamics Preferences dialog box

To access the Preferences dialog box, from the Dynamics Administration window, choose **File**→**Preferences**. You should set this option before you open the **Translation Control** window to ensure that all translatable items are shown in the list presented in the **Translation Control** window.

4.2.8 Translation Control window

To create language translations for text objects during development, choose **Application**→**Translation Control** from the Progress Dynamics Administration tool menu bar. This opens the **Translation Control** window, as shown in [Figure 4-10](#).

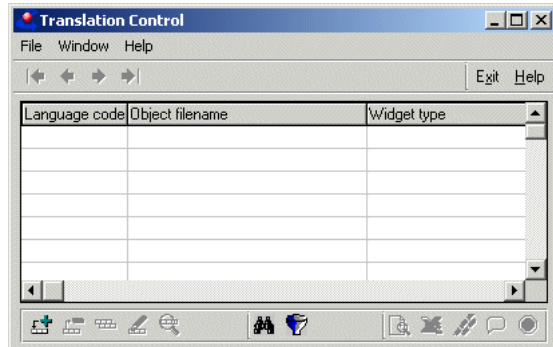


Figure 4-10: Translation Control window

If you do not see all the fields for a container, make sure that you set the **Initialize all pages when translating** option in the **Preferences** dialog box. See the [“Setting translation preferences”](#) section for more information.

4.2.9 Translation Maintenance window

The **Translation Maintenance** window, as shown in [Figure 4-11](#), allows for editing of all the fields to specify a translation for a specific widget label. This example shows a translation for the Progress Dynamics **Administration** window title.

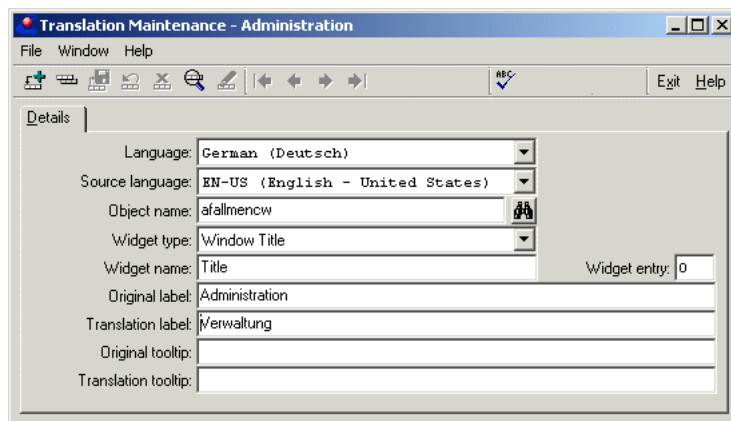


Figure 4-11: Translation Maintenance window

The valid widget types are Browse Column, Button, Combo Box, DataField, Editor, Fill In, Global translation, Radio Set, Slider, Tab Folder Page, Text, Toggle Box, and Window Title. (You cannot create a translation label for Text widgets in dynamic viewers.) Also notice that the **Translation Maintenance** window allows for translations of ToolTips as well as widget labels.

There is a size limitation on widget labels. They cannot be longer than 60 characters.

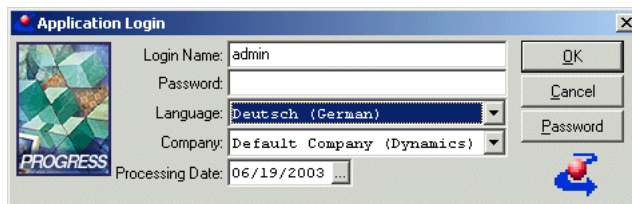
See the online help for more information about the fields in the **Translation Maintenance** window.

After you save, the new translation appears in the **Translation Control** window.

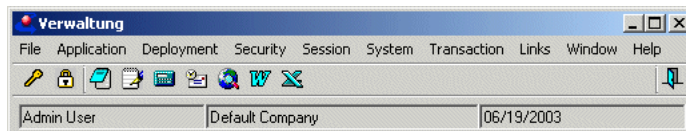
4.2.10 Testing the new translation

Follow these steps to test the new translation in the previous example:

- 1 ♦ Choose **File→Re-Login** from the Progress Dynamics Administration window. The **Application Login** dialog box appears:



- 2 ♦ Select the language that you specified for the translation, and log in as before.
- 3 ♦ Open a new instance of the Progress Dynamics Administration window using either the **Tools** menu on the AppBuilder window or the **Links** menu on the existing Progress Dynamics Administration window. The new window opens, as shown:



4.2.11 User run-time translations with the Translate window

If the Progress Dynamics installation administrator has enabled translation using the Security Control window (see [Chapter 1, “Overview of Progress Dynamics Administration”](#)), any users running an application can enter their own translations for text objects if they do not like the ones that they see. Every Progress Dynamics window container has a Translate item on the File menu that opens the **Translate Window** window, as shown in [Figure 4–12](#).

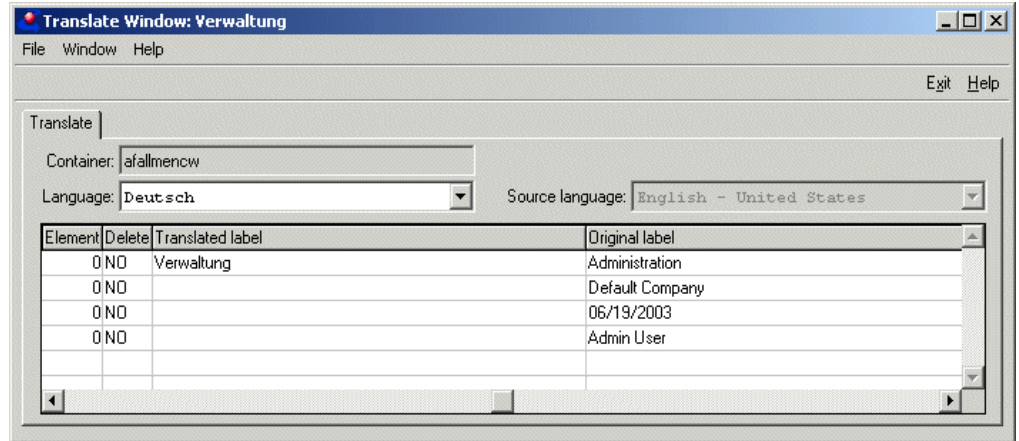


Figure 4–12: Translate window for user run-time translations

This tool is a Progress Dynamics folder window with a static updateable browse. This browse shows all the translateable widgets that are in the window from which the user opened the Translate Window tool. Most of the fields are read-only. Those fields that the user can change take effect only if translation is enabled for the installation. Otherwise, any attempts to change them are ignored.

[Figure 4–13](#) shows the user changing the existing valid translation to a nonsensical one.

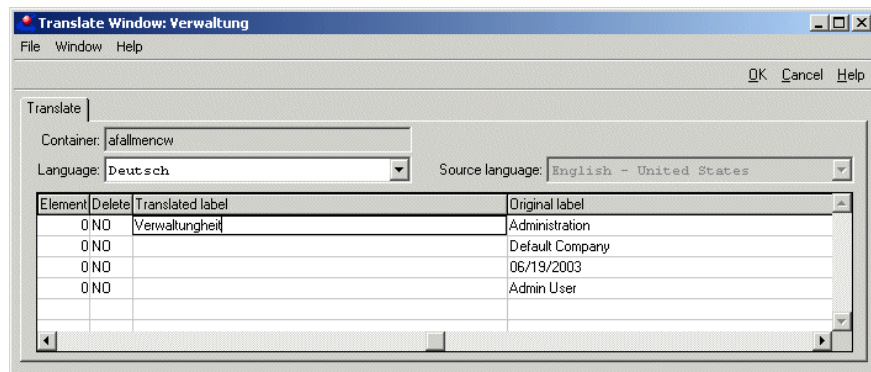


Figure 4–13: Translate window with new translation

The next instantiation of the Progress Dynamics Administration window opens with the new translation, as in [Figure 4–14](#).

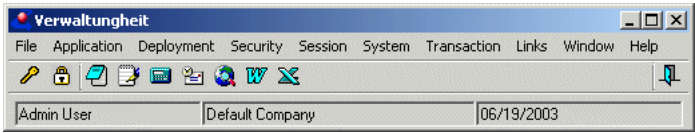


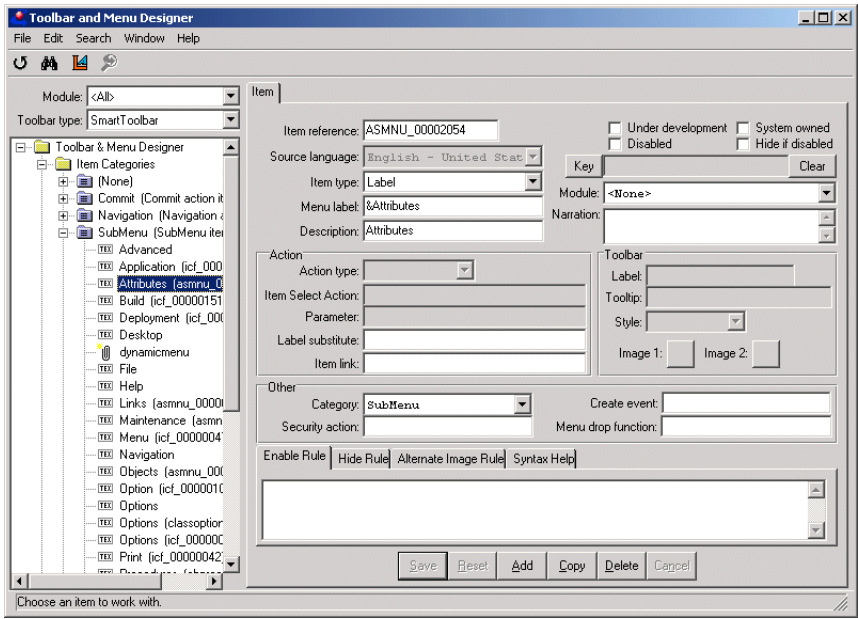
Figure 4–14: Administration window with run-time translation

4.2.12 Menu translations


To translate menu items, you use the Toolbar and Menu Designer tool.

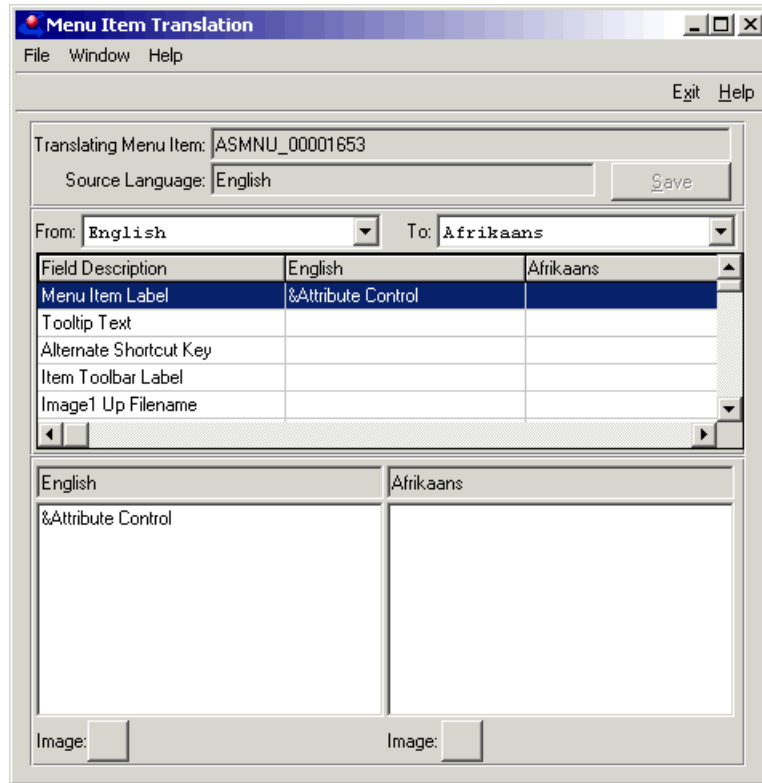
Follow these steps to translate a menu item:

- 1 ♦ From the Dynamics AppBuilder main window or from the Development window, choose **Build→Toolbar and Menu Designer**, as shown:



- 2 ♦ Choose the **Item Category** that you want to translate.

- 3 ♦ In the Toolbar and Menu designer, choose the **Translate Menu Item**  button. The **Menu Item Translation** window appears:



- 4 ♦ Select the **From** language, which is called the *source language*. The source language lets you specify the default language of a menu item. Due to the global nature of Dynamics, the language in which an application is **developed** is the source language.
- 5 ♦ Select a row in the browser for the text element you want to translate.
- 6 ♦ For a short translation, type the text directly in the updateable browse. Alternatively, for a longer translation, enter the text in the text editors below the browse.
- 7 ♦ Choose **Save**.

4.2.13 Entity translations

Translations in the previous sections apply primarily to field labels associated with specified visual objects at run time (*widget translations*), such as browser columns and widget objects on viewers. You can also specify translations (*entity translations*) at the entity object (DataField) level of an entity, no matter what visual object displays them.

Dynamics makes entity translations available to dynamic browser (DynBrow) and dynamic viewer (DynView) objects that are linked to dynamic Progress SmartDataObjects™ (DynSDOs), and also to DynView objects linked to a static SDO. Entity translations also apply to the field labels displayed in the standard Progress Dynamics Find/Filter browser. If DataFields with entity translations are used by visual objects that have widget translations, the widget translations override any entity translations when the fields are displayed using these visual objects.

NOTE: Automatic resizing and positioning of widgets to accommodate translated labels does not occur when using entity translations. When using entity translations, you must provide enough layout space to display the translated labels to avoid overlapping fields.

You can specify entity translations using these tools:

- Translation Maintenance window
- Migrate Widget Translation tool

Using Translation Maintenance

The Translation Maintenance window allows you to manually specify entity translations for individual fields similar to widget translations. (See the [“Translation Maintenance window”](#) section.) Specify the field values in this window as you do for widget translations, except for the following fields:

- **Object name** — The name of the entity where the translated field resides.

NOTE: Be careful not to select a field name in the lookup. Select only the name of an entity. You can eliminate field names from the selection by using the lookup filter with the Object type code From and To values set to **Entity**, along with other filter settings, such as appropriate Object filename values.

- **Widget type** — DataField.

NOTE: The value, Global translation, specifies a single widget translation that applies to all visual objects that display values for the specified field.

- **Widget entry** — 1 specifies to translate the field label; 2 specifies to translate the field column label.
- **Original tooltip** — Not used (blank).
- **Translation tooltip** — Not used (blank).

Using Migrate Widget Translation

The Migrate Widget Translation tool allows you to automatically convert existing widget translations to entity translations for specified entities in specified databases. You also have the option to remove any existing widget translations and to skip entity translation for any fields that have multiple widget translations.

The Migrate Widget Translation tool is separate from the AppBuilder tools. To use this tool, run the `afmigtrnww` container with the Dynamic Launcher (**Compile**→**Dynamic Launcher** from the AppBuilder window).

4.3 Adding generic database comments and auditing

Progress Dynamics provides built-in support for generic commenting and auditing on database records and transactions. Both the Generic Comments and Auditing features are available to any user, by default, on the Progress Dynamics browser toolbar (and control window File menu). Using the Comments feature, users can add their own comments to any record that they select in a browse, and specify additional options for how their comments are handled by Progress Dynamics. Using the Auditing feature, users can display any audit records that have been generated from transactions on the selected browse record. As with any Progress Dynamics application function, you can create security restrictions to prevent certain users from accessing either feature.

The following sections provide more information on these features of the Progress Dynamics UI, including how to use them and make them available to application users.

4.3.1 Generic comments

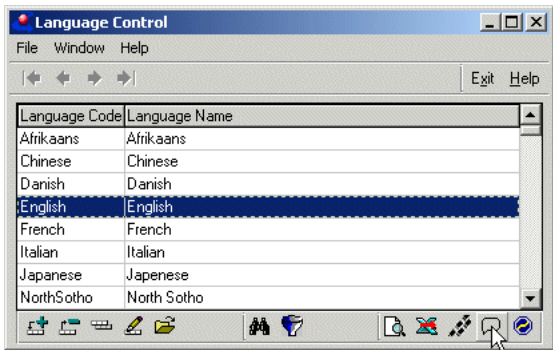
Throughout an application, the ability to attach memos or comments to various data records in the application might be required. As this functionality can be required on one or many of the tables in an application, a generic method is available to accomplish the storing of additional information against a record, such as comments or memos. Using the Generic Comments feature, this functionality is available on all tables in an application without any application database or code changes.

The Comments functionality is added to the standard ADM2 and is therefore generically available throughout any application built using the Progress Dynamics framework. For more information, see the [Progress Dynamics ADM2 API Reference](#). The use of Generic Comments on a table requires that one or more fields in a unique index uniquely identify a record.

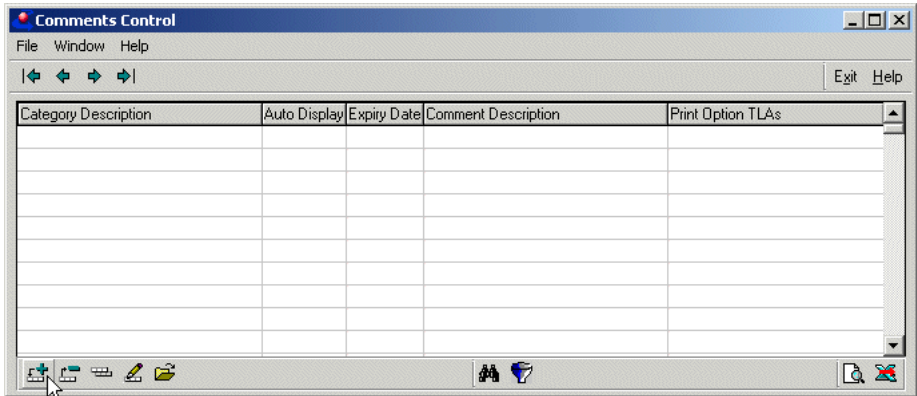
Adding comments to a record

Follow these steps to add comments to a record:

- 1 ♦ Open an control window with a browse containing the record you want to comment, For example, from the **Application** menu, open the **Language Control** window:



- 2 ♦ Choose the **Comments** button in the browser toolbar or choose **File→Comments** from the menu bar. The **Comments Control** window appears:



- 3 ♦ In the **Comments Control** window, choose the **Add record** button on the toolbar. The fields in the **Details** folder are enabled, as shown:

The screenshot shows the 'Comments Folder Window' with the 'Details' tab selected. The window has a menu bar (File, Window, Help) and a toolbar with icons for adding, deleting, and editing records. The 'Add record' button is highlighted. The 'Details' tab contains the following fields:

- Category:** A dropdown menu showing 'Notes (NOT ALL NOT)'.
- Comment Description:** A text box containing 'Default Language'.
- Comment Text:** A large text area containing 'This is the default language in Dynamics'.
- Expiry Date:** A date picker showing '/ /'.
- Auto Display:** A checked checkbox.
- Print Option TLAs:** A text box.
- language_code:** A text box containing 'English'.

- 4 ♦ Fill in the fields in the **Details** tab. Use the online help for descriptions of each of these fields.
- 5 ♦ Choose **Save**. The new comment appears in the **Comments Control** browse, as shown:

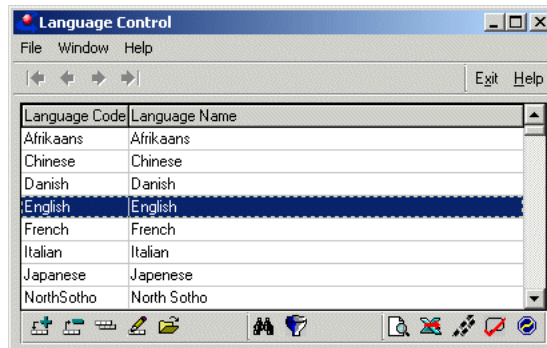
The screenshot shows the 'Comments Control' window with a table of comments. The table has the following columns: Category, Description, Auto Display, Expiry Date, Comment Description, and Print Option TLAs. The first row is highlighted in blue and contains the following data:

Category	Description	Auto Display	Expiry Date	Comment Description	Print Option TLAs
Notes		YES		Default Language	

Accessing a commented record

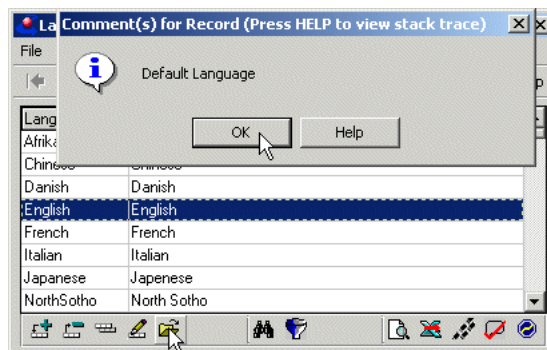
Follow these steps to trigger the Auto Display of a comment on a record:

- 1 ♦ Select the record in the browse, as shown:

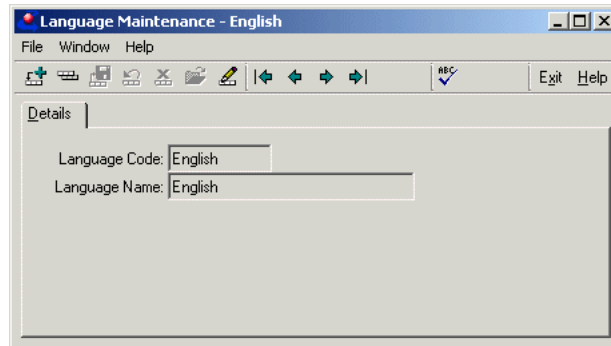


NOTE: The **Comments** button on the browse toolbar displays with a red check mark when you select a record with a comment.

- 2 ♦ If you open the record for viewing a **Comment** information box appears:



- 3 ♦ After dismissing the comment information box, the viewer appears displaying the record:



If you modify the record in the viewer, the same comment box appears when you save the changes.

4.3.2 Generic auditing

Throughout an application, various data entries can be added, modified, or deleted. For security and other auditing-related reasons, there must be a way to track these changes by keeping a history of them. As auditing can be required on one or many tables in an application, Progress Dynamics provides a generic method to acquire and store the information.

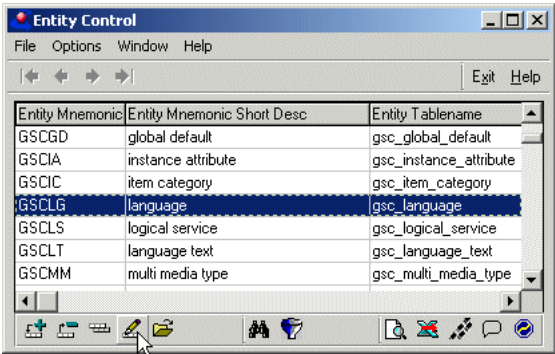
Progress Dynamics includes a generic auditing procedure as a standard component of each database trigger. If auditing is enabled for the relevant table, Progress Dynamics automatically launches the audit procedure as part of trigger execution for a transaction on that table. Any application user can then review the audit history on a selected record by accessing the Audit function available on the browse toolbar (or File menu) of an control window.

NOTE: You can also run the auditing procedure manually from within your application business logic. For more information, see the [Progress Dynamics Developer's Guide](#).

Enabling auditing on a table

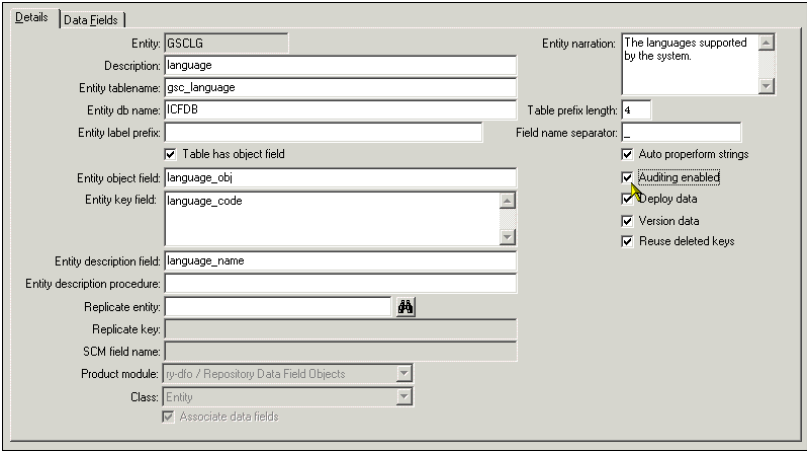
To enable auditing on a table, you must activate a toggle box on the Progress Dynamics entity definition for the table by following these steps:

- 1 ♦ Choose **System→Entity Control** from the Progress Dynamics Administration menu bar. The **Entity Control** window opens:



- 2 ♦ In the browse, select the entity record for the table where you want auditing enabled and open it for update by choosing the Modify record function on the toolbar (or File menu). The **Entity Maintenance** folder window opens.
- 3 ♦ Activate the **Auditing Enabled** toggle box and save the record. From this moment, the audit procedure will execute for any transaction on the gsc_language table.

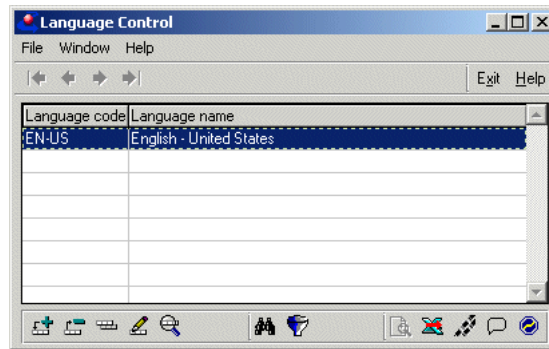
The example enables the gsc_language table for auditing:



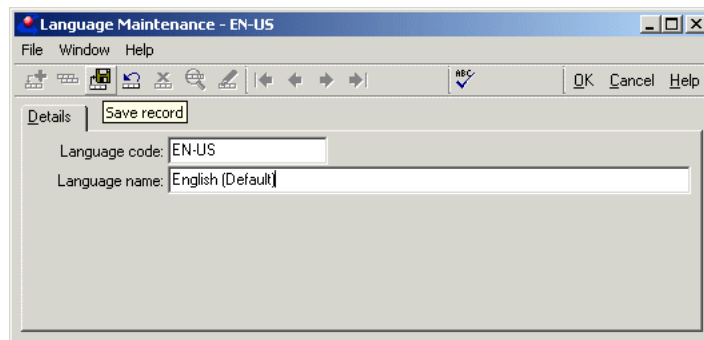
Triggering an audit entry

Follow these steps to trigger an audit entry on the gsc_language table just enabled for auditing:

- 1 ♦ Choose **Application**→**Language Control** from the Progress Dynamics Administration menu bar.
- 2 ♦ In the **Language Control** window that opens, select and open the **English** record for update, as shown:



- 3 ♦ Choose the **Modify** record button, then make a change to the **Language Name** field, for example append **(Default)** to the end, as shown:

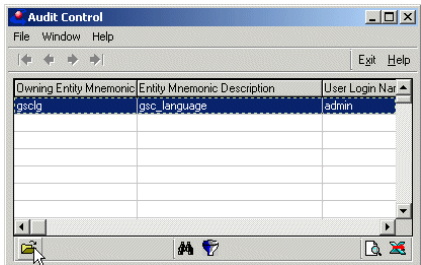


- 4 ♦ Save the record.
- 5 ♦ Close the **Language Maintenance** and **Language Control** windows.

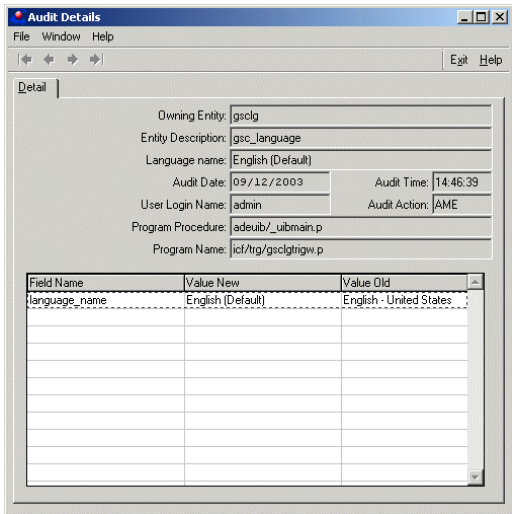
Viewing an audit

Follow these steps to view the audit on the record:

- 1 ♦ Choose **Application→Language Control** from the Progress Dynamics Administration menu bar. The **Language Control** window opens.
- 2 ♦ Choose **File→Audit**. The **Audit Control** window opens, as shown:



- 3 ♦ Open the audit record for viewing. The **Audit Details** window opens, as shown:



This is a view-only function, because audit records cannot generally be changed in a typical audit situation (whether for legal or other reasons). The **Audit Action** field is a code that indicates the trigger for the audit:

- CRE — CREATE trigger.
- AME — WRITE trigger (AMEndment).
- DEL — DELETE trigger.

4.4 Specifying multi-media types

Progress Dynamics provides a mechanism to define multi-media files that can be generically linked to any entity. These multi-media files can be of a specific multi-media type, such as bitmap, wave, audio/visual, and so on.

The Multi Media Type Maintenance suite of programs to define these types consists of an control window and a folder window in typical Progress Dynamics style.

4.4.1 Multi Media Type Control window

To specify multi-media types, choose **Application→Multi Media Type Control** from the Progress Dynamics Administration tool menu bar. This opens the **Multi Media Type Control**, window as shown in [Figure 4–15](#). This example adds a multi-media type.

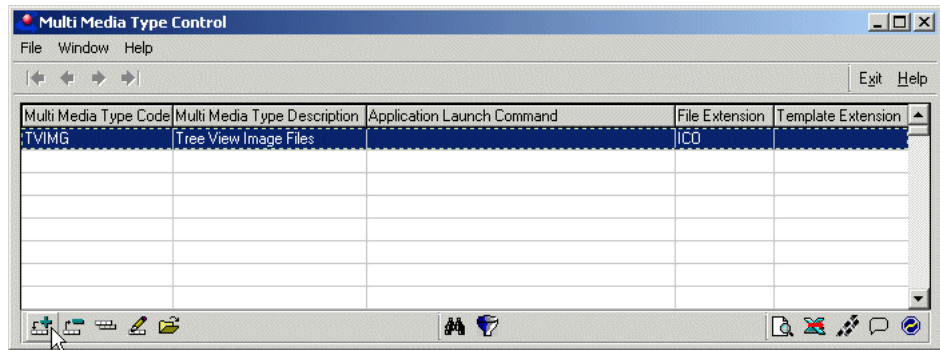


Figure 4–15: Multi Media Type Control window

Like most control window installed with Progress Dynamics, it contains a single browse that controls access to the records of a query, in this case, records that specify multi-media file types and the programs that create them. You can use the toolbar at the bottom of the control window (a standard Progress Dynamics TableIO toolbar) to add new records or to delete, copy, edit, or view a currently selected record. Any add, edit, or view function opens the **Multi Media Maintenance** window, with fields initialized and displayed according to the selected row and function (default values for an add record).

4.4.2 Multi Media Type Maintenance window

The **Multi Media Type Maintenance** folder window, as shown in [Figure 4–16](#) allows for editing of all the fields for a multi-media type, in this case, for MS Word documents.

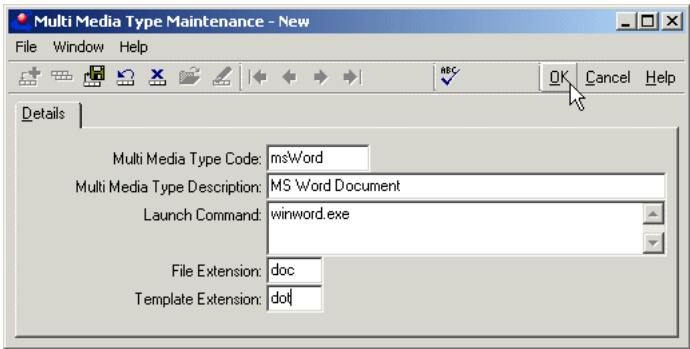


Figure 4–16: Multi Media Type Maintenance window

Once the type is saved in the **Multi Media Type Maintenance** window, it appears in the control window as shown in [Figure 4–17](#).

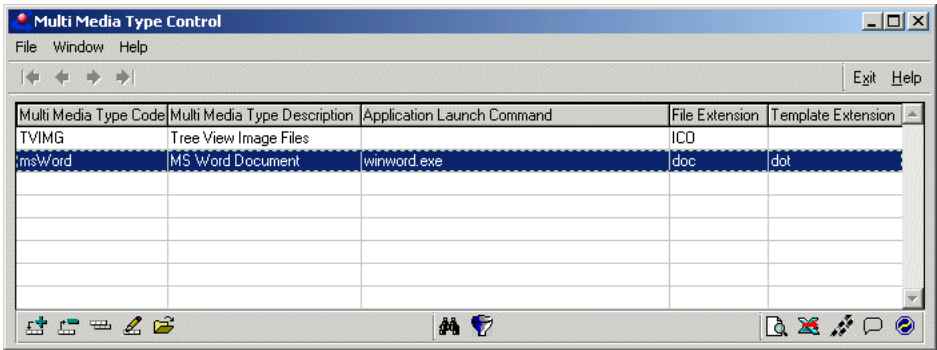


Figure 4–17: Multi Media Typed Control window with new media type

4.5 Defining application categories

Progress Dynamics includes a generic mechanism for splitting any type of data into categories using a multi-purpose category table. It is often a common requirement to categorize data to multiple levels for filtering and reporting. This category table facilitates this, along with a mechanism to define valid value lists for use in an application.

4.5.1 Category definition structure

To define a category of data for a table, the related entity mnemonic of the category is assigned the entity mnemonic code for the table, which is the unique key to identify the table. For example, to define a category of comments from the `gsm_comment` table, the related entity mnemonic would be set to `GSMCM`, which is the unique code for the comment table. To define a category for valid value lists, the related entity mnemonic is assigned to `SYSYS` to indicate that it is not data related to a specific table. The `SYSYS` code is a system code that points to no particular entity.

Understanding category grouping levels

The category table supports three levels of grouping for data in a related table:

- Category type
- Group
- Subgroup

Subgroups can be logically sequenced within a group using an integer sequence number. The number of grouping levels used for a category and their meaning is dependant on the data being categorized. Categories that are required in order for an application to function can be defined as system owned.

Using an owning entity for the categorized table

There is another entity mnemonic in the category table called an owning entity mnemonic. This is only relevant if the table to which the category applies is generically joined to a second table using an owning object field. In this case, the owning entity mnemonic defines the table referenced by the owning object field.

An example of this use can be described with status codes. Progress Dynamics supports the definition of user-defined status codes to allow users to set up customized process flows. The category table is used to define the categories of statuses that are used in the application and defined by the `gsm_status` table. For example, categories can be set up for the available statuses for an account, such as open or closed.

When allocating a particular status to a record, the generic table gsm_status_history is used as the related table, which records the current status for a particular record in another table. This other table is referenced by a generic join on an owning_obj field. In order to identify what table this object field references related to this particular status, the owning entity mnemonic of the category (status) is defined. In this example, the owning entity mnemonic is the entity mnemonic for the account table, as these open and closed statuses pertain to records in the account table.

Categories used in and supported by Progress Dynamics

Progress Dynamics makes use of categories to facilitate the Generic Comments feature, multi-media file types (TreeView images), Status support, user profiles, custom procedures, and language text. For more information on Status support, see the [“Maintaining user-defined status information”](#) section.

The Category Maintenance suite of programs consists of an control window and a maintenance window in typical Progress Dynamics style.

4.5.2 Category Control window

To defining application categories, choose **Application→Category Control** from the Progress Dynamics Administration tool menu bar. This opens the **Category Control** window, as shown in [Figure 4–18](#).

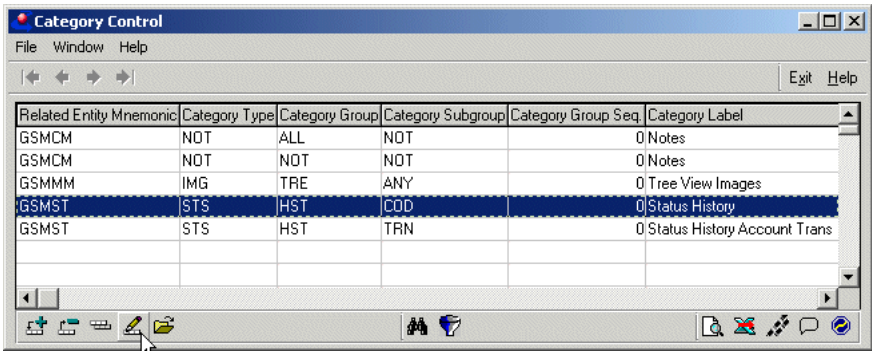


Figure 4–18: Category Control window

Like most control window installed with Progress Dynamics, it contains a single browse that controls access to the records of a query, in this case, records that define application categories that you can use to reference and organize related data in an application. You can use the toolbar at the bottom of the control window (a standard Progress Dynamics TableIO toolbar) to add new records, or to delete, copy, edit, or view a currently selected record. Any add, edit, or view function opens the **Category Maintenance** window, with fields initialized and displayed according to the selected row and function (default values for an add record).

4.5.3 Category Maintenance window

The **Category Maintenance** folder window, as shown in [Figure 4–19](#), allows for editing of all the fields for a category.

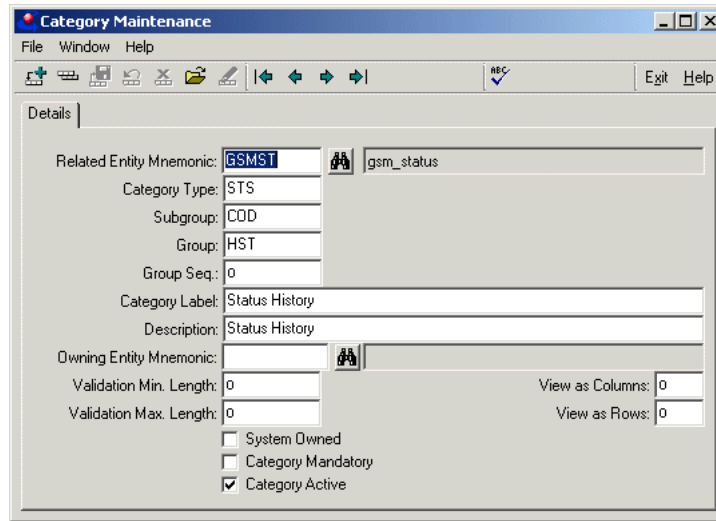


Figure 4–19: Category Maintenance window

4.6 Maintaining user-defined status information

Progress Dynamics supports a mechanism that allows you to organize the flow of data or processes in an application according to a system of status codes that you define. The General Manager provides an API for referencing this status information as part of an internal status tracking system that you can design and implement for your applications.

4.6.1 How Progress Dynamics sets up the status mechanism

Status information is organized into categories that are implemented using the Progress Dynamics Category Maintenance suite of programs. The actual valid status codes are set up in the category table, in part, thus:

- **Related entity mnemonic** — GSMST for gsm_status.
- **Category type** — STS for Status.
- **Category group** — HST for History.
- **Category subgroup** — COD for Code.

In this case, the category subgroup is the actual status. The categories of status are always system-owned and mandatory. The category mandatory flag is used to indicate whether an object at this status can be modified. For more information and a description of a status tracking example, see the “[Defining application categories](#)” section.

4.6.2 Information on adding status tracking

For more information on designing and implementing the internals of status tracking, see the [Progress Dynamics Developer’s Guide](#) and the relevant General Manager API calls in the [Progress Dynamics Managers API Reference](#).

The following sections describe how to define and maintain status codes for any status tracking system that you might devise. Progress Dynamics also provides a **Status** button on the browser toolbar for application users to view status indications on a selected record.

The Status Maintenance suite of programs for defining and maintaining status codes and histories consists of an control window and a maintenance window in typical Progress Dynamics style.

4.6.3 Status Control window

To maintain entity status codes and histories, choose **Application→Status Control** from the Progress Dynamics Administration tool menu bar. This opens the **Status Control** window, as shown in [Figure 4–20](#).

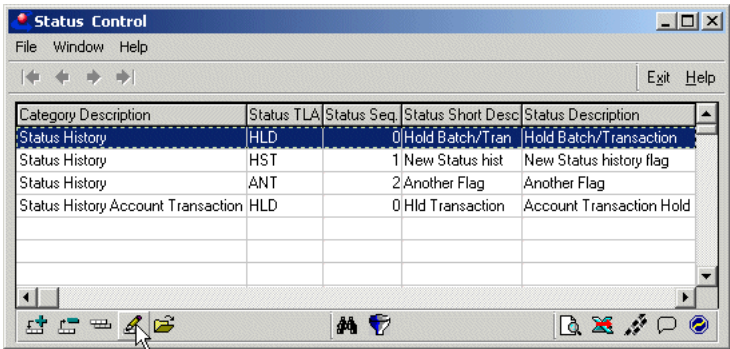


Figure 4–20: Status Control window

Like most control windows installed with Progress Dynamics, it contains a single browse that controls access to the records of a query, in this case, records that maintain status and status histories for selected entities in your application. You can use the toolbar at the bottom of the control window (a standard Progress Dynamics TableIO toolbar) to add new records or to delete, copy, edit, or view a currently selected record. Any add, edit, or view function opens the Status Maintenance window, with fields initialized and displayed according to the selected row and function (default values for an add record).

4.6.4 Status Maintenance window

The **Status Maintenance** window, as shown in [Figure 4-21](#), allows editing of all the fields for a status category.

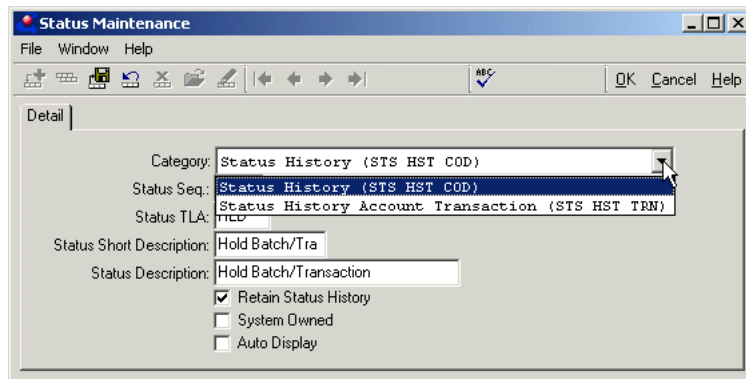



Figure 4-21: Status Maintenance window

4.7 Exporting data through Print Preview

The standard Browse Toolbar includes a Print Preview  button. This function allows you to export the displayed data as one of the following formats:

- XML (default)
- HTML
- Crystal Reports

4.7.1 Setting session properties

You need to set this option in advance through session properties. [Table 4–1](#) lists the session properties that control this function.

Table 4–1: Print Preview session properties

Session property	Description
print_preview_preference	The format for the exported data. Valid values are the following: <ul style="list-style-type: none">• XML (default)• HTML• Crystal
print_preview_stylesheet	An XML or HTML style sheet to apply to the exported data. Progress Dynamics supplies the following default style sheets: <ul style="list-style-type: none">• src\icf\af\rep\xmlreport.xml• src\icf\af\rep\htmlreport.css

The default settings are specified in the **Dynamics** session type, as shown in [Figure 4-22](#). The defaults are inherited by any session type that extends the Dynamics session type.

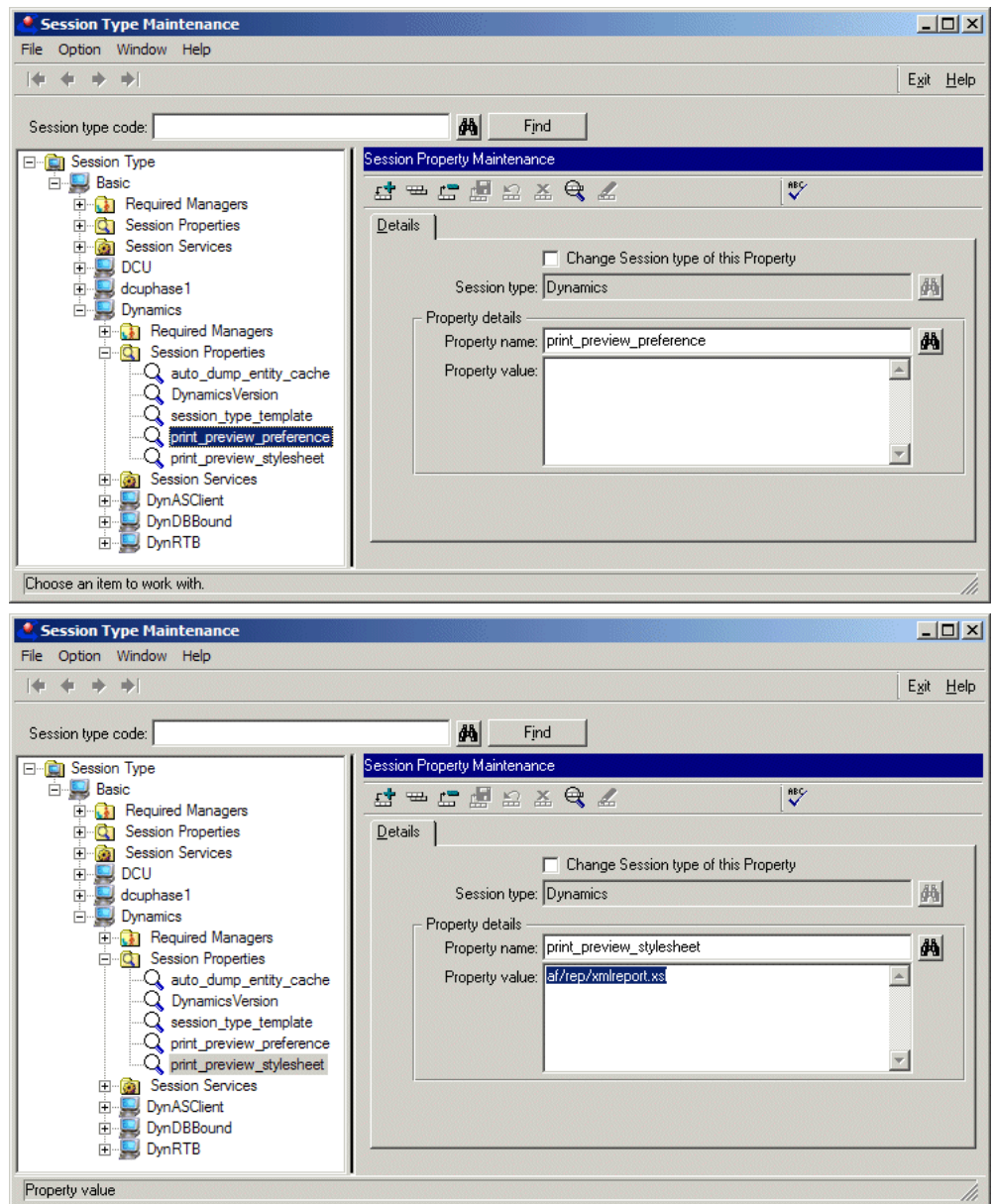


Figure 4-22: Default Print Preview settings

For more information on session type maintenance, see [Chapter 2, “Defining and Managing Sessions.”](#)

You can override the default setting for certain session types by adding the session property directly to the session type and assigning it a different value, as shown in [Figure 4–23](#).

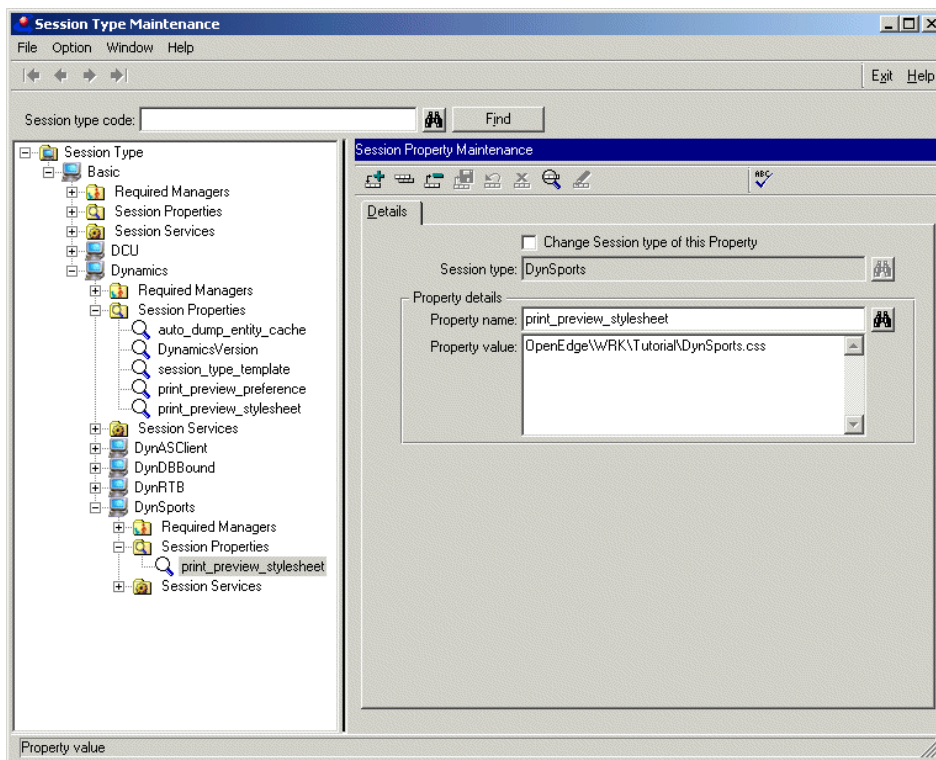



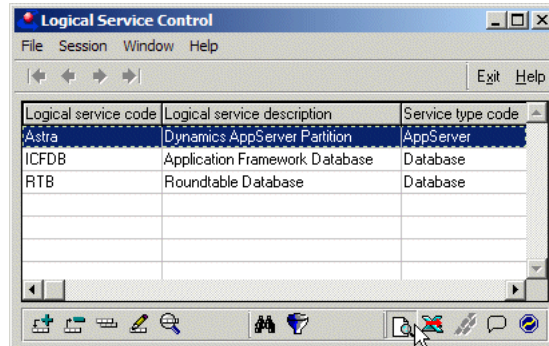
Figure 4–23: Overriding Print Preview settings

4.7.2 Using Print Preview

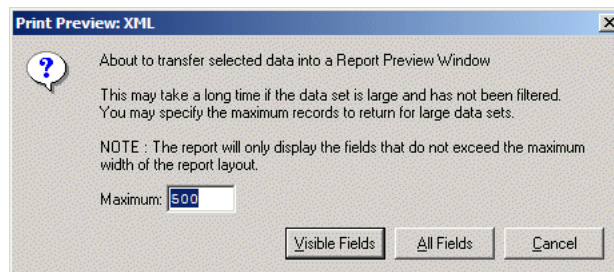
Suppose you are using the default session settings and you want to view the information for a logical services. The following procedure describes how you can do this.

To view logical service information using Print Preview:

- 1 ♦ Open the **Logical Service Control** window by choosing **Session→Logical Service Control** in the Administration tool.
- 2 ♦ Choose the **Print Preview**  button, as shown:

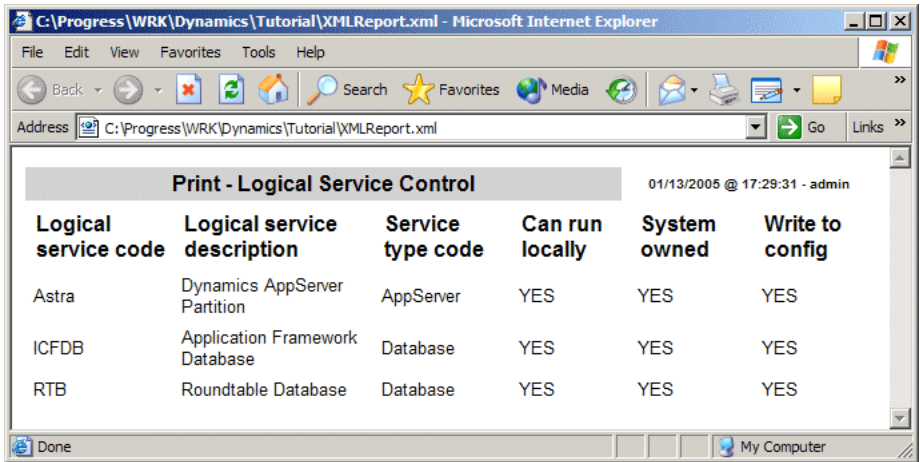


The **Print Preview** dialog box opens, as shown:



Note that the title bar indicates the type of output you are generating for Print Preview, in this example, XML. You can choose to view only the fields that are visible in the browser or all the fields that the browser contains.

- 3 ♦ If you choose to view all fields, choose the **All Fields** button, and the result (for XML) displays in your default Web browser, as shown:



Extending the Progress Dynamics Configuration Utility

As your Progress Dynamics application evolves, you have to deploy updates to your users. The first deployment is usually the easiest, since you do not have to protect any existing data. With any subsequent deployment, you are faced with the challenges of making the upgrade as quick and easy as possible while protecting your users' existing data.

Each deployment has its own unique challenges. The Progress Dynamics framework provides some tools and techniques to aid you in managing your particular deployment issues. This chapter discusses the following aspects of deployment:

- [Planning ahead](#)
- [Progress Dynamics Configuration Utility overview](#)
- [Deployment stages](#)
- [DCU XML files](#)
- [DCU API](#)
- [Creating a customized DCU deployment](#)
- [Running the DCU in batch mode](#)
- [Dumping and loading site-specific data](#)

5.1 Planning ahead

Because deployment comes at the end of the production cycle, it is often not considered until the application is nearing its final version. However, this increases the risk of critical upgrade, integration, and quality control issues. If you want to make your applications easier to deploy, you should think about deployment from the beginning of the design process. For example, Progress Dynamics works best in configurations that use the AppServer. You should familiarize yourself with the best practices for distributing applications in an AppServer environment and use them while designing your applications.

Each deployment has its unique elements. Often, you must make trade-offs between competing factors. Which trade-offs you find acceptable might depend on what type of application you are designing. For example, a financial application would not want to reduce security to gain a little more performance.

A good practice to adopt is to create regular builds of an application throughout the development cycle. You can then test deploying the application throughout the development cycle. This allows you to notice issues and take corrective steps while your application design is more flexible.

The design of your application and the requirements of your deployment strategy impact each other. You might find that the design of a particular module of your application increases the complexity of your deployment. By redesigning the module, you might ease the deployment problems. For example, when calculated fields support was added to the Progress Dynamics framework, it had a significant impact on deployment. After seeing the deployment issues the new feature caused, the design of the feature was changed to take that into consideration.

Application design and deployment strategy are interlinked. For complex applications, a developer who understands the deployment process should be a lead contributor in the application design process. That developer would monitor the deployment aspects of all the application modules.

As [Figure 5–1](#) shows, designing the application and designing the deployment strategy should be an iterative process.

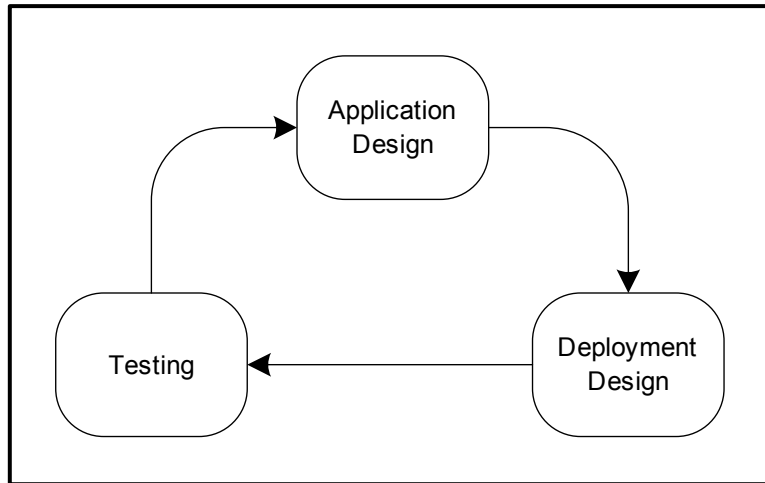


Figure 5–1: Application design cycle

Development databases often contain test objects and other extraneous data. However, you only want to deploy the data necessary to run your application. You can simplify your deployment process by eliminating this extraneous data before you begin building a deployment.

To accomplish this, you can set up central Repository and application databases that are only used to build deployment files. Developers only promote data to this central Repository when it has been tested and is ready to deploy. With this arrangement, you do not need to spend as much time figuring out which data needs to be deployed while building your deployment files.

5.2 Progress Dynamics Configuration Utility overview

You saw the Progress Dynamics Configuration Utility (DCU) when you first installed Progress Dynamics. When you upgraded to another version of Progress Dynamics, the DCU ran again to upgrade your Repository. The DCU is the framework's deployment tool. You can customize and extend it to deploy your applications as well.

The DCU presents upgrading Progress Dynamics applications as a wizard-style installation process. Its main task is to upgrade the Progress Dynamics Repository. However, the DCU is a Progress 4GL application. It has places where you can hook in other procedures to perform other upgrade tasks for your application.

5.2.1 Advantages of the DCU

The following sections briefly outline the advantages of the DCU.

Developed in the Progress 4GL

The DCU is written in the Progress 4GL and runs using the Progress 4GL interpreter. This supports easy access to OpenEdge databases, especially the Repository, and makes upgrade programs easy to invoke.

Script-driven, customizable, and extensible

To enable you to create your own installs, the DCU is a framework in its own right. You can add more libraries to it, add or remove wizard pages, and change the display order of wizard pages. At the end of the wizard process, the data that is prompted for is available so that it can be used to drive the upgrade process.

Requires no database connection

Due to the complexity associated with installing a database, the DCU does not require any database connection to begin its functions. The DCU does connect to the databases you are upgrading or creating as required.

Does not require most managers

Just as it does not require any database connection, the DCU runs independently of framework managers such as the Session Manager, Profile Manager, or General Manager. The managers require database connections at run-time to perform their tasks, so the DCU cannot rely on API calls to those managers.

Can be invoked from the command line

The DCU is invoked as the last part of the installation process. Since it runs outside the Progress Dynamics framework, you can invoke it from the command line.

Creates and updates databases

As part of setting up a new database or upgrading an existing one, the DCU applies schema changes to the database. The DCU applies these changes using the standard dictionary schema load program. When you create a new Progress database, the DCU loads the initial data to populate the database from Progress dump files created from the central Repository.

Runs update programs

Data in one version of an application must be altered to match schema changes in a newer version of the application. These alterations are often performed through update programs. The DCU is designed to support the running of update programs.

Writes list of updates to Repository

When you use the DCU to apply updates to databases, the DCU writes details of the updates to the Repository. There are two reasons for this:

- It provides the data required to continue the update process when you next connect to a Repository after running the DCU.
- It allows you to determine what updates have been applied to a Repository and any application databases.

Invokes load of Progress Dynamics datasets automatically

Once all the necessary changes have been applied to a Repository, the DCU automatically loads the application data into the Repository. This data is deployed using datasets (ADOs). The DCU automates the process for loading ADOs to avoid user error. Since the deployment dataset API requires the Progress Dynamics framework, this process runs when you next log into the environment after running the DCU.

Invokes post-login updates automatically

Beyond the dataset load, you might have other update programs that must run within the Progress Dynamics environment so that schema triggers can fire and calls can be made to the managers. The DCU can automatically initiate update programs both before and after loading ADOs.

Can be run unattended in batch mode

If an end-user site you want to update is maintained with limited on-site technical support, you might prefer to have the DCU perform its updates in a completely unattended fashion. You can do this by configuring the DCU to run in batch mode.

Can retain site-specific data during Repository updates

You might need to update a site that contains site-specific or non-standard application definitions. Depending on your application update, it might well unintentionally remove or modify these site-specific definitions in the process of updating other components of your application. Dynamics provides a set of utilities that you can run manually, or modify the DCU to run automatically, to save your site-specific data before the DCU performs its update, then to restore the site-specific data after the DCU update completes.

5.2.2 How the DCU differs from a Progress Dynamics session

In essence, the DCU is a minimal Progress Dynamics session running without most of the normal framework managers (such as Session, General, Profile, and Referential Integrity). The

startup procedure starts only the Configuration File Manager and the Connection Manager. The startup procedure then runs the Configuration File Manager's initializeSession procedure. This means that the DCU has access to the Configuration File Manager APIs, such as getManagerHandle, getSessionParam, and setSessionParam. The DCU can also use the Configuration File Manager's XML parsing facility to establish the initial environment.

Beyond the Configuration File and Connection Managers, the DCU also loads two special managers which contain the DCU's API, the Installation User Interface and ICFDB Install Managers. You can add other custom managers and APIs to what the DCU loads. You can then access them for your own upgrade processes.

Because the DCU uses the Configuration File Manager, parameters supported by the Configuration File Manager on the command line are also supported for the DCU. You can, for example, override the standard configuration file by specifying the ICFCONFIG parameter.

5.2.3 DCU interface

The DCU is driven from one window that contains several component sections. Every aspect of the user interface can be customized, but the main DCU window must remain the same size. The components displayed on each page are defined in the deployment's DCU driver file. [Figure 5-2](#) shows the sections of the main DCU window.

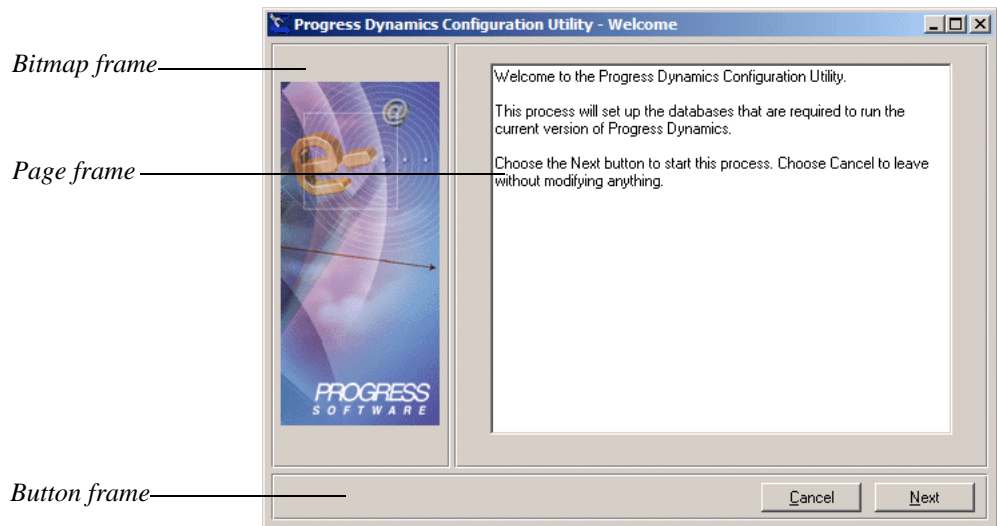


Figure 5-2: DCU interface

The DCU's component frames are as follows:

- **Bitmap Frame** — Contains a bitmap that is specified in the DCU driver file.
- **Page Frame** — Contains a Progress 4GL frame that has been derived from the DCU's template frame. The pages that are displayed in this frame are derived from DCU driver file.
- **Button Frame** — Contains the buttons that are defined within DCU driver file for each page of the DCU wizard.

Each page displayed by the DCU is a frame that has been defined using the template, `src/install/intmplframe.w`. There is little code inside the frame definition, because the behavior for all the controls is defined inside the DCU's API and XML files. When a page is displayed, the actions defined in the XML file are associated with the appropriate event on the controls. Then, any necessary processing is activated.

Users can even customize attributes of the individual controls that appear in the frame from within XML files by associating the attribute with the object's variable name. During the process of displaying the page, the DCU API determines the name of each control and associates attributes with the control as necessary.

While you can build new pages for the DCU wizard using the template, `src/install/intmplframe.w`, the existing pages can be adapted for most of your needs. If an existing page has an object that you do not need, you can hide it by adding a `<Hidden>` node to the corresponding XML control description.

The DCU has two trigger procedures that work with objects on the Wizard pages. The trigger code for a button's CHOOSE event runs the `btnChoose` API, as follows:

```
DO:
  RUN btnChoose IN THIS-PROCEDURE.
END.
```

For any other event for any object, there is a single API, eventProc. It takes two parameters, the name of the UI event and the name of the object the event. The following example shows the code for LEAVE event on a fill-in field:

```
DO:
  RUN eventProc IN THIS-PROCEDURE ("LEAVE":U,"{&SELF-NAME}":U) NO-ERROR.
  IF ERROR-STATUS:ERROR THEN
    RETURN NO-APPLY.
END.
```

To code any UI event on a page, you would add this code to the object’s trigger section, substituting the appropriate parameter values.

The existing pages are stored in the src\icf\install\obj directory. [Table 5–1](#) describes the existing frames.

Table 5–1: DCU standard pages and their objects (1 of 2)

Page	Description	Objects
Indbconn	Page for gathering database connection information.	edComment (Editor) lCreate (Toggle-box) lBuild (Toggle-box) fiPath1 (Fill-in) buPath1 (Button) fiPath2 (Fill-in) buPath2 (Button) fiConnect (Fill-in) edConnect (Editor)
Indbinfo	This page is no longer used by the DCU.	edComment (Editor) lICFDB (Toggle-box) lRVDB (Toggle-box) fiPath4 (Fill-in) buPath4 (Button)
Inpath	Page for gathering paths.	edComment (Editor) fiPath1 (Fill-in) buPath1 (Button) fiPath2 (Fill-in) buPath2 (Button) fiPath3 (Fill-in) buPath3 (Button) fiPath4 (Fill-in) buPath4 (Button)

Table 5–1: DCU standard pages and their objects

(2 of 2)

Page	Description	Objects
Inselect	Page with editor and a selection list.	edComment (Editor) seList (Selection-list)
Insite	Page for gathering site and sequence information.	edComment (Editor) fiNumber (Fill-in) fiSiteSeq1 (Fill-in) fiSiteSeq2 (Fill-in) fiSessionID (Fill-in)
Instatus	Page with two editors.	edComment (Editor) edStatus (Editor)
Inwelcome	Page with one editor.	EdEditor (Editor)

5.3 Deployment stages

The DCU provides a simple mechanism for applying various upgrade programs to a database in a scripted, effective way. Upgrade programs can provide virtually any needed function. Upgrade programs can be invoked at several stages during the process.

Phase 1 of the DCU process uses a specialized subset of the entire Progress Dynamics framework. In general, you should only run programs during Phase 1 that do schema updates and associated tasks on your databases. During this first phase, the DCU writes details of the entire upgrade process to the Repository's update table.

Whenever the framework starts a new session with a direct database connection for a user who has permission to modify system data, it checks whether there are any outstanding updates. If there are, the second phase of the DCU is invoked when the session starts, and the remaining upgrades are applied. Phase 2 upgrade programs have all the standard Progress Dynamics managers available, however the environment caches might not have been built yet. ADOs should usually be loaded before the caches are built.

NOTE: Phase 2 processing does not occur unless both conditions are met. The user must have permission to modify system data, and the new session must have a direct database connection. This means that you cannot initiate Phase 2 processing through an AppServer or WebSpeed connection for example.

The state of the various caches might affect when you need to run certain upgrade programs. You might also encounter stale data in caches. If an earlier upgrade program changed the data in the Repository, any cache that uses that data is stale. You might want to force the refresh of any needed cached data as the first step in an upgrade program.

5.3.1 Phase 1 and Phase 2 processing stages

Table 5–2 describes the deployment stages of the DCU process.

Table 5–2: DCU deployment stages (1 of 2)

Phase	Stage Order	Stage Name	Description
Phase 1–Apply schema updates (DCU session)	1	PreDelta	Use this stage for processing that became necessary after the last schema update was applied and that needs to occur before the new schema update is loaded.
	2	Delta	Use this stage to specify the name of the delta file for upgrading the schema to the new version.
	3	PostDelta	Use this stage to specify any upgrade programs that need to take place before data is loaded into the schema. This is where most schema conversion programs are inserted.
	4	PreDataLoad ¹	This stage is rarely used. Use it to perform any processing that might be needed before data is loaded.
	5	DataLoad ¹	Use this stage to specify the list of .d files to be loaded after a schema update has been completed. For new databases, loading all the data from .d files is recommended, rather than using the ADO load mechanism.
	6	PostDataLoad ¹	This stage is rarely used. Use it to perform any processing that should take place after data is loaded into the database.

Table 5–2: DCU deployment stages

(2 of 2)

Phase	Stage Order	Stage Name	Description
Phase 2—Apply data updates (Full Progress Dynamics session)	7	PreADOLoad	Use this stage to modify data prior to the loading of ADO datasets. Note that these changes might be overwritten by the load of the ADOs. Occasionally, that might be what you want to happen, but always consider the possibility for data loss. Also note that the manager caches have not been built at this stage.
	8	ADOLoad	Use this stage to specify the names of the ADO dataset files which need to be loaded to upgrade the database. Note that it is possible to place upgrade programs between ADO loads so that upgrade programs can be run between the load of different ADOs.
	9	PostADOLoad	Use this stage to perform any updates that need to take place after the ADO datasets are loaded.
¹ The data referred to here is data from .d files, not ADOs. It is loaded through the 4GL IMPORT statement and generally relies on the Data Dictionary's load_df.p program for loading into the database tables.			

5.3.2 Logging for Phase 1 and Phase 2

The DCU provides logging for both the Phase 1 and Phase 2 stages, and by default it writes the log information to a file named `dcu.log` in the initial session's `PROPATH`. Note that if you change the path specified for the log file during Phase 1 to a path outside of the initial session's `PROPATH` or you change the name of the file from `dcu.log`, Phase 2 of the DCU cannot find the file and creates a second file named `dcu.log` in the session "Start in" directory. However, if more than one file named `dcu.log` appears in the session `PROPATH`, the DCU appends updated log information only to the first `dcu.log` file found in the `PROPATH`.

5.3.3 DCU processing sequence

When you design deployment programs, you must consider the sequence in which the DCU applies your programs. Deploying a single version of your application might require several passes through the DCU's processing stages. A *patch level* is all the programs that run during one pass.

The DCU processing sequence runs as follows:

1. The DCU parses all the XML files, loading the data into a temp table, and writing it to the Repository.
2. It sorts the temp table records by patch level and stage.
3. It processes Stages 1–6 for the first patch level.
4. It repeats process for the remaining patch levels in ascending order.
5. The DCU stops when it reaches one of the following end conditions:
 - A mandatory patches returns an error.
 - The DCU completes Stage 6 processing for the final patch level.
6. The DCU writes the status of the upgrade process and a list of patches that still need to be completed to the Repository.
7. When the next user who can change system data starts a Progress Dynamics session that has a direct database connection, the framework checks for information on unfinished upgrades in the Repository.
8. If there are unfinished upgrades to apply, the framework starts the DCU.
9. Beginning with the first unapplied patch level, the DCU works its way through Stages 7–9 for each patch level in ascending order.

5.3.4 Picking the correct processing stage

Plan your upgrades carefully. Think about the order in which to perform tasks in each patch level. For example, assume you are moving a field from one table to another. To save any existing data in that field, you have to spread the process over three upgrade programs as follows:

- 1 ♦ Add the field to the new table during Stage 2 of the current patch level.
- 2 ♦ Copy the data from the old field to the new field in either Stage 3 of the current patch level or in Stage 1 of the next patch level.
- 3 ♦ Drop the field from the old table during Stage 2 of the next patch level.

You should also think about how previous upgrade programs will be impacted each time you design a new patch level. Remember that the DCU runs through Stages 1–6 for all patch levels in Phase 1 and then completes Stages 7–9 for each patch level in Phase 2. Because of the break between the phases, the changes for a later upgrade could cause problems for an earlier upgrade. For example, Stage 2 of a later upgrade changes the format of the field into which the earlier upgrade loads data during Stage 7. The data in the earlier upgrade is now in the wrong format according to the database schema when the DCU begins Phase 2 processing.

In general, you should design future upgrades to avoid this sort of problem. If you cannot avoid such a problem, you need to run two different DCU sessions to complete the deployment. In the first DCU session, you apply all the patches levels before the level that would cause the conflict. Then, you design the second DCU session using the minimum version attribute so it only runs on databases that have all the previous patch levels applied.

5.4 DCU XML files

The DCU is driven by the data stored in a collection of XML files. The main file is the configuration file, `icfsetup.xml` by default. This file is a custom version of the standard Progress Dynamics configuration file, `icfconfig.xml`. The DCU uses the Configuration File Manager to parse the file. The configuration file contains details of the properties required to drive the session and the managers that need to be loaded.

Because the DCU is driven by the contents of XML files, you should not need to modify the DCU procedures themselves. By modifying the XML files, you can use the DCU to deploy your application databases as well as the Progress Dynamics Repository. You can design extra pages for the DCU using the page template, and hook them into the DCU process by adding some trigger code to the pages and some XML to the DCU driver file.

The DCU's standard configuration file looks like the following:

icfsetup.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<sessions>
  <session SessionType="ProgressSetup">
    <properties>
      <ICFCM_Database>DatabaseConnectionManager</ICFCM_Database>
      <physical_session_list>GUI,WBC</physical_session_list>
      <session_date_format>mdy</session_date_format>
      <session_year_offset>1950</session_year_offset>
      <session_numeric_format>,</session_numeric_format>
      <window_title>Progress Dynamics Configuration Utility</window_title>
      <valid_os_list>WIN32</valid_os_list>
      <setup_type_file>db/icf/dfd/setup1000B.xml</setup_type_file>
      <setup_type>ProgressSetup</setup_type>
    </properties>
    <managers>
      <manager>
        <cManagerName>ConnectionManager</cManagerName>
        <cFileName>af/app/afconmgrp.p</cFileName>
        <cHandleName>NON</cHandleName>
      </manager>
      <manager>
        <cManagerName>DatabaseConnectionManager</cManagerName>
        <cFileName>af/app/afdbconmgrp.p</cFileName>
        <cHandleName>NON</cHandleName>
      </manager>
      <manager>
        <cManagerName>InstallUIManager</cManagerName>
        <cFileName>install/prc/inuimngrp.p</cFileName>
        <cHandleName>NON</cHandleName>
      </manager>
      <manager>
        <cManagerName>ICFDBInstallManager</cManagerName>
        <cFileName>install/prc/inicfdbmng.p</cFileName>
        <cHandleName>NON</cHandleName>
      </manager>
    </managers>
  </session>
</sessions>
```

For your own deployments, you can add custom managers to the list in this file. The listed managers are started by the Connection Manager at the start of the DCU session. It is recommended that you do not alter the Install UI or the ICFDB Install Managers. Instead, use them as examples for building custom managers for your application databases.

5.4.1 DCU driver files

The configuration file specifies another file, the DCU driver file. That file contains the information needed to set up the DCU's wizard interface. You cannot use more than one DCU driver file during a DCU session.

Examine the default DCU driver file, `setup1000B.xml`. To customize the DCU for your own deployments, you can use the existing file as a template to add your own messages and other data. The beginning of the file contains basic information, including messages and system information, that is useful for gathering information during setup, as shown in this example:

setup1000B.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Setups>
  <setup SetupType="ProgressSetup">
    <ImageLowRes>install/img/imglores.bmp</ImageLowRes>
    <ImageHiRes>install/img/img hires.bmp</ImageHiRes>
    <IconFile>install/img/icfdu.ico</IconFile>
    <StartPage>Welcome</StartPage>
    <SkipButtons>Close,Cancel,Quit,Finish,Check</SkipButtons>
    <VersionNo>10.0B1P</VersionNo>
  . . .
  <message>
    <MessageCode>MSG_blank_path</MessageCode>
    <MessageDesc>The %1 path may not be left blank.</MessageDesc>
  </message>
  . . .
  <registrykey>
    <KeyName>reg_install_path</KeyName>
    <KeyValue>::Startup:DLC</KeyValue>
    <ExpandTokens>YES</ExpandTokens>
  </registrykey>
  . . .
  <path>
    <PathName>path_progress</PathName>
    <PathValue>#\\Startup\DLC#</PathValue>
    <ExpandTokens>YES</ExpandTokens>
  </path>
```

The `<registrykey>` nodes point the DCU to the Windows Registry for needed information. As shown by the `<path>` node, the information can then be used to derive other needed values. The Configuration File Manager parses the XML in a single pass. Therefore, the order of the nodes is important. The parser cannot expand a token if a node uses that token before the node that supplies the data is parsed.

Page nodes

The DCU driver file also contains the description of each page that the DCU displays and defines the actions that can occur on those pages. The following excerpt describes the DCU's **Installation Paths** page:

Page example from setup1000B.xml

```
<page Name="GetInstallPaths">
  <Title>Installation Paths</Title>
  <Group>Path</Group>
  <Proc>install/obj/inpath.w</Proc>
  <control>
    <Type>Editor</Type>
    <Name>edComment</Name>
    <DefaultValue>For this utility to set up the environment, it needs to know
the installation path that you chose during the first part of the installation.
It has attempted to derive the path from the install.

As the upgrade takes place, a log file will be written that contains details
of what occurred during the upgrade.

Please confirm that the Progress Dynamics Installation Path and Log File Name
below is correct.</DefaultValue>
  </control>
```

Note that the page is named and belongs to a named group. All the pages that cover a certain task should belong to the same group. The DCU uses the page name to target the correct page when an action is invoked. Because the pages are called by name, their nodes do not need to be listed in any order. This flexibility makes it easier for you to reuse individual pages in different sequences for different setup types. The `<Proc>` node gives the filename for the page. Each DCU page is created from the template, `src/install/intmplframe.w`.

Action nodes

Each <control> node describes an object on the page and how it behaves. The previous excerpt shows the description of the editor that appears in the page frame. The following excerpt shows the description of a **Cancel** button:

Cancel button from setup1000B.xml

```
<control>
  <Type>Button</Type>
  <Panel>Yes</Panel>
  <Label>&Cancel</Label>
  <Name>Quit</Name>
  <Justify>Right</Justify>
  <action>
    <Event>CHOOSE</Event>
    <Action>QUIT</Action>
  </action>
</control>
```

The <action> node describes how a control reacts to one of the standard Progress 4GL events, which is listed in the <Event> subnode. The <Action> subnode can only contain a single character string, the name of the API that is launched by the event. The DCU assumes that the API is in the Install UI Manager by default. If you want to run an API from another manager, you can specify the manager using an <ActionTarget> node. The value in <ActionTarget> must match the named assigned to the manager in the <cManagerName> node in the DCU configuration file.

The DCU recognizes two special keywords in addition to any APIs in the managers: QUIT and FINISH. QUIT ends the DCU session without completing the upgrade. FINISH ends the DCU and signals that the upgrade completed successfully. The difference is important for release versioning.

While most actions are linked to events for specific controls, you can also attach actions directly to a page. The following excerpt shows an event linked directly to the **GetICFDBParams** page:

Action linked to a page from setup1000B.xml

```
<page Name="GetICFDBParams">
  <Title>ICFDB Parameters</Title>
  <Group>ICFDB</Group>
  <Proc>install/obj/indbconn.w</Proc>
  <action>
    <Event>INITIALIZE</Event>
    <Action>checkForDB</Action>
    <ActionParam>ICFDB</ActionParam>
  </action>
```

Action parameters

The following excerpt shows the description for the **Back** button on the **Installation Paths** page. Its two actions are processed in the order in which they are listed:

Back button from setup1000B.xml

```
<control>
  <Type>Button</Type>
  <Panel>Yes</Panel>
  <Label>&Back</Label>
  <Name>Back</Name>
  <Justify>Right</Justify>
  <action>
    <Event>CHOOSE</Event>
    <Action>restoreProperties</Action>
  </action>
  <action>
    <Event>CHOOSE</Event>
    <Action>gotoPage</Action>
    <ActionParam>Welcome</ActionParam>
  </action>
</control>
```

The second action shows an example of passing a parameter. In this excerpt, it is the name of the page which the DCU will display when the **Back** button is chosen. An action can only take a single parameter, which must be a character string.

However, the DCU recognizes two special strings for passing conditional parameters. These strings allow you to embed IF...THEN...ELSE and CASE expressions in the XML. [Table 5–3](#) lists the operators that you can use in the expressions.

Table 5–3: Operators for conditional expressions

Operator	Symbol
AND	&&
OR	::
Equals	=
Greater than or equal	>=
Less than or equal	<=
Not equal	<>

An example of the syntax for an IF expression is shown in this excerpt from the **GetICFDBParams** page:

IF...THEN...ELSE conditional expression

```
<action>
  <Event>CHOOSE</Event>
  <Action>gotoPage</Action>
  <ActionParam>?db_build_icfdb=YES;GetICFSiteData;GetICFDBPatches
</ActionParam>
</action>
```

The IF expression begins with a question mark (?) followed by the expression the DCU evaluates. Semicolons (;) separate the results. The value after the first semicolon is the THEN result, and the value after the second semicolon is the ELSE result. So, the expression above reads “If the value of db_build_icfdb is YES, then go to the GetICFSiteData page, else go to the GetICFDBPatches page.”

NOTE: While you can create complex IF expressions using the available operators, you cannot nest IF expressions. The results must be single character strings as with normal parameters.

An example of the syntax for a CASE expression is shown below:

CASE conditional expression

```
<action>
  <Event>CHOOSE</Event>
  <Action>gotoPage</Action>
  <ActionParam>:session_date_format:dmy|page1:mdy|page2:ymd|page3:default|page9
</ActionParam>
</action>
```

The CASE expression begins with a colon (:) and the name of the property to evaluate. Each result begins with a colon followed by a pipe (|) delimited value pair. The DCU uses the first value pair that matches the evaluated property in the list of results. So, the expression above reads, “Check the session_date_format property. If the value is dmy, go to page1. If the value is mdy, go to page2. If the value is ymd, go to page3. If none of the conditions are met, go to page9.”

Data capture nodes

The DCU captures data from its wizard interface with a screen scrape procedure. How the data is stored is controlled through the use of two nodes: <StoreTo> and <TableVariable>. The following excerpt shows a fill-in field description that employs both methods:

Fill-in field from setup1000B.xml

```
- <control>
  <Name>fiPath2</Name>
  <DefaultValue>#_start_in_directory#\dcu.log</DefaultValue>
  <ExpandTokens>YES</ExpandTokens>
  <StoreTo>path_log</StoreTo>
  <Label>Log File</Label>
  <TableVariable>LogFile</TableVariable>
</control>
```

Which method you use to capture the data depends on when you want to have the data available. The DCU stores the data from a <StoreTo> node in a session parameter. The parameter can be defined in either the Configuration File Manager or in the DCU driver file. The data is then available to the DCU while the user steps through the rest of the wizard pages. The DCU stores the data from a <TableVariable> node in a temp-table. The data in the temp-table is used after the DCU has finished collecting data and starts processing. As the example shows, you can store the data in both places if you need it both before and after the DCU begins processing.

Database nodes

The last section of the standard DCU driver file describes the databases to upgrade. Each database has its own node. The DCU works through the databases in the order they are listed here. A database node specifies the information needed to connect to the database and lists a set of XML files that contain descriptions of the upgrades and datasets to be applied to the database.

The following excerpt shows the contents of a database node:

Database example from setup1000B.xml

```
<database>
  <DBName>ICFDB</DBName>
  <VersionSeq>seq_icfdb_dbversion</VersionSeq>
  <MinimumVersion>020021</MinimumVersion>
  <ConnectParams>-1</ConnectParams>
  <DBDir>#path_db#\icfdb\icfdb.db</DBDir>
  <DBDump>#path_src#\#dynamics_rootname#\db\icf\dump</DBDump>
  <patch PatchLevel="0" DBBuild="Yes" NodeURL="db/icf/dfd/icfdbbuild.xml"/>
  <patch PatchLevel="020022" NodeURL="db/icf/dfd/icfdb020022adolist.xml"/>
  <patch PatchLevel="020022" NodeURL="db/icf/dfd/icfdb020022patch.xml"/>
  <patch PatchLevel="020023" NodeURL="db/icf/dfd/icfdb020023patch.xml"/>
  <patch PatchLevel="020024" NodeURL="db/icf/dfd/icfdb020024patch.xml"/>
  <patch PatchLevel="020025" NodeURL="db/icf/dfd/icfdb020025patch.xml"/>
  <patch PatchLevel="020025" NodeURL="db/icf/dfd/icfdb020025adolist.xml"/>
  <patch PatchLevel="020026" NodeURL="db/icf/dfd/icfdb020026patch.xml"/>
  <patch PatchLevel="100001" NodeURL="db/icf/dfd/icfdb100001patch.xml"/>
  <patch PatchLevel="100002" NodeURL="db/icf/dfd/icfdb100002patch.xml"/>
  <patch PatchLevel="100002" NodeURL="db/icf/dfd/icfdb100002adolist.xml"/>
</database>
```

This example shows the information for setting up the Repository. It is recommended that you use the same technique for tracking database version for your application databases. To enable identification of the current version of a database, each database requires a sequence with a name in the following format:

```
seq_logicaldbname_DBVersion
```

Where *logicaldbname* is the logical database name for the database.

This sequence tracks the version of the database as a six digit integer. The first two digits are the version. The next two digits are the revision level. The final two digits are the patch level.

The current value of the sequence is not used, but the DCU reads the sequence maximum value to determine the current database version. The DCU uses this value to decide which patches must be applied to the target database to complete the upgrade. When a database definition delta file is loaded, the database definition should update the sequence definition to its new maximum value. The `<MinimumVersion>` node contains the minimum version of the database on which a particular setup can run.

The remaining nodes list the patch levels to apply during this upgrade. The associated files detail all the patch programs and ADOs that are part of a particular deployment. Notice that they are listed in ascending order by patch levels. Your files should always be listed in this order. The DCU reads the files in the order they are listed to build the temp-table of patches.

5.4.2 Upgrade files

The XML files that specify the changes the DCU applies to the Repository and application databases are the upgrade files. Each upgrade file is assigned to a patch level that controls when the DCU applies the upgrade. The data in an upgrade file should generally be limited to one of the following categories:

- New database creation data.
- A list of all ADOs to load during a given patch level.
- Upgrade programs, including:
 - Schema changes.
 - Data loads.
 - Fix programs.

After patch level “0”, the next patch level is the lowest one that can be applied during the upgrade. A patch level can have multiple files associated with it. Upgrade files are listed in a specific order under the database node. All the upgrade files for a given patch file are listed together. The DCU parses the files in the order they are listed under the database node.

Database creation upgrade files

The first patch level listed under the database nodes controls the creation of new databases. The new database creation patch level is always numbered “0”. The associated upgrade file provides the details building a new database, as in the standard Repository build file shown below:

icfdbbuild.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<SetupInclude>
  <Patch PatchLevel="0">
    <patchstage UpdateStage="Delta">
      <Program>
        <FileType>df</FileType>
        <FileName>db/icf/dfd/icfdbfull.df</FileName>
        <Description>Applying Full DB Schema</Description>
        <Rerunnable>no</Rerunnable>
        <NewDB>yes</NewDB>
        <ExistingDB>no</ExistingDB>
        <UpdateMandatory>yes</UpdateMandatory>
      </Program>
    </patchstage>
    <patchstage UpdateStage="DataLoad">
      <Program>
        <FileType>d</FileType>
        <FileName />
        <Description>Loading database table contents</Description>
        <Rerunnable>no</Rerunnable>
        <NewDB>yes</NewDB>
        <ExistingDB>no</ExistingDB>
        <UpdateMandatory>yes</UpdateMandatory>
      </Program>
      <Program>
        <FileType>s</FileType>
        <FileName />
        <Description>Setting site number</Description>
        <Rerunnable>yes</Rerunnable>
        <NewDB>yes</NewDB>
        <ExistingDB>no</ExistingDB>
        <UpdateMandatory>yes</UpdateMandatory>
      </Program>
    </patchstage>
  </Patch>
</SetupInclude>
```

There are several points to observe in this file. an upgrade file can contain information on several programs that run at different stages. The patch stages must be listed in the correct order. The programs within a patch stage must be listed in the order in which you want them to run.

A peculiarity of this file is that it does not list specific file names for the programs in the DataLoad stage. The DCU automatically loads all the files of the appropriate type in the target directory.

NOTE: For an explanation of the program attribute nodes, see the [“Upgrade program attributes”](#) section.

After patch level “0”, the other patch levels are listed in ascending order from the lowest one that can be applied during the upgrade. A patch level can have multiple files associated with it. The files are listed in the order the DCU should apply them.

ADO list upgrade files

The ADO list upgrade file is usually the largest upgrade file. This file contains a list of all the ADOs that are applied during a patch level. The ADO list upgrade file is created by the release versioning tools. The tools automatically list the ADOs in the order in which they should be applied. For more information on the release versioning tools, see the *Progress Dynamics Version 2.1A Application Deployment* white paper on the following PSDN Web site:

<http://psdn.progress.com>

NOTE: The release versioning tools build a completely new ADO list file each time. Any edits you made in a previous ADO list are lost, so generally you should not customize the file.

The following is an excerpt from an ADO list file:

icfdb100002adolist.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<SetupInclude>
  <Patch PatchLevel="100002">
    <PatchStage Stage="ADOLoad">
      <Program>
        <FileType>ADO</FileType>
        <FileName>db/icf/dump/gscem.ado</FileName>
        <Description>Loading ADO for db/icf/dump/gscem.ado</Description>
        <Rerunnable>yes</Rerunnable>
        <NewDB>no</NewDB>
        <ExistingDB>yes</ExistingDB>
        <UpdateMandatory>yes</UpdateMandatory>
      </Program>
      <Program>
        <FileType>ADO</FileType>
        <FileName>db/icf/dump/gscer.ado</FileName>
        <Description>Loading ADO for db/icf/dump/gscer.ado</Description>
        <Rerunnable>yes</Rerunnable>
        <NewDB>no</NewDB>
        <ExistingDB>yes</ExistingDB>
        <UpdateMandatory>yes</UpdateMandatory>
      </Program>
    </PatchStage>
  </Patch>
</SetupInclude>
```

Other upgrade files

The other upgrade files that might be part of a patch level detail the steps to upgrade an existing database, including schema updates, data loads through ADOs, and programs to fix existing data. A file can include several procedures happening at various stages of the upgrade process.

The following example shows the XML structure for applying a schema change and running a procedure file:

icfdb020026patch.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<SetupInclude>
  <Patch PatchLevel="020025">
    <PatchStage Stage="Delta">
      <Program>
        <FileType>df</FileType>
        <FileName>db/icf/dfd/icfdb020025delta.df</FileName>
        <Description>Applying 020025 Delta</Description>
        <Rerunnable>no</Rerunnable>
        <NewDB>no</NewDB>
        <ExistingDB>yes</ExistingDB>
        <UpdateMandatory>yes</UpdateMandatory>
      </Program>
    </PatchStage>
    <PatchStage Stage="PreADOLoad">
      <Program>
        <FileType>p</FileType>
        <FileName>db/icf/dfd/fixrmdhtmlattr.p</FileName>
        <Description>Removing old DHTML attribute deletions</Description>
        <Rerunnable>yes</Rerunnable>
        <NewDB>no</NewDB>
        <ExistingDB>yes</ExistingDB>
        <UpdateMandatory>no</UpdateMandatory>
      </Program>
    </PatchStage>
  </Patch>
</SetupInclude>
```

5.5 DCU API

The procedure files that contain most of the code specific to the DCU are located in your *install-dir\src\icf\install\prc* directory. [Table 5–4](#) lists the DCU’s main procedure files.

Table 5–4: DCU procedure files

File	Description
<i>inuimngrp.p</i>	Handles the UI events that run the DCU wizard.
<i>inupgrdapip.p</i>	Handles the data definition and data loads. (Phase 1 processing)
<i>insessupdp.p</i>	Handles the dataset (ADO) loads. (Phase 2 processing)
<i>insaxparserp.p</i>	Reads the XML files.
<i>inrytupdstatp.p</i>	Stores details of the upgrade process in the Repository.
<i>inicfdbsitep.p</i>	Sets the Repository’s site number.
<i>inicfdbsetseqp.p</i>	Increments the Repository’s version sequence during each upgrade.
<i>inicfdbmgrp.p</i>	Creates and updates new repositories.
<i>inicfdbgetseqp.p</i>	Retrieves the current value of the Repository’s version sequence.

The two manager procedures, *inuimngrp.p* and *inicfdbmgrp.p*, can serve as examples for your custom deployment managers. These two managers also contain all the APIs that you should normally need to make calls to. The following sections describe the public APIs.

5.5.1 analyzeCase

This function analyzes a CASE expression from an XML <ActionParam> node and returns the appropriate value to use for the parameter.

A CASE expression begins with a colon (:) and the name of the property to evaluate. Each result begins with a colon followed by a pipe (|) delimited value pair. The function stops processing when it reaches the first value pair that matches the evaluated property in the list of results. So, the following expression:

```
:session_date_format:dmy|page1:mdy|page2:ymd|page3:default|page9
```

Evaluates as “Check the session_date_format property. If the value is dmy, go to page1. If the value is mdy, go to page2. If the value is ymd, go to page3. If none of the conditions are met, go to page9.”

Location: install\prc\inuimngrp.p

Parameters:

INPUT pcCaseStatement AS CHARACTER

The CASE expression from the XML <ActionParam> node.

Returns: Character

Notes: None

5.5.2 **analyzef**

This function analyzes an IF expression from an XML <ActionParam> node and returns either the THEN or ELSE value to use for the parameter.

The IF expression begins with a question mark (?) followed by the expression the DCU evaluates. Semicolons (;) separate the results. The value after the first semicolon is the THEN result, and the value after the second semicolon is the ELSE result. The following expression evaluates as “If the value of db_build_icfdb is YES, then go to the GetICFSiteData page, else go to the GetICFDBPatches page.”:

```
?db_build_icfdb=YES;GetICFSiteData;GetICFDBPatches
```

Location: install\prc\inuimngrp.p

Parameters:

INPUT pcIfStatement AS CHARACTER

The IF expression from the XML <ActionParam> node.

Returns: None (procedure).

Notes: While you can create complex IF expressions using the available operators, you cannot nest IF expressions. The results must be single character strings as with standard parameters.

5.5.3 **btnChoose**

This procedure handles the CHOOSE event for a button on one of the DCU’s wizard pages. It calls the eventProc procedure supplying the proper parameters for a button.

Location: install\prc\inuimngrp.p

Parameters: None.

Returns: None (procedure).

Notes: None.

5.5.4 checkForDB

This procedure checks to see if the databases exist. If the databases exist, it sets the following properties:

- **connect_params_dbname** — The connection parameters for the database as derived from the <database> node.
- **dbname_exists** — Indicates if the database exists on disk.
- **dbname_does_not_exist** — Indicates that the database does not exist on disk.

Where **dbname** is the name of the specific database.

Location: install\prc\inuimngrp.p

Parameters:

INPUT pcDBToCheck AS CHARACTER

 The name of the database to check.

Returns: None (procedure).

Notes: The two opposite properties for whether the database exists are need because you cannot evaluate NOT for a property and have the token expanded.

5.5.5 connectDatabase

This procedure confirms that the database is available and can be connected. It then registers the connection with the Connection Manager.

Location: install\prc\inuimngrp.p

Parameters:

INPUT pcDBName AS CHARACTER

 The name of the database to check.

Returns: None (procedure).

Notes: None

5.5.6 eventProc

This procedure handles the event processing for objects on the DCU's wizard pages.

Location: install\prc\inuimngrp.p

Parameters:

INPUT pcEvent AS CHARACTER

The Progress 4GL event.

INPUT pcObject AS CHARACTER

The name of the object.

Returns: None (procedure).

Notes: None

5.5.7 getDBFile

This procedure calls SYSTEM-DIALOG GET-FILE to find a database file name.

Location: install\prc\inuimngrp.p

Parameters:

INPUT pcControl AS CHARACTER

The name of the control.

Returns: None (procedure).

Notes: None

5.5.8 getDirectory

This procedure calls the Microsoft standard dialog to prompt for a directory.

Location: install\prc\inuimngrp.p

Parameters:

INPUT pcPathField AS CHARACTER

The name of the fill-in field.

Returns: None (procedure).

Notes: None

5.5.9 gotoPage

This procedure loads a named page into the DCU's wizard interface.

Location: install\prc\inuimngrp.p

Parameters:

INPUT pcPageName AS CHARACTER

The name of a page as specified in the XML file.

Returns: None (procedure).

Notes: The conditional expressions described in the [“Action nodes”](#) section are often used to decide the page to call with this API.

5.5.10 obtainICFSeqVals

This procedure obtains the sequence values for the ICFDB database and populates the UI with the data.

Location: install\prc\inicfdbmgrp.p

Parameters:

INPUT pcInput AS CHARACTER

This parameter is not currently used.

Returns: None (procedure).

Notes: None

5.5.11 obtainPatchList

This procedure obtains the list of upgrade files for a certain database.

Location: install\prc\inuimngrp.p

Parameters:

INPUT pcParams AS CHARACTER

Returns: None (procedure).

Notes: None

5.5.12 processParams

This procedure copies the data from the ttControl records into the ttValue table. It also copies the value of all the database session parameters into the ttDatabase records.

Location: install\prc\inuimngrp.p

Parameters:

INPUT pcParams AS CHARACTER

This parameter is not currently used.

Returns: None (procedure).

Notes: None

5.5.13 restoreProperties

This procedure restores cascaded properties to their original values.

Location: install\prc\inuimngrp.p

Parameters:

INPUT pcInput AS CHARACTER

This parameter is not currently used.

Returns: None (procedure).

Notes: None

5.5.14 screenScrape

This procedure goes through all the fields in the ttControl temp-table for the current page and stores the value from the screen into the ttControl.cFieldValue field.

Location: install\prc\inuimngrp.p

Parameters:

INPUT pcInput AS CHARACTER

This parameter is not currently used.

Returns: None (procedure).

Notes: None

5.5.15 startUpgradeProcess

This procedure launches the upgrade process using the private upgrade APIs.

Location: install\prc\inuimngrp.p

Parameters:

INPUT pcEditor AS CHARACTER

Returns: None (procedure).

Notes: None

5.5.16 validateDirectory

This procedure validates that the directory exists and can be created if necessary.

Location: install\prc\inuimngrp.p

Parameters:

INPUT pcParamString AS CHARACTER

A comma-delimited value pair. The first entry is the name of the fill-in field that contains the directory name to validate. The second entry is an optional TRUE/FALSE value that indicates whether a blank filename is permitted. The default for the second entry is FALSE.

Returns: None (procedure).

Notes: None

5.5.17 validateSiteNumber

This procedure validates that the value for Progress Dynamics site number field is not zero.

Location: install\prc\inicfdbmgrp.p

Parameters:

INPUT pcWidget AS CHARACTER

Name of the fill-in field that holds the site number.

Returns: None (procedure).

Notes: None

5.5.18 verifyDBVersion

This procedure determines whether the database version of the database being upgraded is valid.

Location: install\prc\inuimngrp.p

Parameters:

INPUT pcParams AS CHARACTER

Returns: None (procedure).

Notes: None

5.6 Creating a customized DCU deployment

The following sections present advice on customizing a DCU session to deploy your own applications. Remember that every deployment is unique. There is not a single correct solution for all deployments.

5.6.1 Prepare schema and data files

Every deployment should be able to build the databases from scratch for a new user and also bring existing databases up to the latest version. Therefore, you need to prepare the following files from your central Repository and application databases:

- A full schema definition (.df) file.
- Complete data (.d) files.
- Incremental schema definition (.df) files.
- Dataset (ADO) files.

NOTE: For DCU deployments, you should follow the Progress Dynamics naming conventions for .df files as shown in the [Progress Dynamics Repository Reference](#).

The DCU uses the full .df file and the .d files to create a new database. Because .d files load faster than ADOs, you should use .d files when loading data into a new database. However, the data in ADOs is independent of the database's session properties. When using .d files, you need to take into account those session properties.

The DCU uses the OpenEdge defaults for the session properties listed in [Table 5–5](#).

Table 5–5: DCU session properties

Property	Default value
DATE-FORMAT	MDY
YEAR-OFFSET	1950
CODEPAGE	UTF-8

When creating files for your deployment, you should set these properties to the values for the environment for which you are creating the deployment. Then, you should customize a DCU session to use the same values. For more information on setting session properties, see the information on creating session properties in [Chapter 2, “Defining and Managing Sessions.”](#)

For more information on creating .df and .d files, see *Progress Database Administration Guide and Reference*.

Remember that it is best to create these files from a clean copy of your Repository and application databases. Make all the changes you can on your central databases and then deploy those changes to your users through ADOs. In general, you should only run fix programs during a user deployment if absolutely necessary. If the changes can be done on the central databases and the results deployed as ADOs, do it that way.

5.6.2 Designing upgrade programs

Because of how the DCU processes upgrades, all upgrade programs must conform to the following requirements:

- **Runs non-persistent** — An upgrade program runs non-persistently. The DCU acquires a schema lock on the databases it is updating during Phase 1. Persistent procedures with database access code might cause problems when .df files are loaded. If an upgrade program invokes any persistent procedures, especially ones that access the database, the upgrade program must also shut down those procedures.
- **Uses no parameters** — An upgrade program takes no parameters.
- **Runs without user interaction** — Upgrade programs run as background procedures in a non-interactive mode. There can be no user interaction inside an upgrade program.
- **Generates a RETURN at completion** — All upgrade programs should end the main block with either a RETURN or a RETURN ERROR.

- **Upgrade fails on RETURN ERROR** — When an error condition is raised and returned to the DCU, the DCU treats it as a failed condition. The DCU stops processing and quits the upgrade. This is an extreme way to end an upgrade program and should only be used for fatal errors. RETURN ERROR should only be used to end the upgrade process.
- **Writes to the log file** — The log file is the only recording of what took place during a DCU session. Therefore, it is very important that status information is regularly written to the log file. You can update the log file using a PUBLISH statement, like this:

```
PUBLISH "DCU_WriteLog":U ("Message").
```

- **Handle schema triggers** — In general, it is not good practice to override schema triggers. Overriding the triggers can result in data problems. However, there are certain cases where you must override the triggers, such as with upgrade programs that run without the complete Progress Dynamics framework during Phase 1. In these cases, the upgrade program should perform the tasks that the schema triggers would normally perform. You will need to supply defaults for information that the triggers would normally obtain through calls to managers.
- **Handle cache issues** — During Phase 2, remember that the Progress Dynamics environment caches data in many places. Loading ADOs can invalidate the existing caches. Any of your upgrade procedures that make use of the framework's cached data should refresh the cache before using it. This should correct problems caused by upgrade procedures that ran before it.

5.6.3 Upgrade program attributes

Every program run by the DCU has several standard attributes. These are set in subnodes under the <program> node. [Table 5–6](#) describes their use.

Table 5–6: Standard program properties

Property	Description
Rerunnable	Indicates whether the program can be rerun on the same database if the DCU processing is interrupted and must be restarted. NOTE: Before you define an upgrade as rerunnable, be sure it will not make changes to data that it changed on previous runs.
NewDB	Indicates whether the upgrade program should be applied to a newly created database.
ExistingDB	Indicates whether the upgrade program should be applied to an existing database.
UpdateMandatory	Indicates whether the upgrade program must complete successfully. If YES and the program returns an error condition, the DCU stops processing and treats the upgrade session as a failure. If NO and the program returns an error condition, the DCU logs a note on the failure of that program and continues processing the other upgrades.

5.6.4 Setting up new databases with the DCU

The main focus of the DCU is setting up the Repository database. Therefore, it is simple to extend a DCU session to also set up your application database as well. Every deployment should be able to create a new database. You can customize a deployment to create a new application database as follows:

- 1 ♦ Create a custom session type and generate a configuration file for it.
- 2 ♦ Create a custom DCU driver file for the new session type.
- 3 ♦ Add a page to the DCU driver file to collect install information.
- 4 ♦ Add a node for the new database in the DCU driver file.
- 5 ♦ Create a database creation upgrade file for your application database.

Creating a custom session type

It is good practice to have a separate session type for each application you deploy. The Session Type Control enables you to extend existing session types to create customized ones.

To create a new DCU session type:

- 1 ♦ In the **Session Type Maintenance** window, select the **Basic→DCU→ProgressSetup** node.
- 2 ♦ Choose **Add** to create a new session type. The new session type automatically extends ProgressSetup.
- 3 ♦ Name the new session type and save it. For example, a session type for the DynSports sample application might be **DynSportsSetup**.
- 4 ♦ Create overrides for the ProgressSetup session properties under the new session type. For example:

Session property	Sample value
setup_type	DynSportsSetup
setup_type_file	db/dynsports/dfd/DynsportsSetup01.xml
window_title	DynSports Configuration Utility

- 5 ♦ Generate a configuration file using your new session type. By convention, the filename would be *applicationsetup.xml*, for example, *dynsportssetup.xml*.

By changing the DCUSETUPTYPE and ICFCONFIG parameters to your new session type and configuration file, you can now override the DCU with your custom deployment.

Creating a custom DCU driver file

Your custom configuration uses the DCU driver file that you named in the setup_type_file session property. Your next step is to create that file. While you could build one from scratch, you can also alter a standard DCU driver file.

To create a custom DCU driver file:

- 1 ♦ Save a copy of the latest standard DCU driver file to the directory and filename specified in the `setup_type_file` session property of your custom session type.
- 2 ♦ Change the `<setup>` node to your custom session type. For example:

```
<setup SetupType="DynSportsSetup">
```

- 3 ♦ In the **Welcome** page description, change the contents of the editor control's `<DefaultValue>` to reflect your application. For example, you might change the standard text to the following for the DynSports application:

```
Welcome to the DynSports Configuration Utility.

This process will set up the databases that are required to run the
current version of DynSports.

Choose the Next button to start this process. Choose Cancel to leave
without modifying anything.
```

- 4 ♦ You can also alter the settings of the following nodes to change the appearance of the DCU to reflect your own application:
 - `<ImageLowRes>`
 - `<ImageHiRes>`
 - `<IconFile>`

If you run your custom DCU session now, it brings up the altered Welcome page. It cannot install your application database yet. It needs to gather path information on where the database files are located.

Adding a page to the DCU

The DCU needs to know where the source files for your application database are. One of the existing DCU pages, the **ICFDB Parameters** page, captures this type of information for the Repository. You can reuse it to capture the information for your application database.

The page you are reusing supports upgrading existing databases, as well as creating new databases. This section concentrates on those parts of the page that build a new database. Using the DCU for upgrades is discussed in the [“Upgrading existing databases with the DCU”](#) section.

To add a new page to the DCU driver file:

- 1 ♦ In your DCU driver file, copy the GetICFDBParams page description. It begins with the `<page Name="GetICFDBParams">` node and ends with the next `</page>` node.
- 2 ♦ Paste a copy of the page description before the **Status** page description.
- 3 ♦ Change the page name, title, and page group to reference your application. For example, the beginning of the DynSports page description would look like this:

```
<page Name="GetDynSportsParams">
  <Title>DynSports Parameters</Title>
  <Group>DynSports</Group>
```

- 4 ♦ Change checkForDB action to use the logical name of you application database. For the DynSports example, it would be the following:

```
<action>
  <Event>INITIALIZE</Event>
  <Action>checkForDB</Action>
  <ActionParam>DynSports</ActionParam>
</action>
```

- 5 ♦ Set the **Back** button to go to the **ICFDB Parameters** page. For the DynSports example, it would be the following:

```
<control>
  <Type>Button</Type>
  <Panel>Yes</Panel>
  <Label>&Back</Label>
  <Name>Back</Name>
  <Justify>Right</Justify>
  <action>
    <Event>CHOOSE</Event>
    <Action>gotoPage</Action>
    <ActionParam>GetICFDBParams</ActionParam>
  </action>
</control>
```

NOTE: There are several paths through the ICFDB pages. With conditional processing, you could have the DCU return to the page from which the user came.

- 6 ♦ Set the **Next** button to go to the **Status** page in the Wizard sequence.

You do not need to modify the other actions that occur when the user chooses the **Next** button. The `screenScrape` action records the values from the objects. The `connectDatabase` and `verifyDBVersion` actions are conditional and will not occur during the build of a new database.

- 7 ♦ Change the text in the editor's default value to describe your application database.
- 8 ♦ Change the remaining ICFDB references to refer to your application database.

Your customized DCU can now gather the information on where to find the build files for your application database. The last thing you need is an upgrade file that tells how to use the files.

Adding a database node

The DCU driver file needs a node for each database in the deployment.

To add a node for your application database, copy the ICFDB node and modify the values for your application database. For example, a DynSports database node might look like this:

```
<database>
  <DBName>DynSports</DBName>
  <VersionSeq>seq_dynsports_dbversion</VersionSeq>
  <MinimumVersion>010000</MinimumVersion>
  <ConnectParams>-1</ConnectParams>
  <DBDir>#path_db#\dynsports\dynsports.db</DBDir>
  <DBDump>#path_src#\#dynamics_rootname#\db\dynsports\dump</DBDump>
  <patch PatchLevel="0" DBBuild="Yes"
    NodeURL="dynsports/dfd/dynsportsbuild.xml"/>
</database>
```

Creating a database creation upgrade file

As before, the simplest method for creating the upgrade file is to copy and customize the standard file.

To create a database creation upgrade file:

- 1 ♦ Copy and rename the Repository creation upgrade file, `icfdbbuild.xml`. For example, the DynSports file would be `dynsportsbuild.xml`.
- 2 ♦ Change the full schema definition filename to the one for your application database. For the DynSports example, it might look like this:

```
<patchstage UpdateStage="Delta">
  <Program>
    <FileType>df</FileType>
    <FileName>dynsports/dfd/dynsportsfull.df</FileName>
    <Description>Applying Full DynSports DB Schema</Description>
    <Rerunnable>no</Rerunnable>
    <NewDB>yes</NewDB>
    <ExistingDB>no</ExistingDB>
    <UpdateMandatory>yes</UpdateMandatory>
  </Program>
</patchstage>
```

- 3 ♦ In the next patch stage, remove the program node for the “Setting Site Number” program. That program is specific to the Repository.

Your custom DCU can now create a new application database in addition to creating and upgrading the Repository.

5.6.5 Upgrading existing databases with the DCU

Deployments need to upgrade existing databases as well as create new ones. Modifying the DCU to upgrade your application database is not difficult. The most important part of the process is planning when to apply each upgrade. You can apply incremental schema definition files, fix programs, or load ADOs to upgrade existing databases.

Patch levels are tracked using the database sequences described in the [“Database nodes”](#). Even if you have not changed anything else in the schema, that sequence gets incremented for each patch level. That means that each patch level includes the application of an incremental .df file.

A deployment can apply several patch levels to a database. The database node for each database specifies a minimum version that sets a lower bound on the existing databases that can be upgraded by a deployment.

The general process for adding upgrades to your customized DCU is as follows:

- 1 ♦ Plan on how to properly apply your upgrades. In particular, consider when the upgrade should happen during the DCU process.
- 2 ♦ Create an upgrade file for the patch level. For example, a DynSports upgrade file might look like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<SetupInclude>
  <Patch PatchLevel="010002">
    <PatchStage Stage="Delta">
      <Program>
        <FileType>df</FileType>
        <FileName>dynsports/dfd/dynsports010002delta.df</FileName>
        <Description>Applying DynSports 010002 Delta</Description>
        <Rerunnable>no</Rerunnable>
        <NewDB>no</NewDB>
        <ExistingDB>yes</ExistingDB>
        <UpdateMandatory>yes</UpdateMandatory>
      </Program>
    </PatchStage>
    <PatchStage Stage="PreADOLoad">
      <Program>
        <FileType>p</FileType>
        <FileName>dynsports/dfd/removeinvalidlinks.p</FileName>
        <Description>Removing invalid links</Description>
        <Rerunnable>yes</Rerunnable>
        <NewDB>no</NewDB>
        <ExistingDB>yes</ExistingDB>
        <UpdateMandatory>yes</UpdateMandatory>
      </Program>
    </PatchStage>
    <PatchStage Stage="ADOLoad">
      <Program>
        <FileType>ADO</FileType>
        <FileName>dynsports/dump/armcu.ado</FileName>
        <Description>Loading ADO for Customer table.</Description>
        <Rerunnable>yes</Rerunnable>
        <NewDB>no</NewDB>
        <ExistingDB>yes</ExistingDB>
        <UpdateMandatory>yes</UpdateMandatory>
      </Program>
    </PatchStage>
  </Patch>
</SetupInclude>
```

- 3 ♦ Use the release versioning tools to create ADO List upgrade file if any ADOs are being applied for the patch level.

- 4 ♦ Add the patch level description to the DCU driver file. For example, a DynSports patch level description might look like this:

```
<database>
  <DBName>DynSports</DBName>
  <VersionSeq>seq_dynsports_dbversion</VersionSeq>
  <MinimumVersion>010000</MinimumVersion>
  <ConnectParams>-1</ConnectParams>
  <DBDir>#path_db#\dynsports\dynsports.db</DBDir>
  <DBDump>#path_src#\#dynamics_rootname#\db\dynsports\dump</DBDump>
  <patch PatchLevel="0" DBBuild="Yes"
    NodeURL="dynsports/dfd/dynsportsbuild.xml"/>
  <patch PatchLevel="010002"
    NodeURL="dynsports/dfd/dynsports010002patch.xml"/>
</database>
```

- 5 ♦ Test each new patch level to see how it interacts with previous upgrades.

5.6.6 Tips for customized DCU sessions

The following are some points to remember while customizing a DCU session for your deployment:

- Follow the flow of the standard DCU session. Collect all the information you need to complete the deployment first, then start processing. This approach will become mandatory in future versions of the DCU.
- The order in which you make changes is critical. For example, applying datasets can overwrite existing user data. If you need to save the existing data, run a program to move the data to a temporary storage area before applying the datasets that would overwrite the data. Then run another program to add the saved data back to the appropriate tables.
- Because the full Progress Dynamics environment is not available during Phase 1, it is impossible to execute schema triggers when data is written. Procedures running during this phase should alter as few records and as little data as possible.
- In general, the only upgrade procedures that should run during Phase 1 are those needed to migrate data from one schema version to another.
- Test how each change to your application affects your deployment. Consider changing the application design to avoid complex fixes during deployment. Designing your application and deployment should be part of the same iterative cycle.

- Examine all the patch levels for a given deployment for potential conflicts. Each time you add a new patch procedure or ADO, you need to verify that it does not cause problems for any of the earlier patches.
- In general, you should use ADOs for your deployments. When possible, run all fix programs on a given set of data in your central development Repository. Then generate the ADOs for your deployment. You should avoid using fix programs in deployments whenever possible.

5.6.7 Starting the DCU

The DCU can be started either from a desktop shortcut or from the command line so that it can be run from third-party installations.

The command line used to start the DCU is as follows:

```
Progress_Install_Dir\bin\prowin32 -p icfcfg.w -icfparam  
DCUSETUPTYPE=user-defined-setup-type
```

Where:

- *Progress_Install_Dir* is the Progress installation directory.
- *user-defined-setup-type* is a setup type defined in the DCU driver file.

The DCU should be started in the directory that contains the *icfcfg.w* procedure (normally *Dynamics_Install_Dir\gui\icf*). Since the DCU uses components of the Progress Dynamics framework, the *\af\app* and *\install* subdirectories should be available under DCU's start directory.

5.7 Running the DCU in batch mode

You can run the DCU in batch mode and completely unattended. This facility allows you to provide an automated application installation or upgrade procedure for end-user sites where technical knowledge might be limited. This also allows you to install/upgrade applications in environments without access to a GUI client-server license, such as for sites running WebSpeed-only clients.

For creating a batch-mode DCU procedure, Progress Dynamics provides three components:

- **User interface** — This is the standard DCU GUI that you run with additional command-line parameters. The GUI runs exactly like the full-interactive DCU, except instead of configuring Dynamics (updating the Repository), the additional parameters direct the DCU to store all of the configuration data that you enter in XML files that you specify for use with the DCU batch procedure.
- **Phase 1 configuration** — This is the first DCU batch component, consisting of a Progress 4GL procedure (`dcuphase1.p`) that uses the XML files you generate from the user interface component. This batch component updates the Repository schema and executes various fix programs.
- **Phase 2 configuration** — This is the second DCU batch component, consisting of a Progress 4GL procedure (`dcuphase2.p`). This batch component completes the same Repository update procedure that the full-interactive DCU initiates after logging into the Progress Dynamics environment as an admin user.

Thus, to create a batch-mode DCU procedure:

- 1 ♦ Run the DCU GUI to generate the batch-mode XML data files.
- 2 ♦ Create a script that executes `dcuphase1.p` and `dcuphase2.p` in sequence, and performs any other actions that you require to complete the application installation or upgrade.

The following sections describe these steps.

5.7.1 Generating the batch-mode data

The data required by the DCU batch components consists of two XML files that you generate using the DCU GUI:

- **DCU script file** — Contains all the data related to ADO paths, what patches need to be applied, what ADOs need to be loaded and any other information needed to complete the Repository update that is not site specific.
- **DCU site data file** — Contains any data that is site specific. This includes such data as the site number, the connection parameters to the databases, and other information that is specific to the end-user site.

These two files allow you to tailor your batch installations and upgrades to individual end-users of an application using a different file for each site, while maintaining a common set of upgrade data in a common file for all users.

To generate the data for use by the DCU batch components:

- 1 ♦ Create a new Windows shortcut from a copy of the standard Dynamics Configuration Utility shortcut.
- 2 ♦ Open the properties window for the new shortcut and modify the Target field by expanding the -icfparam startup parameter to include the DCUSCRIPTFILE and DCUSITEDATAFILE options using the following syntax:

SYNTAX

`DCUSCRIPTFILE=script-file[,DCUSITEDATAFILE=site-data-file]`

The parameters specify:

- *script-file* — The filename of the DCU script file, including the .xml extension.
- *site-data-file* — The filename of the DCU site data file, including the .xml extension.

If you do not specify the DCUSITEDATAFILE option, the DCU writes all of the site-specific data to the specified *script-file*.

For example, the complete -icfparam value might look like this:

```
-icfparam
DCUSETUPTYPE=ProgressSetup,DCUSCRIPTFILE=dcuscript.xml,DCUSITEDATAFILE=
site42.xml
```

- 3 ♦ Start the DCU GUI with the new shortcut, connecting to your updated deployment Repository.
- 4 ♦ Respond to the prompts as if you were running the DCU to install or upgrade this Repository at a specified end-user site.

After you choose the Process button, the DCU writes out the specified files instead of doing a Repository upgrade. These files are written to the shortcut Start in directory, unless you specify a different directory.

5.7.2 Creating a batch-mode DCU script

Write a batch command file or script that executes the two DCU batch procedures, `dcuphase1.p` and `dcuphase2.p`, in sequence. Progress Dynamics provides a sample batch command file for you to use as a model (see the [“Sample batch-mode DCU script”](#) section).

Both procedures start a complete Dynamics session in order to complete the installation or upgrade of the target Repository. They connect to the target site Repository as necessary and write details of their activity to the log file you specify in the DCU GUI. Each procedure returns a value to the calling script indicating success or failure.

Like any Dynamics session, these procedures require configuration files (-icfparam ICFCONFIG) and session types (-icfparam ICFSESSTYPE) to work. You generate the appropriate configuration files, and Progress Dynamics provides a default session type for each procedure that supports batch clients.

You must also use the -ini startup parameter, like the following for each procedure, to specify a .ini file that sets the PROPATH for the batch-mode DCU as follows:

```
[Winchar Startup]
PROPATH=.,Dynamics_Install\tty\icf,Dynamics_Install\src\icf,Dynamics_Install\src\icf\af\sup2,Dynamics_Install\tty,
Progress_Install\tty\adedict.pl,Progress_Install\tty\prodict.pl,Progress_Install\PROBUILD\EUCAPP\EUC.PL,Progress_Install\PROBUILD\EUCAPP
```

NOTE: These procedures are installed in `Dynamics_Install\src\icf`.

Starting up dcuphase1.p

The `dcuphase1.p` procedure updates the Repository schema and runs various fix programs to prepare for loading the installation or upgrade data. To start up `dcuphase1.p`, use a command line like this:

```
bpro -p dcuphase1.p -ini dcubatch.ini -pf dcuphase1.pf
```

The `dcubatch.ini` file is typically an edited copy of your `dynamics.ini` file that you use to set the `PROPATH` appropriately for the batch-mode DCU. You must use a `.pf` file (`dcuphase1.pf` in the example) to specify the `-icfparam` options for the Dynamics session. (The batch character process in Windows does not accept the `-icfparam` startup parameter specified directly on the command line.) For `dcuphase1.p`, you specify the same DCU XML files for input using the `-icfparam` options that you specified to generate these files using the DCU GUI (see the [“Generating the batch-mode data”](#) section). For example, you might specify this `-icfparam` setting in `.pf` file:

```
-icfparam DCUSCRIPTFILE=dcuscript.xml,DCUSITEDATAFILE=site42.xml
```

The DCU script file provides all of the environment settings as you specified them in the DCU GUI. If you do not specify the `DCUSITEDATAFILE` option, the procedure assumes that the necessary site data is included in the DCU script file.

Note also that `dcuphase1.p` looks for `dcuphase1.xml` as the default `ICFCONFIG` setting, which you generate as a configuration file from your central Repository, and it uses `dcuphase1` as the default `ICFSESSTYPE` setting, a session type that supports batch clients.

Starting up dcuphase2.p

Start up this procedure only if `dcuphase1.p` completes successful. This procedure makes a connection to the Repository to load the actual installation or upgrade data. To start up `dcuphase2.p`, use a command line similar to starting up a Progress Dynamics session in the AppServer environment, except you must specify a session type that supports batch clients and you must use a `.pf` file to specify the `-icfparam` settings, like this:

```
bpro -p dcuphase2.p -ini dcubatch.ini -pf dcuphase2.pf
```

Again, you use a `.ini` file (`dcubatch.ini`) file to set the `PROPATH` appropriately for the batch-mode DCU.

The dcuphase2.pf file might contain the following settings:

```
-icfparam ICFCONFIG=icfconfig.xml,ICFSESSTYPE=dcuphase2
```

Progress Dynamics provides the dcuphase2 session type for you to use for this procedure.

Sample batch-mode DCU script

This is the sample batch-mode script (*Dynamics_Install/src/icf/install/dcubatch.bat*) provided with Progress Dynamics for you to use as a guide for writing your own batch-mode DCU installation or upgrade procedure:

Sample batch-mode DCU script

(1 of 2)

```
@echo off
SET DLC=c:\apps\progress\91e
SET PATH=%DLC%;%DLC%\bin;%PATH%

:CHECKPHASE1ERRFILE
    echo Starting DCU Phase 1...
    if exist dcuphase1err.txt goto DELPHASE1ERRFILE
    goto PROCESSPHASE1

:DELPHASE1ERRFILE
    del dcuphase1err.txt

:PROCESSPHASE1
    SET DISPANNER=no
    call mbpro -b -p dcuphase1.p -ini dcubatch.ini -pf dcuphase1.pf
    if exist dcuphase1err.txt goto PHASE1FAILED
    echo DCU Upgrade Phase 1 Successful.
    goto CHECKPHASE2ERRFILE

:PHASE1FAILED
    echo *** DCU Upgrade Phase 1 Failed. Please see log file for more
information.
    type dcuphase1err.txt
    del dcuphase1err.txt
    goto CLEANUP

:CHECKPHASE2ERRFILE
    echo Starting DCU Phase 2...
    if exist dcuphase2err.txt goto DELPHASE2ERRFILE
    goto PROCESSPHASE2

:DELPHASE2ERRFILE
    del dcuphase2err.txt
```

Sample batch-mode DCU script

(2 of 2)

```
:PROCESSPHASE2
  SET DISPANNER=no
  call mbpro -b -p dcuphase2.p -ini dcubatch.ini -pf dcuphase2.pf
  if exist dcuphase2err.txt goto PHASE2FAILED
  echo DCU Upgrade Phase 2 Successful.
  goto CLEANUP

:PHASE2FAILED
  echo *** DCU Upgrade Phase 2 Failed. Please see log file for more
information.
  type dcuphase2err.txt
  del dcuphase2err.txt
  goto CLEANUP

:CLEANUP
set DISPANNER=
:END
```

The `dcubatch.ini`, `dcuphase1.pf`, and `dcuphase2.pf` files are also sample files provided with `dcubatch.bat`, which you can modify as appropriate. You must modify `dcubatch.bat` for the paths in your Windows environment and reworked to run in a Unix environment.

5.8 Dumping and loading site-specific data

When you upgrade an end-user Repository, it is often necessary for the end-user to retain data that is specific to their own site. For example, a user may add menu items to existing menus. When an application upgrade takes place, if no other action has been taken to save them, these items are lost and the user must manually re-create them after your upgrade is complete.

The Site Data Dump and Load utilities allow users to retain their own site-specific data. It does this by exporting the Repository data for site-specific tables in Progress dump format prior to running your upgrade, then by loading that data back into the Repository after the Progress Dynamics deployment datasets have been applied.

These utilities can be run in one of two ways:

- From within the Progress Dynamics Configuration Utility
- From a free-standing GUI window

5.8.1 Site Data Dump and Load utilities architecture

The Site Data Dump and Load utilities are driven by a configuration file (`dumpconfig.txt`) that is stored in `.d` text (Progress dump file) format in the first `sitedatadump` directory found in the session's `PROPATH`. This file defines a list of dump programs, load programs, data files and site numbers to support the dumping and loading of site-specific data. The `sitedatadump` directory must exist and must contain a valid `dumpconfig.txt` file for this tool to work.

Use of these utilities follows this processing sequence:

1. The dump utility (`dcusitedatadump.p`) reads the configuration file to derive a list of dump programs to run and the site numbers for which data are to be dumped.
2. The dump utility runs the specified dump programs, which export the data to `.d` files that are to be read during the later import phase.

NOTE: You can hook the dump utility into the DCU so the data is automatically dumped during the DCU upgrade process.

3. The load utility (`dcusitedataload.p`) then reads the configuration file to derive a list of load programs to run.
4. The load utility runs the specified load programs to read the data dumped by the dump utility and import it back into the Repository.

NOTE: As with the dump utility, you can hook the load utility into the DCU so the data is automatically loaded during the DCU upgrade process.

The dump and load programs called by these utilities are partly user-defined procedures. These procedures must adhere to a certain standard described in the following sections. Most of the code in these programs comes from standard include files. You must add only the small pieces of information needed to specialize the dump or load program for a particular table.

Three sets of default dump and load programs have been specifically written to support dumping and loading the contents of the following framework tables:

- `gsm_menu_structure_item`
- `gsm_toolbar_menu_structure`
- `gsm_object_menu_structure`

To run the Site Dump and Load utilities interactively, without the DCU, Progress Dynamics provides a utility program (`sitedatadmpload.w`) that displays a simple GUI for you to invoke each dump or load utility as you require.

5.8.2 Setting up the utilities

To set up the Site Data Dump and Load utilities, follow these steps:

- 1 ♦ Write the dump/export programs.
- 2 ♦ Write the load/import programs.
- 3 ♦ Create the `dumpconfig.txt` file.
- 4 ♦ Invoke the dump and load utilities as appropriate.

All of the code and example code for the utilities resides in `Dynamics_Install_Dir/src/icf/db/icf/dfd`. [Table 5–7](#) lists the files that are included in that directory (relative to the default PROPATH entry, `Dynamics_Install_Dir/src/icf`):

Table 5–7: Site Data Dump and Load installed files (1 of 2)

Path name	Description
<code>db/icf/dfd/dcusitedata.xml</code>	Sample XML file that shows how to hook these utilities into the DCU.
<code>db/icf/dfd/dcusitedatadump.p</code>	The DCU-callable procedure that calls the dump programs.
<code>db/icf/dfd/dcusitedataload.p</code>	The DCU-callable procedure that calls the load programs.
<code>db/icf/dfd/dumpconfig.txt</code>	A sample configuration file that can be used to dump all data in the <code>gsm_menu_structure_item</code> , <code>gsm_toolbar_menu_structure</code> , and <code>gsm_object_menu_structure</code> tables.
<code>db/icf/dfd/gsmitin.p</code>	Import program for <code>gsm_menu_structure_item</code> .
<code>db/icf/dfd/gsmitout.p</code>	Export program for <code>gsm_menu_structure_item</code> .
<code>db/icf/dfd/gsmomin.p</code>	Import program for <code>gsm_object_menu_structure</code> .
<code>db/icf/dfd/gsmomout.p</code>	Export program for <code>gsm_object_menu_structure</code> .
<code>db/icf/dfd/gsmtmin.p</code>	Import program for <code>gsm_toolbar_menu_structure</code> .
<code>db/icf/dfd/gsmtmout.p</code>	Export program for <code>gsm_toolbar_menu_structure</code> .

Table 5–7: Site Data Dump and Load installed files*(2 of 2)*

Path name	Description
db/icf/dfd/sitedatadmpload.w	User Interface screen for invoking these utilities outside of the DCU.
db/icf/dfd/sitedatahdrou.t.i	Standard header for export programs, including parameter definitions.
db/icf/dfd/sitedataprocout.i	Procedure block code for a completely standard export program.
db/icf/dfd/sitedatahdrin.i	Standard header for import programs, including parameter definitions.
db/icf/dfd/sitedataprocin.i	Procedure block code for a completely standard import program.
db/icf/dfd/siteapplyrechdr.i	Definition of the applyRecord internal procedure required for standard import programs.
db/icf/dfd/siteapplyrecftr.i	Definition of the applyRecord internal procedure footer required for standard import programs.
db/icf/dfd/ttdatafile.i	Definition of the ttDataFile temp-table.
db/icf/dfd/ttdumpfileloc.i	Definition of the ttDumpFileLocation temp-table.
db/icf/dfd/ttfixprogram.i	Definition of the ttFixProgram temp-table.

5.8.3 Writing a dump program

This is a sample dump program using default code that you can use to export the contents of the `gsm_menu_structure_item` table:

gsmitout.p sample using the default code

```
/* gsmitout.p */
{db/icf/dfd/sitedataprocout.i
  &OutputFile = "'gsmitout.d'"
  &OutputTable = "gsm_menu_structure_item"
  &ObjField = "menu_structure_item_obj"}
```

This program assumes that the data to be dumped is from a table that has an object ID field of data type decimal (ObjField include file directive). The object ID's mantissa is used to determine the site number of the data to be dumped. The OutputFile include file directive provides a character string name for the file to be dumped. All dumped data is written to the first directory named `sitedatadump` that can be found in the `PROPATH`. The OutputTable directive contains the name of the table to be exported.

Using the default `sitedataprocout.i` include file to provide all of the code is the easiest way to produce a dump program. This code automatically exports the specified data for all sites listed in the `dumpconfig.txt` file to the dump files that are later read by a load program.

If you need to add any special customizations, you can use code based on the following example to dump the data:

gsmittout.p sample using customized code

```

/* Include the parameters required for this program */
{db/icf/dfd/sitedatahdout.i}

/* Find the record in the ttDumpFileLocation table for the output file */
FIND ttDumpFileLocation
  WHERE ttDumpFileLocation.cDumpFile = {&OutputFile}
  NO-ERROR.

/* If no record exists, or the path is ?, we do not need to write this file
out. Simply return to the caller. */
IF NOT AVAILABLE(ttDumpFileLocation) OR
  ttDumpFileLocation.cDumpFilePath = ? THEN
  RETURN.

/* Open the stream that we will write the data out on and make sure we APPEND
to it. The file may have been created on a previous call to this program. The
caller will have ensured that the file was a new file before the first call */
OUTPUT TO VALUE(ttDumpFileLocation.cDumpFilePath) APPEND.

FOR EACH /* table that we need to dump */
  WHERE (Table.ObjID - TRUNCATE(Table.ObjID,0)= pdSiteNo):
/* This where clause gets us only the records for the site number that was
passed into this procedure as a parameter. */
  EXPORT Table.
END.

OUTPUT CLOSE.

```

The comments in the above code describe what needs to be in the code. Two parameters defined in `sitedatahdout.i` are passed to this procedure when it is called. The first is the site number for which data should be dumped in the mantissa of the `pdSiteNo` decimal field. The second is the table containing the dump file locations. It might be that your dump program requires more than one export file. The `dumpconfig.txt` file provides for this by allowing you to refer to the export file by the name you specify. For more information, see the [“Creating a dumpconfig.txt file”](#) section.

5.8.4 Writing a load program

The load program is slightly more complicated than the dump program because it must verify the referential integrity of the data. The following listing shows a sample load program using both default and user-supplied code to import the contents of the `gsm_menu_structure_item` table:

gsmitin.p sample with user-supplied code for referential integrity

```
/* gsmitin.p */
&SCOPED-DEFINE InputTable gsm_menu_structure_item
&SCOPED-DEFINE ObjField menu_structure_item_obj

{db/icf/dfd/sitedataprocin.i
  &InputFile = "'gsmitout.d'"
}

{db/icf/dfd/siteapplyrechdr.i}

  IF NOT CAN-FIND(FIRST gsm_menu_structure
                  WHERE gsm_menu_structure.menu_structure_obj =
tt_{&InputTable}.menu_structure_obj) THEN
    RETURN.

  IF tt_{&InputTable}.menu_item_obj <> 0.00 AND
    NOT CAN-FIND(FIRST gsm_menu_item
                  WHERE gsm_menu_item.menu_item_obj =
tt_{&InputTable}.menu_item_obj) THEN
    RETURN.

  IF tt_{&InputTable}.child_menu_structure_obj <> 0.00 AND
    NOT CAN-FIND(FIRST gsm_menu_structure
                  WHERE gsm_menu_structure.menu_structure_obj =
tt_{&InputTable}.child_menu_structure_obj) THEN
    RETURN.

  /* Duplicate record */
  IF CAN-FIND(FIRST gsm_menu_structure_item
              WHERE gsm_menu_structure_item.menu_structure_obj =
tt_{&InputTable}.menu_structure_obj
              AND gsm_menu_structure_item.menu_item_sequence =
tt_{&InputTable}.menu_item_sequence) THEN
    RETURN.

{db/icf/dfd/siteapplyrecftr.i}
```

The InputTable compiler directive contains the name of the table for which data is to be imported. The ObjField compiler directive specifies the field that is the object ID field for the table. The InputFile include file directive specifies the name of the file to be used as the source of the data to be imported.

This load program takes the ttDumpFileLocation table as an input parameter (`sitedatahdrin.i` from within `sitedataprocin.i`), and the above default code reads that data into a temp-table that is constructed "LIKE" the table being imported (`sitedataprocin.i`). The temp-table is named `tt_Table` where *Table* is the name of the table that is being imported.

Once the code in `sitedataprocin.i` has imported the code into the temp-table, a buffer to the temp-table is passed to an internal procedure called `applyRecord`. This is the only piece of code that needs to be customized. The header for this internal procedure is contained in `siteapplyrechdr.i`. By default, this code checks to see if a record exists with the object ID of the current record. If it does, the code assumes that the data has been supplied by the ADOs and that it should not be changed.

The footer for the `applyRecord` procedure is contained in `siteapplyrecftr.i` and includes the transaction that commits the data in the temp-table to the database.

You (or your end-user) supply all of the code specified between the `siteapplyrechdr.i` and `siteapplyrecftr.i` include file references. This code is responsible for verifying the referential integrity of the data being imported. If any of the integrity checks fail, control returns to the caller. This results in the main block importing the next record from the `.d` file and verifying that it is correct.

You can completely customize the code for the load program, as in the following example:

gsmitin.p sample with completely customized code

```
/* sitedatahdrin contains the parameter definitions for the import program. */
{db/icf/dfd/sitedatahdrin.i}

DEFINE VARIABLE iLineNo      AS INTEGER      NO-UNDO.
DEFINE TEMP-TABLE tt_{&InputTable} LIKE {&InputTable}.
/* Find the import file name */
FIND ttDumpFileLocation
  WHERE ttDumpFileLocation.cDumpFile = {&InputFile}
  NO-ERROR.

/* If its not available, ignore it and let the caller call the next import
program */
IF NOT AVAILABLE(ttDumpFileLocation) OR
  ttDumpFileLocation.cDumpFilePath = ? THEN
  RETURN.

/* Look for the dump file on disk. */
IF SEARCH(ttDumpFileLocation.cDumpFilePath) = ? THEN
DO:
  PUBLISH "DCU_WriteLog":U ("WARNING: Import file not found: " +
    ttDumpFileLocation.cDumpFilePath).

  RETURN.
END.
ELSE
DO:
  PUBLISH "DCU_WriteLog":U ("Reading data from: " +
    ttDumpFileLocation.cDumpFilePath).
END.

/* Open the input stream for the dump file */
INPUT FROM VALUE(ttDumpFileLocation.cDumpFilePath).

/* Loop until we have read the whole dump file. */
REPEAT:
  /* Import the data from the .d file */
  CREATE tt_{&InputTable}.
  IMPORT tt_{&InputTable}.
  iLineNo = iLineNo + 1.

  /* Do whatever referential integrity checks here */
  DELETE tt_{&InputTable}.
END.

INPUT CLOSE.
PUBLISH "DCU_WriteLog":U ("Finished reading data from: " +
  ttDumpFileLocation.cDumpFilePath).
```

5.8.5 Creating a dumpconfig.txt file

The dumpconfig.txt file provides the processing directions for the Site Data Dump and Load utilities. This file must exist on the PROPATH as sitedatadump/dumpconfig.txt for these utilities to work.

This file is a dump (.d) formatted file that can be imported using the Progress IMPORT statement to load a table of instructions to follow. The file can contain multiple lines that specify all of the fields listed in [Table 5–8](#).

Table 5–8: Field options for creating the dumpconfig.txt file

Field No.	Name	Data type	Description
1	Record Type	CHARACTER	<p>The type of record that is being imported. This can be one of:</p> <ul style="list-style-type: none"> • "I" for the load program file to run. • "D" for the dump file to read or write. • "O" for the dump program file to run. • "S" for site number.
2	Group Number	INTEGER	<p>If Record Type is one of "I", "D", or "O", it is used to group the records for a dump and load set together.</p> <p>If Record Type is "S" it contains the specific site number of the data to dump and load.</p>
3	Filename	CHARACTER	<p>Depending on the Record Type, contains a:</p> <ul style="list-style-type: none"> • Relative path name of the dump or load program in the PROPATH. • Dump filename. • A special token or string for the site number.

The following listing shows the default `dumpconfig.txt` file that ships with the utility software:

dumpconfig.txt

```
"O" 1 "db/icf/dfd/gsmhout.p"
"I" 1 "db/icf/dfd/gsmhin.p"
"D" 1 "gsmhout.d"
"O" 2 "db/icf/dfd/gsmhout.p"
"I" 2 "db/icf/dfd/gsmhin.p"
"D" 2 "gsmhout.d"
"O" 3 "db/icf/dfd/gsmhout.p"
"I" 3 "db/icf/dfd/gsmhin.p"
"D" 3 "gsmhout.d"
"S" 0 "&S"
```

The first three lines of this file contain the default data needed to dump and load the `gsm_menu_structure_item` table.

Line 1 contains information on the dump program. The Record Type is "O" for output, indicating that the line contains information about the dump program itself. The Group Number is 1, and note that all the first three lines have the same Group Number. This number relates the data file with its corresponding dump and load programs. In other words, a dump program, its corresponding load program, and data files must all have the same Group Number. For the dump program, the File Name field contains the program filename, including its relative path, used to dump the data out.

Line 2 contains information on the load program. The Record Type is "I" for input, indicating that the line contains information about the load program. The Group Number is the same as the group number for the corresponding dump program, and the File Name contains the load program filename, including its relative path.

Line 3 contains information about the data file that is the dump file target. The Record Type is "D" for dump file. The Group Number is the same as for the corresponding dump and load program lines, and the File Name is the name of a file that will be written to and read from the first `sitedatadump` directory in the `PROPATH`. Note that you can associate more than one dump file with a dump and load program group. The File Name field is used as the key to retrieve the file information. During the process of reading the `dumpconfig.txt` file, the utilities determine the full path to this file and verify that it is accessible, storing the result in the `ttDumpFileLocation.cDumpFilePath` field. Because this is a verified path name, all `INPUT FROM` and `OUTPUT TO` statements should use the value of this field to determine the correct file to read from or write to.

The next six lines of this file contain dump and load information for the `gsm_object_menu_structure` and `gsm_toolbar_menu_structure` tables.

The final line of the file contains a site number record. The Record Type is set to "S" for site number. The Group Number is 0 and the File Name is "&S". This line is a special case. If the Group Number is 0 and the File Name is &S, the dump utility determines the site number of the currently connected Dynamics Repository and uses that number as the site number for data to be dumped.

If you want to specify a specific site number for which to dump and load the data, instead of using 0, set the Group Number to the value of the site number and set the File Name to the empty string (""). The following line causes the dump program to dump all data for site 95:

```
"S" 95 ""
```

The default `dumpconfig.txt` file provided in the `icf/db/icf/dfd` directory directs these utilities to dump and load data for the `gsm_menu_structure_item`, `gsm_toolbar_menu_structure` and `gsm_object_menu_structure` tables that were created in a session connected to the current Repository. To make these utilities work, copy the `dumpconfig.txt` file to a directory named `sitedatadump` somewhere in the session `PROPATH` and run the utilities as appropriate.

5.8.6 Using the GUI

To invoke the Site Data Dump and Load utilities using the standalone GUI, run the program, `Dynamics_Install_Dir/src/icf/db/icf/dfd/sitedatadmpload.w`. This program opens a window like the one shown in [Figure 5–3](#).

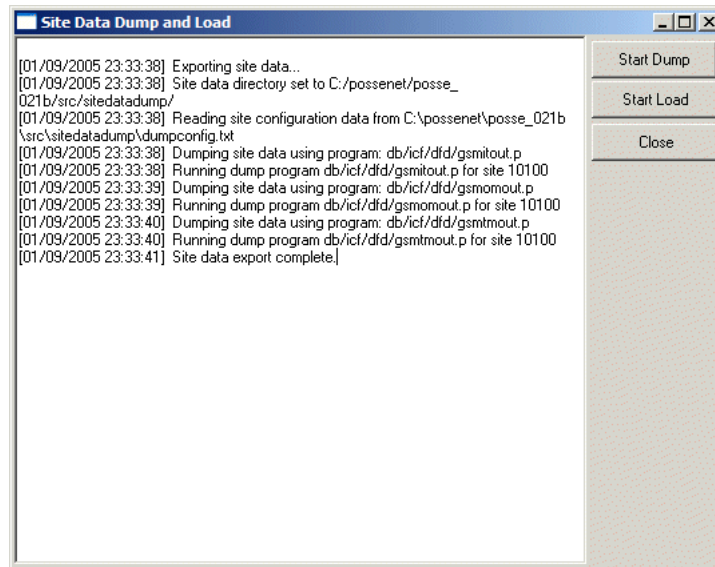


Figure 5–3: Standalone GUI for Site Data Dump and Load utilities

Choosing the Start Dump button invokes the Site Data Dump utility, which exports the data and displays information on the export process in the large editor on the screen. After dumping the data (indicated by the message, “Site data export complete.”), load any ADOs required to update the Repository. After this ADO load is complete, you can invoke the load utility.

Choosing the Start Load button invokes the Site Data Load utility, which imports the data that you just exported and displays information on the import process in the large editor on the screen.

5.8.7 Calling the Utility from the DCU

To invoke the Site Data Dump and Load utilities from the DCU, you need to add the following data to the last patch file used by the DCU for an upgrade:

```
<PatchStage Stage="PreADOLoad">
  <Program>
    <FileType>p</FileType>
    <FileName>db/icf/dfd/dcusitedatadump.p</FileName>
    <Description>Dumping site specific data</Description>
    <Rerunnable>yes</Rerunnable>
    <NewDB>no</NewDB>
    <ExistingDB>yes</ExistingDB>
    <UpdateMandatory>yes</UpdateMandatory>
  </Program>
</PatchStage>
<PatchStage Stage="PostADOLoad">
  <Program>
    <FileType>p</FileType>
    <FileName>db/icf/dfd/dcusitedataload.p</FileName>
    <Description>Loading site specific data</Description>
    <Rerunnable>yes</Rerunnable>
    <NewDB>no</NewDB>
    <ExistingDB>yes</ExistingDB>
    <UpdateMandatory>yes</UpdateMandatory>
  </Program>
</PatchStage>
```

The first PatchStage node adds the dump utility to the PreADOLoad section of the patch file. This invokes the dump utility to export the site data after the schema updates have been applied to the Repository, but before the ADOs are actually loaded.

The second PatchStage node adds the load utility to the PostADOLoad section of the patch file. This invokes the load utility to import the site data that was dumped before the ADOs were applied to the Repository, but after the ADOs are actually loaded.

Using Progress DataServers with Progress Dynamics

This appendix describes some of the factors you should consider when using a Progress DataServer in the Progress Dynamics environment:

- [Overview](#)
- [Connecting the database](#)
- [Programming considerations](#)
- [Other useful documentation](#)

A.1 Overview

When you access a non-Progress database with a Progress DataServer, there are unique considerations in the Progress Dynamics environment. Before using a DataServer with Progress Dynamics, you should be familiar both with the Progress Dynamics framework and with integrating a DataServer in a traditional Progress application. The [“Other useful documentation”](#) section of this appendix lists several documents that cover the use of Progress DataServers in detail.

A.1.1 Progress Dynamics support for DataServers

In Version 2 of Progress Dynamics, you can access non-Progress databases that are supported by the Progress DataServers for ORACLE and Microsoft SQL Server.

A.1.2 When to use DataServers with Progress Dynamics

The database of choice for the Progress Dynamics framework is the Progress database. The framework and database are highly integrated with each other. This allows you to take advantage of unique features that are not available with non-Progress database solutions. Typically, using a Progress database gives the best performance for any Progress-based application.

In some cases, a non-Progress database provides the best solution in a specific business situation. For example, in a business environment that already uses a non-Progress database, your Progress Dynamics application might need to access the existing database. Using a Progress DataServer, you can take advantage of the powerful features available from the Progress Dynamics framework and keep legacy applications that access the existing database in production at the same time.

A.1.3 Requirements for DataServers in Progress Dynamics

When you use a DataServer in a Progress Dynamics application, you must plan for it in your application’s design. In general, Progress code that runs against a DataServer can also run against a Progress database. Exceptions are code that makes calls unique to the non-Progress database, such as to access stored procedures.

The Progress Dynamics framework requires a Repository database. While the data that your application uses can be in a non-Progress database accessed by a Progress DataServer, the Progress Dynamics Repository must be a Progress database.

A.2 Connecting the database

A major difference between Progress Dynamics applications that connect to Progress databases and those that use DataServers is how the database connection is entered. With the DataServer, setting the connection includes making connections for both the schema holder database and the non-Progress database. You create a connection to the schema holder database as normal using the Progress Dynamics tools. For that connection, the area for additional parameters includes the connection information for the non-Progress database. You do not set up a separate database connection for the non-Progress database. The “[Schema holder database](#)” section gives details on setting the connection parameters.

You can set up the connection to the schema holder database by creating a new Physical Service using the Progress Dynamics Administration tool. The connection to the schema holder database is saved in the `icfconfig.xml` file generated by the Progress Dynamics session tools.

You can also handle the connection programmatically. You can use this method if you do not need the non-Progress database connected at application startup.

A.2.1 Configuring Progress DataServer for MS SQL Server

This section provides an example of setting up the connection with the Progress DataServer for MS SQL Server. You must supply the following connection parameters when you set up the connection to the schema holder database:

```
-db <Progress schema holder> -H <host> -S <service> -N TCP  
-db <ODBC DSN name> -ld <SQL Server db> -dt MSS -U <user name> -P <password>  
-H <broker host> -S <broker service> -Dsrv TXN_ISOLATION,1
```

You first supply the Physical Database Name, Host Name, Service Name, and Network Type startup parameters for the schema holder database. The remaining parameters are used to start the DataServer connection to the non-Progress database.

The following example shows how this information is stored in the `icfconfig.xml` configuration file:

```
<service>
<cServiceType>Database</cServiceType>
<cServiceName>MSS</cServiceName>
<cPhysicalService>MSSn</cPhysicalService>
<cConnectParams>-db sh_mydb -N TCP -H localhost -S demo_svc -db mss_dsn_db
    -ld mss_db -dt MSS -U icftest -P icftest -Dsrv TXN_ISOLATION,1
</cConnectParams>
<lDefaultService>no</lDefaultService>
<lCanRunLocal>yes</lCanRunLocal>
<iStartOrder>0</iStartOrder>
</service>
```

NOTES:

- For Microsoft SQL Server, the database type is always MSS.
- The DSN name is the configured DSN that you use to access the MS SQL Server database with ODBC. It does not need to be the same name as the SQL Server database. When the DSN is configured, the name of the physical MS SQL Server database is specified.
- The logical name for the database is the name that you use to access the MS SQL Server database with the 4GL. If a logical name is not specified, the default logical name is used. This is the name of the DSN for MS SQL Server.
- The isolation level is set to one using “-Dsrv TXN_ISOLATION,1”. This provides Progress-like locking using MS SQL Server’s “read uncommitted” transaction isolation level.
- The -ld and -dt parameters are optional.
- If the schema holder is on the same machine as the client, then the -H and -S parameters for the schema holder are optional.
- Because this example is for a system using the Personal version of the MS SQL Server DataServer, the -H and -S parameters are not specified for the database broker. If your system uses the Enterprise version, the -H and -S parameters are needed if the database broker is located on a machine other than the client. The [“Configuring Progress DataServer for Oracle”](#) section provides an example of using these parameters.

A.2.2 Configuring Progress DataServer for Oracle

This section provides an example of setting up the connection with the Progress® Oracle® DataServer. You must supply the following connection parameters when you set up the connection to the schema holder database:

```
-db <Progress schema holder> -H <host> -S <service> -N TCP
-db <Oracle db name> -ld <same Oracle name> -dt ORACLE
-U "name/password@Oracle_instance" -H <broker host> -S <broker service>
```

You first supply the Physical Database Name, Host Name, Service Name, and Network Type startup parameters for the schema holder database. The remaining parameters are used to start the DataServer connection to the non-Progress database.

The following example shows how this information is stored in the `icfconfig.xml` configuration file:

```
- <service>
  <cServiceType>Database</cServiceType>
  <cServiceName>ORA</cServiceName>
  <cPhysicalService>ORAn</cPhysicalService>
  <cConnectParams>-db sh_oratest -N TCP -H client_system -S demo_svc1
    -db ora_db -ld ora_db -dt ORACLE -U "icftest/icftest@test821"
    -H server_system -S demo_svc2</cConnectParams>
  <!--DefaultService-->no</DefaultService>
  <!--CanRunLocal-->yes</CanRunLocal>
  <!--iStartOrder-->0</iStartOrder>
</service>
```

NOTES:

- For Oracle, the database type is always ORA.
- For the Oracle connection, you must use double quotes around the username and password. If double quotes are not used, Progress Dynamics might incorrectly parse the connection string.
- The `-ld` and `-dt` parameters are optional.
- If the schema holder is on the same machine as the client, then the `-H` and `-S` parameters for the schema holder are optional.
- In this example, the second set of `-H` and `-S` parameters specify the location of the database broker. If the database broker is located on a different machine than where the client is running, you must specify the `-H` and `-S` parameters.

A.2.3 Schema holder database

A schema holder database is a Progress database that holds information about the schema of the non-Progress database, such as the names of tables and the offsets of where particular fields or columns begin. The DataServer only accesses the schema holder the first time a table on the non-Progress database is accessed. In a run-time environment, new and modified data are not stored in the schema holder database. Instead, the new and modified data is stored in the non-Progress database. Therefore, the schema holder database usually is connected as read-only. Connecting read-only allows you to access the schema holder database slightly faster and improves performance.

CAUTION: In a Progress Dynamics environment, Progress does not support combining into a single Progress database the tables and data for the schema holder and the tables and data for the Progress Dynamics Repository. The main drawback to that implementation is the loss of both environments if one becomes corrupted. For example, if the schema holder data and the Repository data are stored in one database and there is a problem, such as corruption, then both the schema holder and the Repository must be restored.

A.2.4 Progress Dynamics configurations with DataServers

Configuring a Progress Dynamics environment to use a DataServer is similar to configuring Progress Dynamics when using a Progress database. The only difference is in the database connections. For the DataServer connection, there is a connection to both the schema holder database and the non-Progress database.

For more information on possible configurations, see the “Utilizing Progress DataServers with Progress Dynamics” white paper described in the [“Other useful documentation”](#) section.

A.3 Programming considerations

There are special considerations you need to take into account when you develop a Progress Dynamics application that uses a DataServer. This section describes these considerations.

A.3.1 Indexes

Progress DataServers allow non-Progress database views to be used in a Progress environment. Generally, a view does not need an index when accessed with the 4GL. A view is treated as a read-only table.

Progress Dynamics requires that each table have an index. One reason for this requirement is that Progress Dynamics makes extensive use of ROWID. To use a view in Progress Dynamics, you must add an index to the schema holder for the view. If an index is not added to the schema holder, the view can still be used as a table. However, the data from the view is not displayed.

You can also use a Progress DataServer to access a non-Progress database table that does not have an index defined. For the same reason, you would have similar problems accessing this data. To see the data in such a table using Progress Dynamics, you must add an index.

A.3.2 Two-phase commit

Two-phase commit is not supported when using the Progress DataServer to access non-Progress databases. Without two-phase commit in some cases, you cannot guarantee that either all of the database updates or none of the database updates are written at the end of the transaction. This happens when there are updates to several databases within one transaction, and when one or more of those databases are accessed through a DataServer.

The Progress Dynamics environment always has multiple databases connected. This makes the lack of two-phase commit of special interest when you use DataServers in Progress Dynamics. However, the design of the Progress Dynamics framework limits where a single transaction might update more than one database. The main areas of concern are for the auditing and comment support.

The built-in auditing and commenting features of Progress Dynamics are optional. If you use these features and a Progress DataServer, then a failure during a transaction could cause an inconsistent state in the databases. Although the window for this type of failure is small, you might want to avoid using these optional features if you cannot afford a failure of this type.

A.4 Other useful documentation

There are few differences between developing a Progress Dynamics application that only uses Progress databases and one that uses a DataServer to access non-Progress databases. Remember that the 4GL restrictions and performance considerations documented in the Progress DataServer guides and white papers also apply in a Progress Dynamics environment. All those development requirements apply to Progress Dynamics when accessing a DataServer. Success in this environment depends on fully understanding each of the components involved in your application environment.

Progress Documentation

The guides listed below describe how to use the DataServers to access non-Progress databases. They provide instructions for building the DataServer modules, a discussion of programming considerations, and a tutorial:

- *Progress DataServer For ORACLE Guide*
- *Progress DataServer For Microsoft SQL Server Guide*

White papers

Progress white papers can be found in the PSDN library at the following URL:

http://psdn.progress.com/library/white_papers.htm

Select the DataServers tab to see the available white papers on Progress DataServers. You can find the following white papers either in the Progress Dynamics collection or the DataServers collection:

- *Utilizing Progress DataServers with Progress Dynamics*

This document covers the unique considerations for using DataServers in a Progress Dynamics environment.

- *DataServer Best Practices*

This document is a guide to help project managers and application developers understand and estimate the work involved in adopting the Progress DataServer technology. This document contains advice for the ORACLE, MS SQL Server, and ODBC Progress DataServer products.

- *Building High Performance Applications with the Progress ORACLE DataServer*

This paper helps you create high performance applications with the Progress 4GL and WebSpeed toolsets to integrate with an Oracle® relational database management system (RDBMS). It explains how the Progress ORACLE DataServer supports the development and delivery of high performance applications on an Oracle database.

- *Configuration and Coding Techniques for the Progress DataServer for Microsoft SQL Server*

Achieving high performance from your DataServer for Microsoft SQL Server applications involves optimizing the system's configuration and using the most effective querying and transaction management techniques. This white paper explains how to tune every piece of the DataServer architecture and how to write data-access logic that allows you to take best advantage of Progress and SQL Server resources.

Progress Dynamics Session Properties

This appendix describes the Progress Dynamics™ session properties. For information about defining properties for a session, see [Chapter 2, “Defining and Managing Sessions.”](#)

Table B–1: Progress Dynamics session properties

(1 of 5)

Name	Description
AB_compile_into_directory	AppBuilder object code root directory.
AB_source_code_directory	AppBuilder source code root directory.
allow_anonymous_login	Determines whether anonymous logins are allowed for WebSpeed.
anonymous_user_name	Used if allow_anonymous_login is on.
anonymous_user_password	Password to apply to anonymous_user_name.
auto_dump_entity_cache	Indicates to Dynamics if it is to dump the entity cache to disk at session shutdown. The default is YES.
bound_icfdb	Specifies whether a session type is bound or unbound. Set this property on sessions that are connected before Progress Dynamics starts to ensure correct generation of icfconfig.xml.
ClassIgnoreContainedInstances	Deprecated.

Table B–1: Progress Dynamics session properties*(2 of 5)*

Name	Description
client_cache_directory	An absolute path where the client cache is generated and accessed.
configuration_source	Defines how a session type is treated when it is initialized.
CustomizationTypePriority	Comma-separated list of customization codes in order of priority.
DCUSETUPTYPE	Derived from DCUSETUPTYPE -icfparam (used only by the DCU).
display_login_screen	Indicates whether the login screen should be displayed.
field_cache_options	Specifies what DynCombo and DynLookup Progress SmartDataField™ classes have their data cached on the client (all or a comma-separated list). If you do not define this property or specify it with a blank value, no SmartDataFields have their data cached.
ICFCONFIG	Derived from -icfparam ICFCONFIG.
ICFPATH	Derived from -icfparam ICFPATH.
ICFPATH1	Derived from -icfparam ICFPATH.
ICFPATHn	Derived from -icfparam ICFPATH.
ICFSESSTYPE	Derived from -icfparam ICFSESSTYPE.
IDEPalette	Comma-separated list of AppBuilder palette objects. The default is PaletteDynamics.
IDETemplate	Comma-separated list of master template objects to load for the AppBuilder. The default list is: templateContainer, templateSmartObject, templateProcedure, templateWebIObjct.
image_path	Directory path of images for Web browser applications.

Table B–1: Progress Dynamics session properties*(3 of 5)*

Name	Description
keep_old_field_api	Specifies if Dynamics uses the performance-optimized API for dynamic combos and lookups or uses the API that is available in versions prior to 2.1B, without these optimizations.
login_procedure	Login procedure to be used at session startup.
MaxHiddenContainers	Specifies the maximum number of dynamic frames that can be kept alive (hidden) for a single dynamic TreeView instance after being closed by the user. The default is 10.
menu_images	Display images on menu (treeview) if set to “enabled”.
OG_ValidateFrom	Used by the Object Generator to determine validation logic within DLCProc.
physical_session_list	Automatically inserted into ICFCONFIG from session type.
print_preview_preference	The format for the data exported by Print Preview, including XML (default), HTML, and Crystal.
print_preview_stylesheet	An XML (.xsl) or HTML (.css) style sheet to apply to data exported by Print Preview.
root_directory	Root directory.
run_local	Specifies to run the session locally.
session_appl_alert_boxes	SESSION:APPL-ALERT-BOXES setting.
session_context_help_file	SESSION:CONTEXT-HELP-FILE setting.
session_data_entry_return	SESSION:DATA-ENTRY-RETURN setting.
session_date_format	SESSION:DATE-FORMAT setting.
session_debug_alert	SESSION:DEBUG-ALERT setting.
session_immediate_display	SESSION:IMMEDIATE-DISPLAY setting.
session_multitasking_interva	SESSION:MULTITASKING-INTERVAL setting.

Table B–1: Progress Dynamics session properties*(4 of 5)*

Name	Description
session_numeric_format	SESSION:NUMERIC-FORMAT and SESSION:SET-NUMERIC-FORMAT setting.
session_propath	PROPATH setting.
session_suppress_warnings	SESSION:SUPPRESS-WARNINGS setting.
session_system_alert_boxes	SESSION:SYSTEM-ALERT-BOXES setting.
session_time_format	Time format used as a format mask.
session_time_source	SESSION:TIME-SOURCE setting.
session_tooltips	Session Tooltips.
session_v6display	SESSION:V6DISPLAY setting.
session_year_offset	Session year offset attribute.
setup_type	Name of a custom session type for setting up an application with the DCU.
setup_type_file	Relative path name of the custom DCU driver file used to run a custom DCU session type.
StartupCacheClasses	Comma-separated list of the names of classes that are to be cached when the session starts.
StartupCacheMenusForObjects	A comma-delimited list of logical names for Repository containers whose menus are to be pre-cached at session startup.
StartupCacheToolbars	A comma-delimited list of logical names for toolbars to pre-cache at session startup.
startup_procedure	Startup Procedure.
startup_procedure10	Startup Procedure 10.
startup_procedure20	Startup Procedure 20.
UseThinRendering	Specifies if Dynamics uses a defined thin rendering procedure to render a given object at run time.

Table B–1: Progress Dynamics session properties*(5 of 5)*

Name	Description
valid_os_list	Automatically inserted into ICFCONFIG from session type.
window_title	Title to use for the DCU window when running with a custom session type for application setup.
_debug_tools_on	Switches on special tools in the framework for diagnostics.
_framework_code_directory	Framework directory containing code that is currently being run.
_framework_directory	Directory from which the framework is running.
_framework_drive_letter	Drive letter from which the framework is running.
_framework_gui_directory	Framework directory containing GUI code.
_framework_root_directory	Framework directory excluding /src, /gui, or /tty.
_framework_source_directory	Framework directory containing source code.
_framework_tty_directory	Framework directory containing ChUI code.
_startup_proc	Name of procedure used to start Dynamics session.
_start_in_directory	Start in directory.

Inside the Progress Dynamics Configuration File

When you install Progress Dynamics, it includes a default configuration file (`icfconfig.xml`) located in the *Dynamics_Install_Path\src\icf* directory.

You should never manually edit the configuration file. It is recommended that you only use the Dynamics Administration window tools to generate and edit it. If you do manually edit the file, you could introduce errors and cause problems when you upgrade from one release of Progress Dynamics to the next.

That being said, this appendix describes the contents of the default configuration file to help you gain an understanding of what this XML file does.

NOTE: After you start Dynamics for the first time, you should create your own session types that work with your application databases. You should then generate a new configuration file that is saved into your working directory. You can add new session services and types and then regenerate this file to use with your own applications. For more information on generating your own configuration file, see [Chapter 2, “Defining and Managing Sessions.”](#)

C.1 Session nodes

The `icfconfig.xml` file is managed by the Configuration File Manager. It is structured in a specific way. The configuration file consists of a single sessions node that contains a separate session node for each different session type. The session node will have a `SessionType` attribute set that maps to the session type specified by the `ICFSESSTYPE` attribute for the `-icfparam` startup parameter.

Thus, an `icfconfig.xml` file is generated with a session node for each of the `SessionTypes` supported for a session. The default `icfconfig.xml` file contains entries for seven distinct session types.

Each session node has three subordinate nodes:

- [Properties node](#)
- [Services node](#)
- [Managers node](#)

C.1.1 Properties node

The properties node contains a separate node for each parameter that is set and available to the entire session. Any parameters that are set inside the properties node are available to the session through a call to `getSessionParam` in the Configuration File Manager. Certain of the properties that can be set have a special usage:

- Most properties prefixed by `session_` map to attributes of the 4GL SESSION system handle. Thus, `session_date_format` specifies the `SESSION:DATE-FORMAT` attribute value. Similarly, `session_year_offset` specifies the `SESSION:YEAR-OFFSET` session attribute value.

NOTE: The `session_propath` is the one special case, which specifies the value of the session `PROPATH` environment variable.

- The `startup_procedure` property specifies non-persistent procedures that are executed in alphabetical order after the managers and service connections have been established.
- Properties prefixed by `ICFCM_` point at an entry in the managers node that contains the name of a specific Service Type Manager. For example, in the `icfconfig.xml` file, the `ICFCM_AppServer` property specifies a value of `AppServerConnectionManager`. In the services node, the `cServiceType` property indicates the type of service (`AppServer` or `Database` in this example) and when the session is started, the Connection Manager automatically starts the Connection Manager and all Service Type Managers that are specified by `ICFCM_` properties so the Connection Manager can connect to all the required

startup services. The filename of the Service Type Manager to be started is specified by a `cFileName` node in a manager node that also contains a `cManagerName` node whose value matches the `ICFCM_` property value that specifies the Service Type Manager. Thus in the default `icfconfig.xml` file, the `ICFCM_AppServer` property specifies the value `AppServerConnectionManager`, which is also specified by a `cManagerName` node in the `managers` node for the same session type. For more information, see the “[Managers node](#)” section.

- The `run_local` property specifies if the entire environment for the session type runs locally or on an AppServer. If this property is set to yes, the local 4GL client session handles all AppServer calls.
- The `physical_session_list` property provides a list of valid physical session types for a particular Progress Dynamics session (*logical session type*). Each *physical session type* identifies a physical Progress session in which the specified logical session type can run. The enumerated list of supported physical session types appears in [Table C–1](#). In other words, there might be more than one valid physical session type that can run the specified Progress Dynamics session. You can determine the current physical session type by a call to `getPhysicalSessionType` in the Configuration File Manager.

Table C–1: Physical session types

Physical session type	Description
APP	AppServer
BTC	Batch client
CUI	Character client
GUI	Graphical client
WBC	WebClient
WBS	WebSpeed Transaction Agent

- The `valid_os_list` property lists the operating systems that can run the specified Progress Dynamics session. Thus, this property can contain an enumerated list of operating systems identified by the same values as are returned by the Progress 4GL OPSYS function.

You can also add your own properties to the configuration XML file, making them accessible from inside the Progress Dynamics environment using the Configuration File Manager API. For more information, see the [Progress Dynamics Managers API Reference](#).

C.1.2 Services node

The services node of the `icfconfig.xml` file contains a subordinate service node for each of the services that are required to be connected for the session to work. Note that only services that are required to gain access to the Repository need to be specified. Once the session has made its connection to the Repository through the Connection Manager, all further information can be derived from the Repository instead of from the `icfconfig.xml` file.

Prior to making the connections to the services, the Configuration File Manager starts the Connection and Service Type managers that are required to make all of the connections. To begin the services connection process, the Configuration File Manager populates a services temp-table (`ttService`) with data contained in the services node. The entire services temp-table is passed to the Connection Manager so that it can establish the appropriate connections. For more information on how the Connection Manager establishes connections, see the [Progress Dynamics Managers API Reference](#).

Each service node contains the following nodes of its own:

- **cServiceType** — This field contains the type of service that is to be connected, defined inside the Progress Dynamics Repository. For each `cServiceType`, there must be an `ICFCM_` property that points to the appropriate manager name. This field maps to `gsc_service_type.service_type_code`.
- **cServiceName** — This field contains the logical name of the service to be connected. This field maps to `gsc_logical_service.logical_service_code`.
- **cPhysicalService** — This field contains the physical name of the service that is being used from the Repository. This field maps to `gsm_physical_service.physical_service_code`.
- **cConnectParams** — This field contains the connection parameters of the service that is being used. This field maps to `gsm_physical_service.connection_parameters`.
- **IDefaultService** — This field contains a logical variable indicating whether or not this is the default service for this service type. This field is derived from `gsc_service_type.default_logical_service_obj`.
- **ICanRunLocal** — This field contains a logical variable indicating whether or not this service can be connected to the local session. This field maps to `gsc_logical_service.can_run_locally`.

C.1.3 Managers node

The managers node of the `icfconfig.xml` file contains a subordinate manager node for each of the individual managers that need to be started.

A manager is simply a persistent procedure that will be run persistently by the Configuration File Manager. The object code for the manager needs to be accessible via the `PROPATH` of the session that is running this code.

Each manager node contains up to three nodes of its own:

- **cManagerName** — The name of the manager, derived from one of two places: `gsc_manager_type.manager_type_code` or a concatenation of `gsc_service_type.service_type_code` and the word `ConnectionManager`. If the manager is a Service Type Manager, the latter mechanism is used to derive the manager name; otherwise the former method is used.
- **cFileName** — The name of the procedure to be instantiated as a manager. The value of this field is derived from the concatenation of `gsc_object.object_path`, a forward slash, and `gsc_object.object_filename`.
- **cHandleName** — A code that indicates a static variable into which the handle of the manager should be stored. Valid values for this field are shown in [Table C–2](#).

Table C–2: Manager static handle codes

(1 of 2)

Handle code	Handle definition
NON	No StaticHandle Available. (default)
AS	<code>gshAstraAppServer</code> — The Default AppServer.
SM	<code>gshSessionManager</code> — Progress Dynamics Session manager.
SEM	<code>gshSecurityManager</code> — Progress Dynamics Security manager.
PM	<code>gshProfileManager</code> — Progress Dynamics Profile manager.
RM	<code>gshRepositoryManager</code> — Progress Dynamics Repository manager.
TM	<code>gshTranslationManager</code> — Progress Dynamics Localization manager.
WM	<code>gshWebManager</code> — Progress Dynamics Web manager.

Table C–2: Manager static handle codes *(2 of 2)*

Handle code	Handle definition
GM	gshGenManager — Progress Dynamics General Manager.
FM	gshFinManager — Reserved for backward compatibility.
AM	gshAgnManager — Reserved for backward compatibility.
AU	appSrvUtils — Reserved for backward compatibility.

Deployment Notes

This appendix describes the following deployment tasks for Progress Dynamics applications:

- [Deploying multi-transaction sequences](#)
- [Generating static-4GL equivalents of dynamic objects](#)
- [Deploying Repository objects between Progress Dynamics versions](#)

For more information on the general tools and techniques available for Dynamics application deployment information, see the deployment white paper posted on the following PSDN Web site:

http://psdn.progress.com

D.1 Deploying multi-transaction sequences

When you deploy multi-transaction sequences, you can overwrite the current value of a sequence. To avoid this problem, data versioning and re-use of deleted keys is turned off, by default, on the sequence table.

As a result, when you add new sequences to the Repository and use the dataset deployment tools, the new sequences are not automatically deployed. Therefore, in order to deploy new sequences, you must deploy the sequence table by explicitly exporting the GSCSQ dataset.

D.2 Generating static-4GL equivalents of dynamic objects

The main concept of Progress Dynamics is the use of a Repository to store abstract definitions for executable objects in an application. This abstraction allows the physical implementation of an object to change without requiring changes to and recompilation of the executable objects themselves. Objects are created and rendered using rendering procedures at runtime. These rendering procedures retrieve object definitions from the Repository in the form of data and then manipulate that data in order to create the objects and whatever other objects they might specify.

While this dynamic approach to representing and executing the objects for an application has a number of advantages, there are two primary disadvantages when used for an n-tier distributed application:

1. The application (object) definition must be transported across the network. In order for an object to be rendered, the client rendering program must retrieve the object's definition from the Repository by making an AppServer request that returns the object definition to the client. There is always some expense incurred in the actual retrieval of the object from the Repository, but the main performance bottleneck occurs in moving the data from the AppServer to the client.

Once on the client, the definition can be cached for future use. However, the client must always get the definition from the AppServer when it first instantiates the object.

2. Even though an object definition can be cached on the client, the rendering program must always transform this definition at run time in order to render a new instance of the object.

Progress Dynamics allows you to avoid these bottlenecks by generating static-4GL objects from the Repository definitions of dynamic objects.

D.2.1 Mechanisms for static object generation

Progress Dynamics allows you to generate a complete physical representation in the 4GL of an abstract object defined in the Repository. You can then compile and deploy the object in an executable form on the client that does not require Repository access in order to render it.

Generating static equivalents of dynamic objects retains your ability to quickly design and prototype an application using the Dynamics Repository and application accelerators. At the same time, it eliminates the performance bottlenecks inherent with rendering these objects from the Repository at run time.

Understanding the two static object generation mechanisms

Dynamics provides two ways to generate static objects from Repository definitions:

- **Saving dynamic objects as static objects from the AppBuilder main window** — This *Dynamics save as static object* mechanism allows you to save a static equivalent of a dynamic Progress SmartDataObject or Progress SmartDataViewer™. (No other dynamic objects can be saved this way.) When you have one of these dynamic objects open in the AppBuilder, the Save Dynamic Object As Static menu item appears on the AppBuilder File menu. You can choose this menu item to save the open dynamic SmartDataObject or SmartDataViewer as a static object. This type of static object works exactly like its dynamic equivalent, including the ADM processing and properties management, except that all of the properties required to render the object at run time are specified in the 4GL and therefore require no Repository access to obtain them. These *save as static objects* become completely independent of the original dynamic objects from which they are generated.
- **Invoking the 4GL Generator tool from the Administration main window** — This *Dynamics 4GL Generator* mechanism allows you to generate static equivalents of any selection of dynamic objects, except for SmartBusinessObjects and Dynamics Web objects. You can make the selection of one or more objects to generate and generate them by choosing the 4GL Generator menu item in the Deployment menu of the Administration main window. This type of static object works differently from its dynamic or static ADM equivalent. For example, while the object specifies the properties required to render it in the 4GL, it manages these properties differently from the ADM property management generated for save as static objects, and processes the object in a manner designed to maximize performance. These *generated-4GL static objects* are essentially static overrides of the original dynamic objects from which they are generated, and they effectively share the same registration status.

Both mechanisms eliminate the need to access the Repository for object rendering information. However, both types of static objects still depend on some access to the Repository in order to participate in the Dynamics security model. Also, while it is possible to modify the 4GL for static objects, you generally modify the generated-4GL equivalents through the original dynamic object using Dynamics tools rather than editing the generated 4GL directly. No changes that you make directly to the 4GL for either type of static object can be automatically reflected back in the Repository.

Choosing Dynamics save as static objects or generated-4GL static objects

The only two dynamic objects you can generate using both mechanisms are the SmartDataObject and SmartDataViewer. So, you only need to choose a mechanism for generating static objects for these two classes of dynamic objects. Because they work differently, your choice of what mechanism to use depends on your working priorities and requirements.

The save as static object mechanism produces an ADM-compatible 4GL procedure that you can edit in the AppBuilder using the properties and section editors, exactly like any static Dynamics object that you create using the AppBuilder. Because it is an ADM object, it can integrate smoothly with any Dynamics container, static or dynamic. This also means, as a static object, that it performs with the same performance characteristics as any static ADM object of its class. The performance of save as static objects might be better than their dynamic object equivalents, but their primary function is to change the development model of an object from dynamic 4GL to static 4GL. For more information on the function of Dynamics save as static objects, see the *Progress Dynamics Developer's Guide*.

The 4GL Generator mechanism produces a 4GL procedure with no reference to the ADM. You can only edit the 4GL for this type of object in the procedure editor or other text editor.

CAUTION: Progress Software Corporation does not support direct modification of the 4GL generated using the 4GL Generator.

A generated-4GL static object functions and interacts with other objects in a similar manner to its static ADM equivalent, but because it does not rely on the same internal ADM property management and processing models, it is likely to perform much more efficiently than the static ADM equivalent. Because of the way dynamic objects are cached on the client after retrieval from the Repository, you can only run generated-4GL static objects in a generated-4GL container object. In general, the 4GL contained in generated-4GL static objects is optimized, as much as possible, for maximum performance, and Dynamics also manages these objects with performance optimization in mind.

Most of this chapter describes how to use the 4GL Generator mechanism. However, the following section (“[Saving dynamic objects as static objects](#)”) describes the basic features of working with the save as static object mechanism.

D.2.2 Saving dynamic objects as static objects

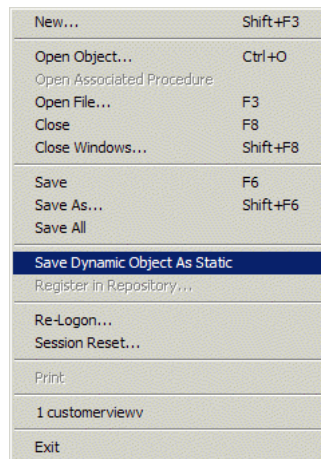
The Progress Dynamics save as static object mechanism allows you to save only dynamic SmartDataObjects and SmartDataViewers as static ADM objects. These static objects function exactly like their dynamic equivalents, except all rendering and behavioral properties defined in the Repository for the dynamic equivalents are specified directly in the 4GL for the static objects. The result is a 4GL procedure that looks and works like any static SmartDataObject or SmartDataViewer that you create using the AppBuilder.

Because they are independent from their dynamic originals, like any Dynamics static object, you must explicitly register these static equivalents of dynamic objects in the Repository in order to use them in a Dynamics application.

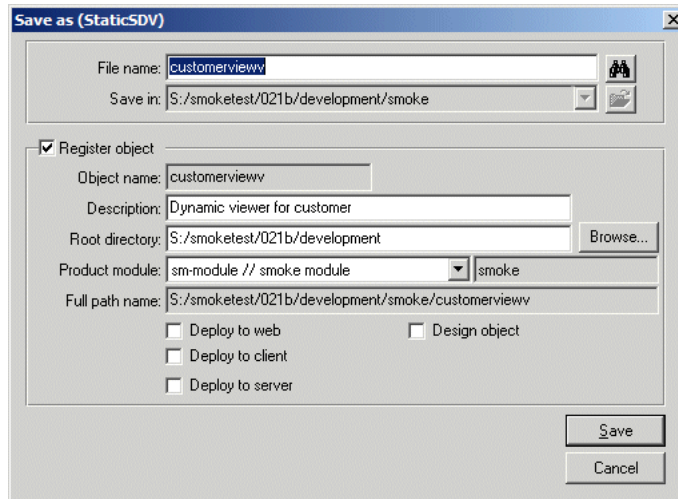
Dynamics provides certain options and imposes some constraints on the construction and use of these static objects in relation to their original dynamic equivalents. The following procedure describes what these options and constraints are and how to handle them.

To save an existing dynamic SmartDataObject or SmartDataViewer as a static object:

- 1 ♦ Open an existing dynamic SmartDataObject or SmartDataViewer in the AppBuilder.
- 2 ♦ Open the **File** menu in the AppBuilder:



- 3 ♦ Choose the **Save Dynamic Object As Static** item, displaying a **Save as** dialog box, as shown:



This figure shows a **Save as** dialog box for a static SmartDataViewer. If the open dynamic object is a SmartDataObject, the title displays as Save as (SDO).

- 4 ♦ Set the following options in this dialog box as you require:
- **File name** — By default, this is the same filename as the open dynamic object. However, note that if you are replacing the existing dynamic object with this static equivalent in a container, the static object cannot have the same name as its dynamic equivalent. Change the name of the static object in this dialog box, then replace the dynamic object in its container with the renamed static equivalent using one of the following windows:
 - Container Builder** — Accessible from the AppBuilder Build menu. For more information, see the *Progress Dynamics Developer's Guide*.
 - Replace Object Instances** — Accessible from **Objects** menu of the Development tool main window. For more information, see the *Progress Dynamics Development Help*.

- **Register object** — By default, this is a check box that is checked and allows you to register the new static object in the Repository. By default, the **Description**, **Root directory**, and **Product module** fields are set to the values specified for the open dynamic object. You can change these settings for the static equivalent and the read-only **Save in** and **Full path name** fields update automatically to show you the new values. You can also check boxes to indicate the deployment types and usage for this static object.

- 5 ♦ Choose **Save** to complete the saving of the static object.
- 6 ♦ If the open dynamic object contains UI events, the save as static function creates triggers in the static code and writes the RUN or PUBLISH statements in the triggers to run or publish the specified events. If you want to move the event code from the object super procedure to the static object, itself, you must do this manually.

D.2.3 Generating 4GL for dynamic objects as static objects

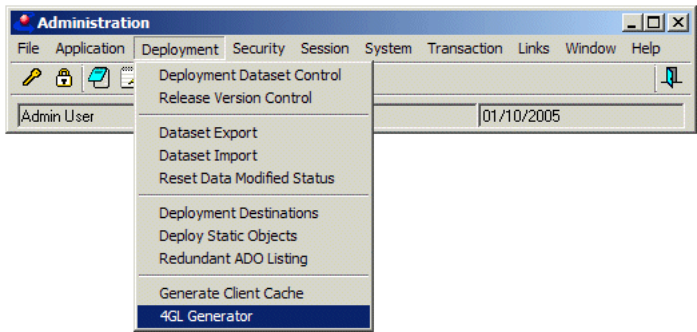
With the 4GL Generator mechanism, you can generate static equivalents of all classes of dynamic objects, except for SmartBusinessObjects and Dynamics Web objects. The 4GL for generated-4GL static objects and their run-time handling is specially optimized for performance. These static objects function exactly like their dynamic equivalents, except all rendering and behavioral properties defined in the Repository for the dynamic equivalents are specified directly in the 4GL for the static objects. Because it is optimized for performance, a generated-4GL object procedure looks and works quite differently from a static object that you create using the AppBuilder.

Unlike other static and dynamic objects, Dynamics does not (and you *must not*) register these generated-4GL static objects in the Repository. They share their registration status, by implication, directly with the original dynamic objects (logical objects) from which they are generated. Dynamics handles the security and some execution features for these generated static objects based on the existing registration of the corresponding logical objects. Thus, generated-4GL static objects are true deployment equivalents of their dynamic originals, and you deploy them together wherever you use them.

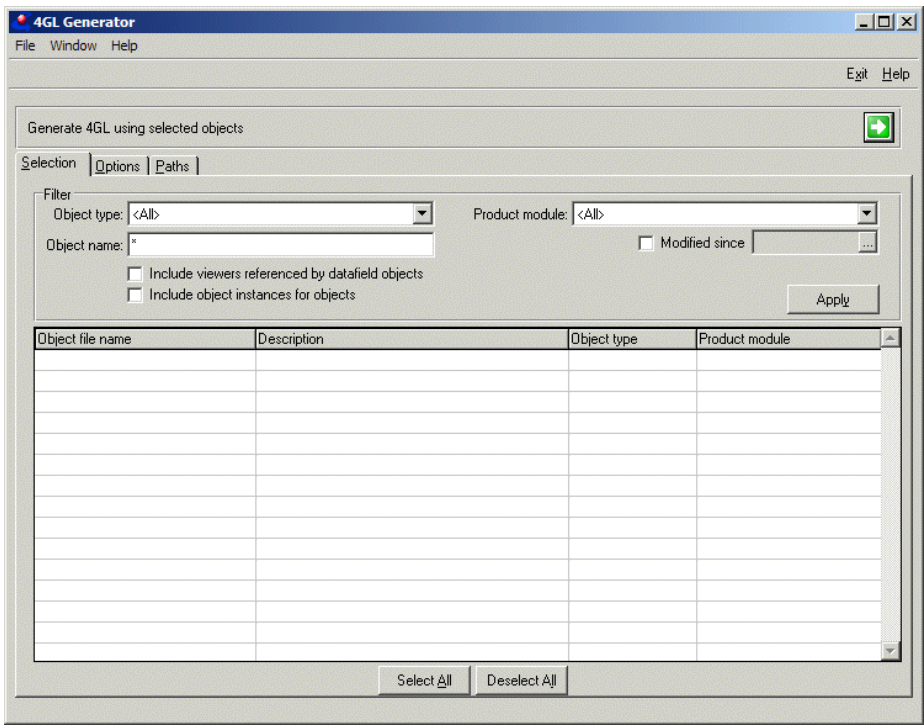
Dynamics offers several options for generating the 4GL for these static objects and imposes some constraints on the construction and use of these static objects in relation to their original dynamic equivalents. The following procedure describes what these options and constraints are and how to handle them.

To generate 4GL for a static equivalent of a dynamic object:

- 1 ♦ In the Administration main window, choose the **Deployment** menu, as shown:



- 2 ♦ Choose **4GL Generator** in the **Deployment** menu, displaying the **4GL Generator** window, as shown:



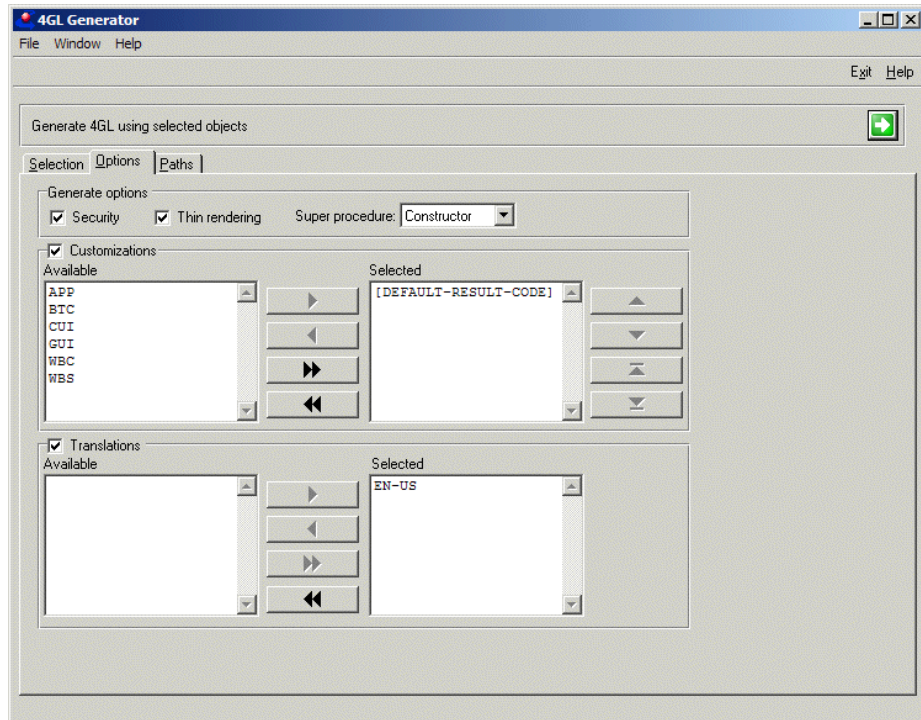
- 3 ♦ Select the **Selection** tab to specify the selection of dynamic objects that you want to generate as static objects:
- a) Set the **Filter** settings to identify the list of objects you want to generate. You can filter this list by a combination of criteria. Note that the **Object name** field must have a value that can include instances of the general wildcard (*) anywhere in the value. For example:

`af*e*w`

You can also separately include viewers that are link targets of DataField objects and and object instances contained in container objects.

- b) After you have specified filter criteria, choose **Apply** to display the corresponding list of objects in the browser.
- c) Individually or as a group, select the objects from this list for which you actually want to generate static 4GL. You can start by selecting all (**Select All**) or none (**Deselect All**) of the items of the list.

- 4 ♦ Select the **Options** tab to specify generation options that apply to all of the object selections you want to generate, as shown:



Use the **Generate options** to specify if Dynamics security, if thin rendering, and how super procedures are generated and invoked for the selected objects. For more information on thin rendering, see the sections on thin SmartObject rendering in [Appendix E](#), “[Performance Notes](#).” The choices for handling custom super procedures include:

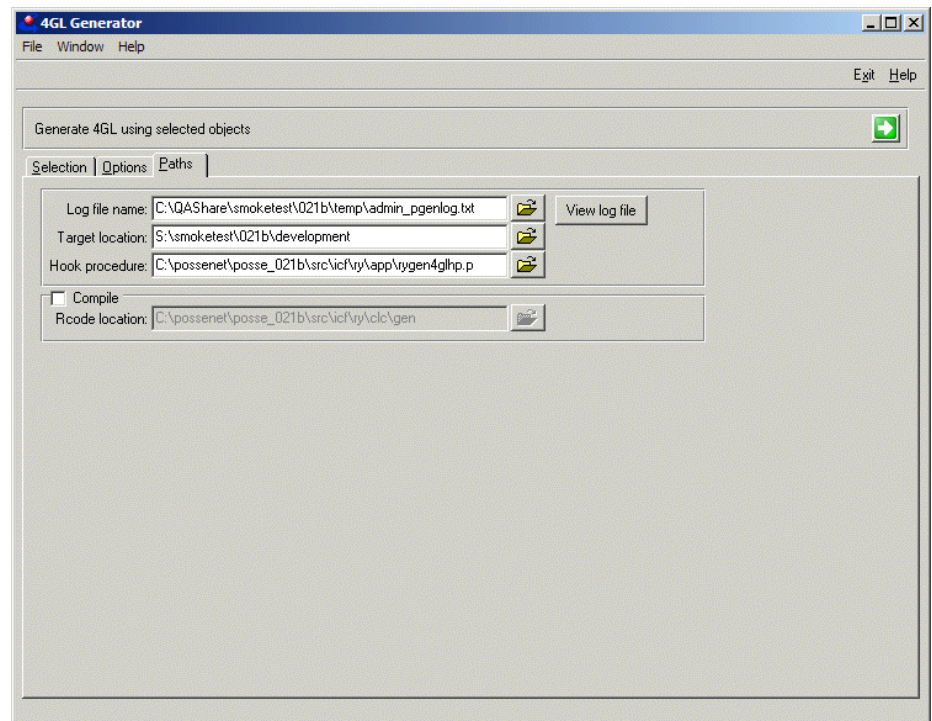
- **Constructor** — Invoked as part of object startup.
- **Property** — Set as a property and invoked by the ADM code.

DataLogic procedures for SmartDataObjects are handled in the same way as super procedures.

Use the **Customizations** selections to specify the result codes (if any) and the order to apply them for generating the selected objects. The selected result codes form part of the filename for each generated object. The result codes in the filename helps Dynamics decide what files to execute for the object. For more information, see the [“Generated-4GL object execution and file naming”](#) section.

Use the **Translations** selections to specify what language translations (if any) to include in the generated objects. Like all other Repository definitions, the generator includes all selected translations in the generated code for access at run time.

- 5 ♦ Select the **Paths** tab to specify various file and path names for the generated objects, as shown:




NOTE: For **Hook procedure**, always accept the default. This setting is reserved for use by Progress Technical Support.

You can choose **View log file** to view the contents of the specified log file in a Notepad window. Note that the generator does not check for existing static objects with the same filenames as currently selected objects and replaces any that it finds in the specified target location with the newly generated files for these objects.

Check the **Compile** toggle box to compile the 4GL and store the r-code for the generated object in the specified location.

NOTE: While the location you choose to store any r-code you generate might be temporary, Dynamics specifies a location where you must deploy the r-code for generated-4GL static objects on the client. For more information, see the [“Deploying generated-4GL static objects”](#) section.

- 6 ♦ In you have completed specifying the generation information, to generate 4GL for the static objects, choose the **Go**  button.

D.2.4 Generated-4GL object execution and file naming

Progress Dynamics invokes generated-4GL static objects as an automatic component of its protocol for invoking any registered object. A generated-4GL static object is not, itself, registered in the Repository, but is associated with an existing logical object that *is* registered in the Repository. When a Dynamics application starts up an object, if Dynamics, locates an appropriate generated-4GL file for this object, it invokes this generated file to render the object. Otherwise, it invokes the base object using its definition in the Repository.

Generated filenames

Dynamics locates generated files according to their filenames and locations. The filenames for generated-4GL static objects derive from the logical object name of the registered object and the customization result codes that you specify for static object generation (see the [“Generating 4GL for dynamic objects as static objects”](#) section).

The general syntax of this generated filename is:

SYNTAX

*LogicalObjectName***[_ResultCodeString]**.pgen

The filename components include:

LogicalObjectName

The logical name of the object as it is registered in the Repository.

[_ResultCodeString]

When you specify customization result codes for object generation, they are appended to the *LogicalObjectName* as shown in the order that you specify them. For more information, see [Table D-1](#).

.pgen

The extension on the filename for a generated-4GL source code file.

[Table D-1](#) shows how the 4GL Generator determines the filename for a generated static object using these components.

Table D-1: Determining the generated filename for a generated-4GL object

Sample result codes used for generation	Object has customizations?	Generated filename components	Sample generated filename for logical object CustWin
Default result code only	No.	<i>LogicalObjectName</i> .pgen	CustWin.pgen
Result codes specified in this order: GUI CLERK ADMIN	Yes. Even if the object is only customized by one of the generation result codes, the whole object is considered customized.	<i>LogicalObjectName</i> <i>_ResultCodeString</i> .pgen	CustWin_GUICLERKADMIN.pgen
Result codes specified in this order: GUI CLERK ADMIN	No. Even if result codes are specified for generation, none are applied to the generated object.	<i>LogicalObjectName</i> .pgen	CustWin.pgen

Locating and executing the files

The 4GL Generator stores all the object source files that it generates in the target directory that you specify. After compiling these files, you must deploy the r-code to a specific subdirectory of the session client cache directory to enable Dynamics to locate and execute them (see the [“Deploying generated-4GL static objects”](#) section).

NOTE: Dynamics uses this deployment area in the session client cache directory instead of the usual `PROPATH` search to locate generated-4GL objects for execution as a performance optimization. For more information on the client cache directory, see [Appendix E, “Performance Notes.”](#)

When Dynamics executes a logical object, it first looks for the r-code for a generated-4GL static equivalent in the client cache deployment area. It identifies the corresponding r-code according to a generated filename whose *LogicalObjectName* and *ResultCodeString* matches the logical object and the current session result code customizations. If an r-code file does not exist with a filename (customized) that matches the session customization settings, Dynamics attempts to execute an r-code file that has the same filename with no *ResultCodeString* component (not customized). If it cannot locate a matching customized r-code file or one that is not customized, or it cannot locate any generated-4GL r-code for the logical object, Dynamics executes the object according to its definition in the Repository.

D.2.5 Deploying generated-4GL static objects

The main complication in the deployment of generated-4GL static objects is that you must also consider deployment of the dynamic equivalents in the process. You must also take care to deploy the r-code for generated-4GL objects in a directory specified by Dynamics. There are also guidelines on when it is necessary to re-generate and re-deploy these objects. This section describe all of these issues.

What to deploy

Table D–2 shows the minimum files that you need to deploy to the client for a given object, depending on the state of the object. As minimum files, ADO indicates the deployment datasets for the abstract definitions in the Repository. PGEN indicates the source and r-code files for the generated-4GL static object equivalents.

Table D–2: Minimum files to deploy for generated-4GL static objects

Object state	Minimum files to deploy
New object.	ADO, PGEN
Modified object, including properties or contained instances, but excluding changes to the object path.	PGEN
Modified object, including properties, contained instances, and changes to the object path.	ADO, PGEN

While, under some circumstances, you can deploy only the PGEN files for generated-4GL objects, Progress Software Corporation recommends that you always deploy both the ADO and PGEN files. Otherwise, you risk losing track of the differences between them. If you find it expedient to deploy only the PGEN files in order to provide a fast and temporary fix or patch, deploy the ADO to the client as soon as possible to maintain consistency between the dynamic and generated equivalents of the object.

Where to deploy

Typically, you deploy the source code files for generated-4GL objects to directories as specified by the object path, per object. You must deploy the r-code for these objects into the gen subdirectory of the client cache directory (typically, *Dynamics_Install\src\icf\ry\clc*). Dynamics ignores the PROPATH and looks *only* in the client cache deployment area for any generated-4GL object r-code that it needs to execute.

When to deploy

After the initial deployment, you need to re-generate and re-deploy generated-4GL static objects when one or more of the following changes occurs:

- Information about the object itself, such as its object path.
- Changes to super procedures, depending on how you generate the object's super procedures (invoked by the object constructor, generated in-line, or generated as an ADM property).
- Properties of the object itself.
- Information about the object's pages and links (any addition, modification, or removal of the links).
- Information about the contained instances, including names, attributes, and UI events.
- For dynamic viewers, any information about the contained objects' master objects. (The DataField and local widgets (not SmartDataFields) are not procedure based. So, all master information must be generated into the viewer. This is also true of class information for the contained widgets. Classes affected include the DataField class and all its subclasses, and the ProgressWidget class and all its subclasses.)

D.2.6 Development impact using generated-4GL objects

The primary development constraint with using generated-4GL static objects in your application is that you can run most such objects only within a generated-4GL container object. So, with one exception, if you decide to deploy a contained instance as a generated-4GL static object, you must also deploy the container as a generated-4GL static object.

The exception to deploying dynamic containers as generated-4GL objects is for dynamic TreeView windows. Because the detail frame objects (run by clicking the nodes of a TreeView) run as master objects, they receive none of their rendering properties from their parent container objects (the dynamic TreeView windows). So, you can run generated-4GL detail frame objects within either dynamic or generated-4GL TreeView windows.

The 4GL Generator mechanism is primarily designed as a performance-enhanced deployment feature. However, if you use generated-4GL static objects in your application, some Dynamics APIs play a particular role in executing these objects that you might need to manage.

Table D–3 lists the APIs that play a particular role and describes how they work with generated-4GL static objects.

Table D–3: APIs for running generated-4GL static objects

(1 of 2)

Name	Manager or Class	Description
menuItemSecurityCheck	Security Manager	Checks the security settings for a specified menu item.
menuStructureSecurityCheck	Security Manager	Checks the security settings for a specified menu structure.
launchContainer	Session Manager	Launches containers. This is the primary point for launching top-level containers, such as menu controller, TreeView, and other windows. It is the API that searches for the generated-4GL for this type of object before attempting to retrieve and render its dynamic equivalent from the Repository.
startDataObject	Repository Manager	Starts SmartDataObjects. It is the API that searches for the generated-4GL SmartDataObject before attempting to retrieve and render its dynamic equivalent from the Repository.
retrieveBandsAndActions	ADM Toolbar	Encapsulates the retrieval and population of the toolbar temp-tables. This call includes the current call to rygetmensp.p and also attempts to populate the menu temp-tables using the generated adm-loadSmartToolbar() call both in the toolbar's container (for object menus) and in itself (for toolbar menus).

Table D–3: APIs for running generated-4GL static objects

(2 of 2)

Name	Manager or Class	Description
constructObject	ADM Container	Launches contained objects. This API starts the object and sets up Page (if relevant) and Container links. It searches for the generated-4GL for the object before attempting to retrieve and render its dynamic equivalent from the Repository.
getMappedFileName	Repository Manager	Returns the full path name for a given logical object name. This API uses the map file concept to determine the filename. If it cannot find a physical file, the API returns the unknown value (?).

For more information on the manager APIs that interact with generated-4GL objects, see the [Progress Dynamics Managers API Reference](#), and on the ADM APIs, see the [Progress Dynamics ADM2 API Reference](#).

D.3 Deploying Repository objects between Progress Dynamics versions

With the release of both the OpenEdge Release 10 and Progress Dynamics Version 2.1 release families, both you and Progress Software Development might find it necessary to fix bugs and add features to both development streams at the same time. When a bug fix results in a source code change, the task is no different than for any other development environment. However, when the bug fix affects data in the Repository, the situation is more complicated, because Repository objects must be deployed between different versions of the Repository (Release 10 and Version 2).

To address this situation, Progress Software Corporation provides deployment tools that manage the differences in Repository versions and recommends a specific set of ordered tasks to manage the deployment of objects between them.

D.3.1 Cross-version deployment situations

You might well be maintaining an application running in both Dynamics release families at the same time.

Two main categories of development might need to be deployed between these application versions:

- New objects built in the development Repository
- Existing objects maintained in either Repository

If you add new functionality to the application in one release family, you often add the same functionality to the application in the other family. If the implementation is identical in both families, it is simply a matter of developing it in one release Repository and exporting it to the other release Repository. Complications arise where you implement different application features or use different implementation techniques between the versions.

When you maintain existing objects between versions, aside from adding features to an object in one version that do not exist in another, you might use implementation techniques in one version that do not exist in another.

Any cross-version deployment must account for all of these scenarios.

D.3.2 Progress Dynamics support for cross-version deployments

The Progress Dynamics deployment tools provide the following features to support cross-version deployments between the OpenEdge 10 and Progress Dynamics 2.1 release families:

- The export and import functions of the deployment tools recognize deployment datasets (ADOs) from both release families.
- The dataset import functions ignore unknown tables (tables in the ADOs that do not exist in the target Repository).
- The dataset import functions ignore unknown fields (fields in import tables that do not exist in the same tables of the target Repository).
- The dataset import functions ignore unknown data types. That is, if a field in an import table has a data type not supported in the target Repository, the import functions ignore those fields.

D.3.3 Recommended procedure for deploying across versions

The procedure that Progress Software Corporation recommends for deploying objects across versions is not much different from deploying an object to a central Repository. For more information on Repository object deployment and the tools described in this procedure, see the deployment white papers posted on the following PSDN Web site:

<http://psdn.progress.com>

To deploy Repository objects across versions:

- 1 ♦ Before undertaking any work that needs to be deployed across versions, determine what objects will be affected and what those objects look like in each version. If you plan to create a new object, you can create it in either version (see [Step 2](#)). If you plan to change an existing object, determine what additional functionality can be lost in either version.
- 2 ♦ Once you understand the affected objects, decide the version where you will do the work. Typically, it is better to do the development work in the later version, where you are likely to add more features. It is always easier to remove functionality from an object than to add it. The version where you do the initial development work is your *source version*, and the version where you deploy this initial development work is your *target version*.
- 3 ♦ Before doing any work in the source version, reset the data modified status for your source development Repository (using **Deployment→Reset Data Modified Status** from the Administration tool) so you can easily determine what changes you have made.
- 4 ♦ Implement your changes.
- 5 ♦ Dump all the data you have modified using the dataset export tool (**Deployment→Dataset Export**). The exported ADOs contain the changes that you made.
- 6 ♦ Load the ADOs that you exported in [Step 5](#) into the central Repository of your source version using the dataset import tool (**Deployment→Dataset Import**).
- 7 ♦ Before doing any work in the target version, reset the data modified status for your target development Repository (using **Deployment→Reset Data Modified Status** from the Administration tool) so you can easily identify the affected objects.
- 8 ♦ Load the ADOs that you exported in [Step 5](#) into the development Repository of your target version using the dataset import tool (**Deployment→Dataset Import**). In Dataset Import, be sure to check the **Set modified status** toggle box so these objects can be easily identifiable after they have been loaded.

- 9 ♦ Test the objects you have loaded and re-do any work that has been lost during the import.
- 10 ♦ Complete any additional functionality required to finish the objects for the target version.
- 11 ♦ Dump all the data you have modified in the target development Repository using the dataset export tool (**Deployment→Dataset Export**). The exported ADOs contain the objects you imported from the source version plus any additional changes that you have made.
- 12 ♦ Load the ADOs that you exported in [Step 11](#) into the central Repository of your target version using the dataset import tool (**Deployment→Dataset Import**).

Your cross-version deployment should be complete.

Performance Notes

This appendix provides information that can help you to tune your Progress Dynamics applications for optimum performance.

Although most of the performance improvements implemented in the current release are generally applicable and therefore automatically delivered to an application, there are additional improvements where your application knowledge is required to make the best choice of where to balance trade-offs. These optimizations are therefore optional and enabled through configuration decisions that you make. This appendix describes these configurable optimizations:

- [Configuring server and client startup options](#)
- [Using static-4GL equivalents of dynamic objects](#)
- [Keeping SmartDataObjects alive on the server](#)
- [Progress Dynamics lookup/combo \(SmartDataField\) cache](#)
- [Dynamic lookup mapped fields](#)
- [Class and entity cache](#)
- [Toolbar Image Optimization Using PicClip Images](#)
- [Caching toolbars and container menus at session startup](#)
- [Thin SmartObject rendering](#)
- [Dynamic TreeView optimizations](#)

- [SmartDataObject data definition and schema location](#)
- [SuperProcedureMode attribute](#)
- [Dynamic call wrapper\(dynlaunch.i\)](#)
- [Creating customized login windows](#)

For more information on performance features available in Progress Dynamics, see the performance white papers posted on the following PSDN Web site:

<http://psdn.progress.com>

E.1 Configuring server and client startup options

Table E–1 lists recommended startup parameter settings for both server and client that can improve both performance and overall execution.

Table E–1: Optimal settings for server or client startup parameters

Startup Parameter	Value	Description
-baseADE	""	Procedure libraries added to the PROPATH in the initialization (.ini) file (on the client).
-Bt	512	Buffer size for temp-tables.
-D	500	Directory size.
-l	1000	Local buffer size.
-mmax	65534	Maximum memory.
-nb	150	Nested blocks.
-s	128	Stack size.
-T	/tmp	Directory to store temporary files.
-TB	31	Speed sort.
-TM	32	Merge number.

For more information on these startup parameters, see the *Progress Startup Command and Parameter Reference* from the Progress Version 9 documentation. You can also find this information online at the following Web site:

<http://www.progress.com/products/documentation/index.ssp>

You can also improve performance by taking these actions:

- If your databases reside on the same machine as the AppServer, use direct database connections, instead of networked connections on the server process. For example, in the server `icfconfig.xml` file, use:

```
<cConnectParams>-db /apps/database/icfdb/icfdb.db</cConnectParams>
```

instead of using:

```
<cConnectParams>-db icfdb -H localhost -S icfdb</cConnectParams>
```

- For any procedure libraries in the `PROPATH`, put them in the optimum order.
- Do not put `src` on the `PROPATH` of the server or the client. Put only `gui` or `tty` and the procedure libraries on the `PROPATH`. You can set the `PROPATH` in the `.ini` file on the client and in the `ubroker.properties` file on the server (using the Progress Explorer tool).

For example, on the client you might specify this `PROPATH` in `dynamics.ini`:

```
PROPATH=.,/dyn21a/gui,/dyn21a/gui/adecomm.pl,/dyn21a/gui/adecomp.pl,/dyn21a/gui/adedesk.pl,/dyn21a/gui/adedict.pl,/dyn21a/gui/adeedit.pl,/dyn21a/gui/adeicon.pl,/dyn21a/gui/aderes.pl,/dyn21a/gui/adeshar.pl,/dyn21a/gui/adetran.pl,/dyn21a/gui/adeuib.pl,/dyn21a/gui/adeweb.pl,/dyn21a/gui/adexml.pl,/dyn21a/gui/product.pl,/dyn21a/gui/prottools.pl,/dyn21a/gui/dynamics
```

On the server, you might specify this `PROPATH` in `ubroker.properties`:

```
PROPATH=.,/dyn21a/tty/dynamics,/dyn21a/tty,/dyn21a/tty/adecomm.pl,/dyn21a/tty/adecomp.pl,/dyn21a/tty/adeedit.pl,/dyn21a/tty/adeshar.pl,/dyn21a/tty/product.pl
```

E.2 Using static-4GL equivalents of dynamic objects

Dynamic objects incur two major performance hits:

- Retrieving their abstract definitions over the network when the first instance starts up.
- Transforming the abstract definition into a physical object when it first starts up.

Progress Dynamics allows you to eliminate most of these performance hits by deploying any selection of dynamic objects in your application as static-4GL objects. You can generate and deploy these static object equivalents for all dynamic objects except for most container objects that contain them, all SmartBusinessObjects, and Dynamics Web objects.

When you deploy static equivalents of dynamic objects, you typically maintain them in dynamic form for ongoing development and regenerate the static equivalents in order to deploy the updated objects that you release.

For more information, see the sections on generating and deploying static 4GL for dynamic objects in [Appendix D, “Deployment Notes.”](#)

E.3 Keeping SmartDataObjects alive on the server

When running over a stateless AppServer, dynamic SmartDataObjects (SDOs) must be retrieved, constructed, and initialized on the server for every client request. In terms of performance, this process can sometimes be more expensive than fetching the data itself. To eliminate this overhead, you can configure SDOs to remain in server-side memory (keep alive) for reuse by subsequent client requests.

NOTE: In the current release, you cannot keep alive static SDOs running in static containers.

E.3.1 Configuring SDOs to remain alive

You can configure the server-side life time of an SDO using the SDO property, `DestroyStateless`. This property is defined in the `Data` class with a default value of `TRUE`. This means that the server-side SDO is destroyed after every client request. By setting this property to `FALSE` you configure the corresponding SDO to remain in AppServer memory after the completion of a client request. You can set `DestroyStateless` at the Class level (affecting all SDOs throughout the application), at the Master object level (affecting all instances of a particular SDO), or at the Instance level (affecting a particular occurrence of an SDO in a container).

E.3.2 Using SDOs kept alive on the server

When you evaluate whether to deploy SDOs to stay alive on the server, consider these guidelines:

- Once started, an SDO remains in memory until the host process is terminated. This effectively means that to clear SDOs, you must either trim agents or stop and restart the broker. This applies to both AppServer and WebSpeed agents.
- Because each client request can be directed to a different agent, the same SDO will most probably be started and remain in memory in multiple agents. Each such SDO takes up more server memory resources. However, each SDO memory footprint is relatively small.
- When processing a client request, if a required SDO is in memory (as determined by its LogicalObjectName), the server re-uses it regardless of the DestroyStateless property value. This property is only used to decide the fate of the SDO upon request completion.
- Although you can set DestroyStateless at the instance level, a kept-alive SDO has no knowledge of the particular instance that starts it. An SDO is only recognizable and retrievable by its LogicalObjectName. So, many kept-alive SDOs can be started after setting this property for a single instance.
- The most time-consuming task when starting an SDO is setting up any required dynamic elements such as dynamic queries, dynamic RowObject and RowObjUpd tables, dynamic buffers, and so on. Therefore, dynamic SDOs benefit the most from being kept alive, especially those with many fields.
- Datasets created as part of a client request are destroyed upon request completion. However, custom data structures that are created, such as temp-tables, remain as part of the SDO. To change this behavior, you can override destroyObject in the SDO (or its DataLogic procedure, if one is used) to specify any custom cleanup between requests.

E.4 Progress Dynamics lookup/combo (SmartDataField) cache

Progress Dynamics provides a client-side cache manager that supports the optional caching of data for use by dynamic combos and lookups. This cache for dynamic SmartDataFields (SDFs) reduces network traffic and improves overall performance.

By reducing the amount of data and having the data in a pre-processed and ready-to-consume form on the client, the data-gathering time and any AppServer requests to fetch the data are eliminated, thus improving the overall performance of the application. This is especially noticeable over a slow network.

This SDF cache is a great benefit for performance. However, it might lead to the display of incorrect values if the data in the cache becomes stale. To force retrieval of the latest information with the correct values, you can clear the cache at any time.

SDF caching is optional and configurable. The following sections describe:

- [Cache operation](#)
- [When Dynamics does not use the SmartDataField cache](#)
- [Enabling and disabling the SmartDataField cache](#)
- [Clearing the SmartDataField cache](#)
- [Disabling SmartDataField \(SDF\) cache usage for a specified SDF](#)

For more information on the impact of using the SDF cache for development, see the [Progress Dynamics Developer's Guide](#).

E.4.1 Cache operation

The SmartDataField cache is dynamically built at run time; therefore, the benefits of having the cache increase with use of the application. Whenever a container runs with dynamic lookups or combos on a viewer, a record is created in a temp-table on the client machine where data retrieved from the server is stored. When ever the same container reruns, it reuses this cached data when other containers having the same type of SDF also run. The cache therefore provides benefit only on subsequent uses of the same data.

For a practical example of where the SDF cache benefits an application, suppose there is a container with a SmartToolbar, dynamic SDO, dynamic browser, and dynamic viewer, and that these objects operate on the Customer table in the Sports2000 database with the SDF cache configured for use.

This viewer contains a dynamic lookup on the State field and a dynamic combo on the SalesRep field. The first time the container runs, the application makes an extra AppServer request to populate the value for the lookup's displayed field as well as for populating the list-items for the SalesRep combo. If the user clicks on the next record, the application will make an AppServer request to get the displayed field information for the State lookup *only* if the state of the second selected record is different from that of the previously selected record. If the user selects the previously selected record again, there is no AppServer request, because the lookup data is retrieved from the SDF cache. If this viewer only had a dynamic combo, there would not be another AppServer request when the second record was selected, because the combo's data were cached on the first AppServer request. If the user closes the container and reruns it, no AppServer request is required, because the data is already cached. As the user selects records in

the browser where the data is not yet cached from the first run, the application makes an AppServer request to retrieve the data and caches it. After values have been cached for all query states, no more AppServer requests are required.

If another container that also has a dynamic combo for the SalesRep field runs, and the base query of the dynamic combo is the same as the base query used to build the list items of the SalesRep combo used in the initial Customer maintenance viewer, the cached information for the first combo is reused for this other combo also.

E.4.2 When Dynamics does not use the SmartDataField cache

Progress Dynamics does not use the SmartDataField cache for dynamic lookups or combos in certain situations, even if the cache is enabled for use.

Cache not used for dynamic lookups

Dynamics does not use the cache for dynamic lookups in the following circumstances:

- The dynamic lookup has a parent filter query where the values of the parent filter fields do not match the values of the data already in the cache.
- When the lookup icon (binoculars) is chosen or the **F4** key is pressed to launch the lookup browser, the data to be displayed in the browser is still retrieved directly from the server as in previous releases.
- When a value is entered in the lookup field and the user leaves the field, either by tabbing off the field or using any other mouse or keyboard event that causes the lookup field to lose focus.
- The base query string in the lookup does not match the base query string exactly of the data already in the cache.

Cache not used for dynamic combos

Dynamics does not use the cache for dynamic combos in the following circumstances:

- When the dynamic combo has a parent filter query where the values of the parent filter fields do not match the values of already cached data in the cache.
- If the base query string in the combo does not match the base query string exactly of that cached in the client cache.

E.4.3 Enabling and disabling the SmartDataField cache

SmartDataField caching is optional for any session and can be enabled for any client session. The SDF cache is only available for client-side sessions.

For any upgrade from a previous version without SDF caching, the SDF cache manager is added to application Repository databases during the upgrade process. To enable SDF caching for a specified session type, add SDFCacheManager as a Required Manager using Session Type Maintenance (Session→Session Type Control from the Administration tool) and generate a new Dynamics configuration .xml file. The result creates the following entries in your icfconfig.xml file:

```
<manager>
  <cManagerName>SDFCacheManager</cManagerName>
  <cFileName>ry/prc/rysdfcmngr.p</cFileName>
  <cHandleName>NON</cHandleName>
</manager>
```

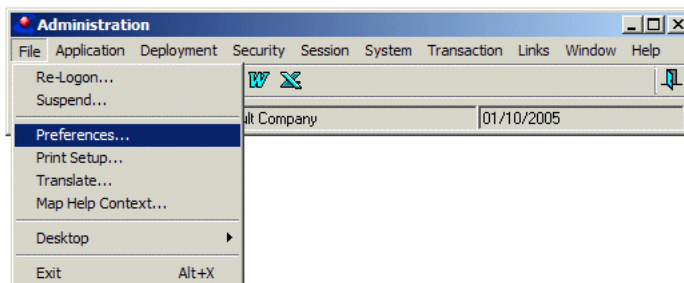
To disable SDF caching from the same session type, remove SDFCacheManager as a Required Manager from the specified session type using Session Type Maintenance and regenerate the icfconfig.xml file.

E.4.4 Clearing the SmartDataField cache

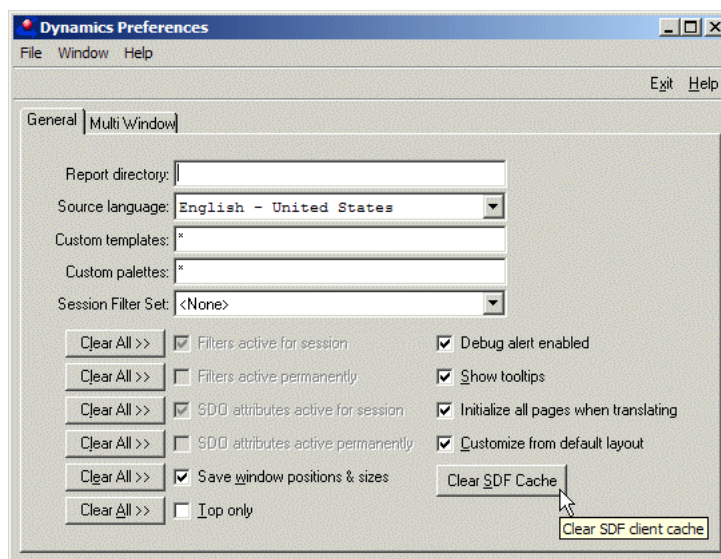
You can clear the SmartDataField client cache at any time by launching the Preferences window available from the File menu available on any Dynamics menu controller window.

Thus, to clear the SmartDataField cache from the Administration tool:

- 1 ♦ Choose **File→Preferences**, as shown:



- 2 ♦ In the **Dynamics Preferences** window, choose the **Clear SDF Cache** button, as shown:



- 3 ♦ Choose **Yes** for the confirmation message.

E.5 Dynamic lookup mapped fields

By default, the dynamic lookup uses its own query-management and data-retrieval mechanism, instead of using SmartDataObjects. Using this mechanism the dynamic lookup requires a separate AppServer request to retrieve lookup data as part of rendering a container and displaying the current record.

To minimize the need for the AppServer request, the dynamic lookup can check if the fields required to display the lookup values for the current record are also available in the viewer SDO. If corresponding SDO fields are available, the dynamic lookup can use these fields instead of making a separate and duplicate AppServer request. If the fields are not available in the SDO, the lookup must make its own AppServer request to populate it with data.

In cases where the fields used on the viewer for a dynamic lookup are not the actual fields contained in the SDO, the affected lookup must make its own AppServer request. To solve this issue and to provide a more specific way to map the lookup fields used on the viewer to the fields in the SDO that contain their data (if applicable), the concept of field mapping was introduced.

To allow dynamic lookups to check the SDO for the required fields, Progress Dynamics supports a field mapping mechanism for use by dynamic lookups on both static and dynamic viewers. Field mapping provides an ability to explicitly state what fields on a viewer are linked to specified fields in an SDO. The need for you to explicitly map the fields is only required if the fields on the viewer do not directly map to the fields in the SDO based on their names.

E.5.1 Benefits and constraints on mapping fields

The most common reason for mapping fields is to enable Progress Dynamics to link the correct field from the SDO with a linked field on the viewer required for dynamic lookups. You can have the fields mapped automatically by specifying fields from the data source on the viewer as the linked fields in the lookup. In cases where you cannot do this, you can explicitly map the fields to ensure that the linked fields are populated with the correct value.

For field mapping to work, you must map all of the fields in the dynamic lookup to the viewer SDO. For more information, see the [“Using mapped fields”](#) section.

Progress Dynamics does not support field mapping for viewers using a SmartBusinessObject (SBO) as their data source.

NOTE: Adding joins to an SDO in order to make fields available for a dynamic lookup as part of the standard SDO data retrieval does not always provide a performance benefit. The expense of the extra join, plus the extra data returned, might cancel any benefit that might be gained were the fields already available in the SDO without the join. You must assess the trade-off individually for each case.

E.5.2 Using mapped fields

You can use the mapped field feature when there is a dynamic lookup on a viewer and you need to display extra information on the viewer for the selected or displayed lookup record. In this case, you can add a field to the viewer and link this field to a selected field in the lookup browser that displays when a selection is made.

For example, we can modify the CustomerCodeLookup in the order viewer that you build as part of the Progress Dynamics tutorial using the dynsports database. (For more information on this tutorial, see [Getting Started with Progress Dynamics](#).) For the tutorial, you create this lookup on the fill-in for the customer object field (customer_obj) that you generate as part of the dynamic viewer (artorviewv) for art_order. For a given order, this lookup displays the customer name that you select from the lookup browser with a query on arm_customer. The lookup browser displays both customer_code and customer_name for each arm_customer, and the lookup also contains a link from customer_balance to a fill-in (fiCustBalance) that you create on the viewer so the balance can be displayed for the selected customer.

Preparing the lookup for field mapping

In our tutorial example, CustomerCodeLookup uses three fields from the query on arm_customer:

- customer_balance
- customer_code
- customer_name

In order to avoid AppServer requests when displaying the fields used by a lookup, you must map *all* of the fields used in the lookup query to the viewer data source (SDO), including the displayed field, all fields specified for the lookup browser sequence, and any specified linked fields. For this example, you must map the three lookup fields to corresponding fields in the SDO for art_order (artorfullo).

Because the tutorial generates the artorfullo SDO as a data source for art_order only, you must explicitly add the arm_customer fields for the lookup to this SDO:

1 ♦ Modify the SDO to join art_order to arm_customer (on customer_obj, by default).

2 ♦ Add the fields from arm customer to the SDO like this:

- **customer_balance** — Not updateable.
- **customer_code** — Not updateable.
- **customer_name** — Updateable.

Mapping lookup fields to the SDO

You can map the lookup fields in the following ways:

1. [Linking lookup fields directly to SDO fields on the viewer](#)
2. [Mapping lookup fields to SDO fields through viewer widgets](#)

These two options essentially provide the same results at run time. However, for cases where you cannot use the first option, you can use the second option to get the lookup data from the SDO.

In our example from the tutorial, you can use either option. However, the first option produces a less practical result, with an extra (and redundant) field on the viewer.

NOTE: Dynamics only uses the field mapping mechanism when records are being displayed from the data source. If the user changes a lookup's value, the default behavior for retrieving these values (from the AppServer) executes. However, the mechanism still improves performance and reduces AppServer requests.

E.5.3 Linking lookup fields directly to SDO fields on the viewer

To link the fields from the tutorial CustomerCodeLookup directly to the SDO fields:

- 1 ♦ Ensure that you have created a join from art_order to arm_customer in the art_order SDO (artorfullo) and added the following fields for the lookup from arm_customer to the SDO:
 - customer_balance
 - customer_code
 - customer_name

For more information, see the [“Preparing the lookup for field mapping”](#) section.

- 2 ♦ After you have saved the updated SDO, open the art_order viewer (artorviewv) in the AppBuilder.

- 3 ♦ Drop the SDO fields for the lookup on the viewer using the DB Fields object from the Palette. The Multi-Field Selector for DB Fields lists the newly-added SDO fields for you to select and add to the viewer.

NOTE: After you drop the fields onto the viewer, one of them, the customer_name object, has the same label (Customer Name) as the CustomerCodeLookup.

- 4 ♦ Right-click on the CustomerCodeLookup and choose **Instance Properties** from the pop-up menu. This displays the SmartDataField Maintenance (Instance Properties) window.
- 5 ♦ In the query field browser, select the row for each of the lookup fields and type the column values as shown in [Table E-2](#).

Table E-2: Column values for lookup field mapping directly to SDO

For Field name...	Set Link field to...	Set Linked widget to...
arm_customer.customer_balance	YES	customer_balance
arm_customer.customer_code	YES	customer_code
arm_customer.customer_name	YES	customer_name

This is SmartDataField Maintenance (Instance Properties) as it appears after you have finished the field links (with customer_name scrolled out of view):

6 ♦ Save the instance properties to complete the field linkage.

As noted in [Step 3](#), using this option to map the lookup fields creates two redundant field objects on the viewer labeled Customer Name:

- The CustomerCodeLookup on the customer_obj field from the SDO (art_order).
- The customer_name field from the SDO (join to arm_customer).

Both of these fields display the same value of customer_name as displayed or selected in the lookup. There is no way to complete the mapping of all lookup fields using this option without duplicating the display of that same field mapped from the SDO.

Thus, this option linking lookup fields directly to SDO fields works best if the SDO field displayed in the lookup is the same as the viewer field on which the lookup is created. For example, you might use this option to map the fields of a lookup on the tutorial viewer for arm_customer (armcuvviewv), where the lookup is created on the customer_code field, which already provides the value from the SDO for display in the lookup.

For the CustomerCodeLookup example created on customer_obj, you can fix the redundancy for the user by permanently hiding the customer_name field object used to map the lookup display field to the corresponding SDO field. However, you might prefer to use the second option for mapping lookup fields, through separate viewer widgets mapped to the SDO.

E.5.4 Mapping lookup fields to SDO fields through viewer widgets

To map the fields from the tutorial CustomerCodeLookup to SDO fields through viewer widgets:

- 1 ♦ Ensure that you have created a join from art_order to arm_customer in the art_order SDO (artorfullo) and added the following fields for the lookup from arm_customer to the SDO:

- customer_balance
- customer_code
- customer_name

For more information, see the [“Preparing the lookup for field mapping”](#) section.

- 2 ♦ After you have saved the updated SDO, open the art_order viewer (artorviewv) in the AppBuilder.
- 3 ♦ Add a new fill-in widget, fiCustCode, to the viewer, formatted to display arm_customer.customer_code.

NOTE: This example assumes that you have already added the fiCustBalance widget as part of creating CustomerCodeLookup in the tutorial. For more information, see [Getting Started with Progress Dynamics](#).

- 4 ♦ Right-click on the CustomerCodeLookup and choose **Instance Properties** from the pop-up menu. This displays the **SmartDataField Maintenance (Instance Properties)** window.

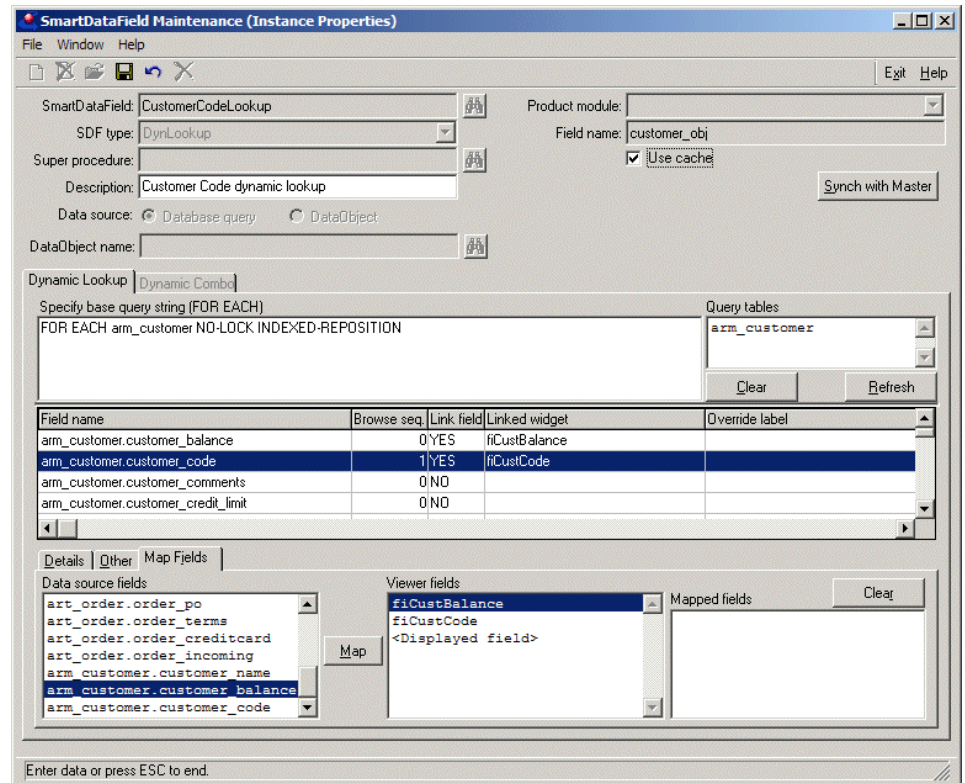
- This is **SmartDataField Maintenance (Instance Properties)** as it appears after you have set these column values:

NOTE: The corresponding column values for the `arm_customer.customer_balance` row are already been set from the tutorial.

- E-18

7 ♦ Map these viewer fields to the corresponding SDO fields:

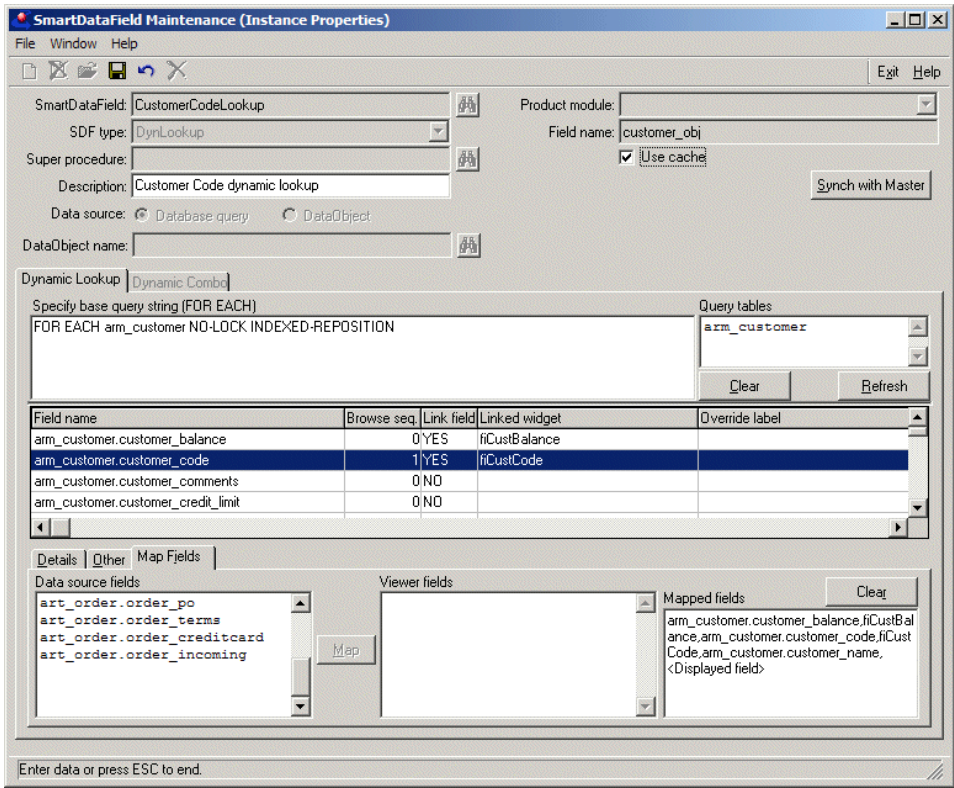
- a) Select the **arm_customer.customer_balance** field in the left selection list and the **fiCustBalance** field in the right selection list. The **Map** button now becomes enabled, as shown:



- b) Choose the **Map** button. The two selected fields disappear from the selection lists and appear in the Mapped fields editor.
- c) Repeat [Step a](#) and [Step b](#) for the **arm_customer.customer_code** field in the left selection list and the **fiCustCode** field in the right selection list.

- 8 ♦ Map the Displayed field for the lookup:
- a) Select the **arm_customer.customer_name** field from the left selection list and **<Displayed Field>** from the right selection list.
 - b) Choose the **Map** button.

The entire field mapping for the lookup appears in the **Mapped field** editor, as shown:



- 9 ♦ Save the instance properties to complete the dynamic lookup field mapping.

The field redundancy that the first mapping option creates for this example (see the [“Linking lookup fields directly to SDO fields on the viewer”](#) section) does not occur using this option. The **Map Fields** tab allows you to specify the mapping of the lookup Displayed field to the corresponding SDO field without the need for a linked viewer widget that would otherwise create a redundant field on the viewer mapped to the SDO that displays the same value. Although this mapping option requires more steps to set up, it has the flexibility to accommodate field mappings for any lookup, no matter how it is defined.

E.5.5 When Dynamics does not use mapped fields

Even if you have mapped the fields for a dynamic lookup to linked fields on the viewer, Dynamics might not use this field mapping mechanism to retrieve data for the lookup. This happens when a later change is made to the data source that renames or removes one of the mapped fields, or a mapped field on the viewer has been renamed or removed. If this happens, to ensure that the user always gets the correct data, Dynamics does not attempt to get the linked field values from the SDO, but instead makes the extra AppServer request to return all of the correct information to be displayed in the lookup.

E.6 Class and entity cache

In Progress Dynamics, a class or object type represents an inheritance mechanism for attributes that defines the behavior of the objects in that class. Entities in Dynamics define a logical schema in the Repository that corresponds to schema tables in the framework and application database. This class and entity information represents a large amount of data that typically does not change often, and, because of the object orientated nature of the framework, this information is expensive to access and organize at run time.

The *class and entity cache* adds an additional deployment step that pre-creates the data in a form that is readily usable at run time, and creates the data persistently so it can be deployed onto each client machine. This cache avoids the expense of constructing the information at run time, and it removes the need to transfer the data over the network at run time, thereby improving the overall performance of the application.

E.6.1 Benefits and constraints on the class and entity cache

Note that even without this cache, the construction and retrieval of entity and class data only happens once—the first time a new entity or class is referenced in a session. So, once all entities and classes have been used, the performance with or without the cache is the same. Therefore, the class and entity cache provides a performance gain for cached entities and classes during the session startup and first-use of session objects.

These performance gains and network data reductions come at the cost of some deployment overhead; so, the use of this cache is optional. If you change class attribute defaults, extend the class hierarchy, add new attributes, or change the application database schema, you must regenerate and redeploy the class and entity cache to each client.

E.6.2 Cache architecture

As the name implies, this caching mechanism accommodates two distinct types of data:

- **Class cache** — Containing attribute and other information that defines the class hierarchy.
- **Entity cache** — Containing Repository and application database schema information.

Class cache

Class information along with its attributes amounts to an average data size of 10Kb per class. The class definition does not change frequently and is needed to render each of the ADM2 objects. The class definition in Progress Dynamics maps to the ADMProps temp-table definition, which is one of the main data structures in the ADM2 architecture.

Caching the class information in separate procedure files and using the local information at run time significantly reduces the amount of data over the network and AppServer requests required to construct the class hierarchy. The server processing time is also greatly reduced, thereby increasing the availability of AppServer agents for other requests.

Entity cache

Dynamic viewers and SDOs are the main consumers of the entity cache, because they rely heavily on the data field information in the Repository. Passing the data field information, such as NAME, DATA-TYPE, FORMAT, LABEL, and COLUMN-LABEL, for each instance of a data field requires expensive server processing (fetching this information from database frequently) and associated data passed over the network. The data field information cannot be overridden at the instance level, so there is no danger of caching many copies of the same data fields. Therefore, entity information can be easily cached on the client machines to reduce the network data traffic, AppServer requests, and processing time.

The entity information is cached as Progress dump (.d) files containing information from the data field masters.

E.6.3 How the Progress Dynamics framework uses the cache

Each object rendering procedure uses the class and entity information to render the objects in the application. This is how Dynamics checks availability of and fetches the information:

1. If the session property, StartupCacheClasses, is provided with the CVS list classes to cache for the session, Dynamics caches these classes in memory at startup.
2. The client checks if the class or entity information is available in memory, and uses it if found. Any of the previous calls in this client session might have fetched this information.

3. If the requested class or entity information does not exist in memory, the client checks its machine's local disk for the class or entity information, and uses it if found. The session property, `client_cache_directory` (absolute path to an existing directory), can specify the search directory.
4. If no cache is generated or information is not found in the cache, the client makes a server request to fetch the information from the server. In an AppServer configuration this represents an extra AppServer request. In a slow network environment, the cost, in time, of a server round trip can be relatively expensive.
5. At session shutdown, Dynamics writes any cached entities out to disk.

NOTE: Dynamics writes only the entity cache out to disk, not the class cache.

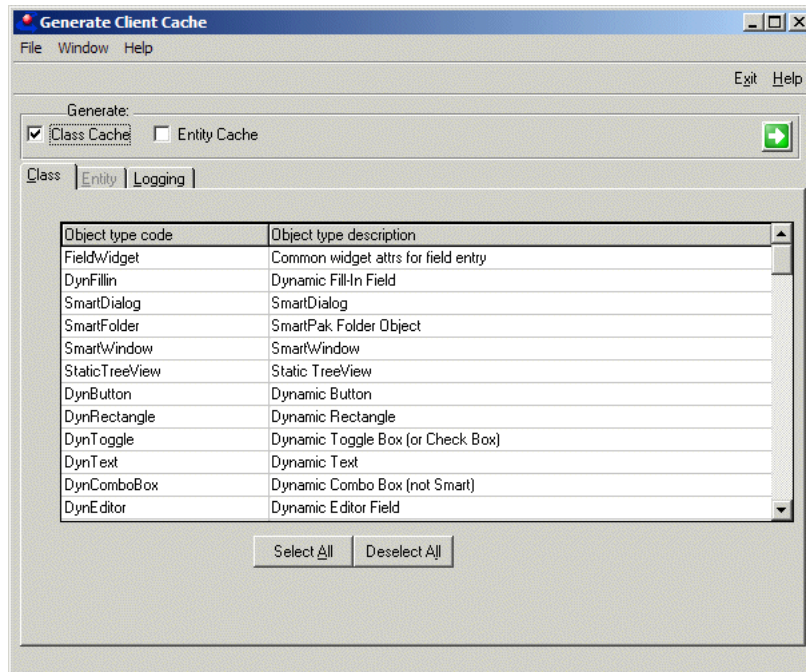
If the `client_cache_directory` session property is specified, the cache is written to that directory. Otherwise, Dynamics writes the entity cache to the directory for the ry-clc product module. The session property, `auto_dump_entity_cache`, allows you to specify whether to have the shutdown process dump the entity cache to disk. For more information on these session properties, see the [“Configuring the class and entity cache”](#) section.

E.6.4 Generating the cache

Progress Dynamics provides a tool to generate the class and entity cache for deployment.

To generate the cache:

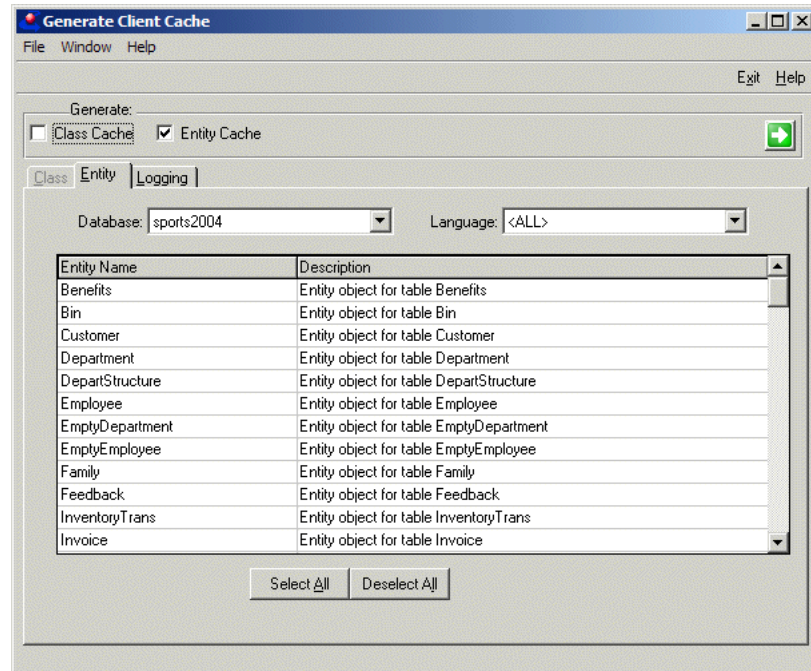
- 1 ♦ From the Administration tool, choose **Deployment**→**Generate Client Cache**, displaying the **Generate Client Cache** window, as shown:




- 2 ♦ If you want to generate a class cache, check the **Class Cache** toggle box and choose the **Class** tab. The browser on the **Class** tab displays classes where the cache_on_client field in the gsc_object_type table is set to TRUE. You can multi-select as many object types as you want to cache.

NOTE: If you extend the classes provided with Progress Dynamics, and if you choose to cache those classes on the client, you can set the cache_on_client value using the **Cache on client** toggle box from either the Object Type Maintenance tool or the Repository Object Maintenance tool. For more information on these tools, see the [Progress Dynamics Developer's Guide](#).

- 3 ♦ If you want to generate an entity cache, check the **Entity Cache** toggle box and choose the **Entity** tab, as shown:



The browser on the **Entity** tab displays the entities that have been imported into the Progress Dynamics Repository. You can multi-select as many entities as you want to cache.

- 4 ♦ Once you select the classes and entities you want to cache, you can choose the **Go**  button to generate the class and entity cache.

By default, Dynamics generates the class and entity cache in the directory specified for the ry-clc product module. For better control, Progress Software Corporation recommends that you use the `client_cache_directory` session property to specify where the client cache is generated. Using this property keeps all client cache content in one place, allowing you to easily deploy the cache to clients. Setting the `client_cache_directory` property ensures that the specified directory is searched for the class and entity cache. Using this property also eliminates one AppServer request (during AppServer configuration) required to obtain the relative path of the client cache based on the ry-clc product module settings.

E.6.5 Configuring the class and entity cache

Progress Dynamics provides these session properties that you can use to configure the class and entity cache:

- **client_cache_directory** — An absolute path where the client cache is generated and accessed. Setting this property eliminates one AppServer request (in AppServer configuration) otherwise required to obtain the relative path of the client cache based on the ry-clc product module.

You can use this property to help separate the development session from your run-time sessions.

- **auto_dump_entity_cache** — Indication telling Dynamics if it is to dump the entity cache to disk at session shutdown. If this property is set to YES, at shutdown, Dynamics writes any entities in memory to disk. This allows you to reuse the entity information from the disk instead of going to the server with additional AppServer requests. The default value is YES.

This property primarily provides additional control to Progress Dynamics developers, where you typically set it to NO. In development sessions, you do not want entities cached, because you have to repeatedly delete the cache when you make changes to your entity information. However, in a runtime environment, you typically set it to YES to provide the performance benefits for the application.

- **StartupCacheClasses** — A CVS list of classes that you want cached at startup, instead of fetching the class information when needed. Concentrating the caching of classes at startup allows faster rendering for containers during the greater part of the session duration.

Specify the "*" wild card to have Dynamics pre-cache all the classes from the Repository at startup.

NOTE: Because a value of "*" results in a large amount of data movement during login, be careful when and how you use this option.

E.6.6 Using the class and entity cache

The major cost of class and entity cache is in the deployment overhead. If you change class attribute defaults, extend the class hierarchy, add new attributes, or change the application database schema, you must regenerate and redeploy the class and entity cache to each client.

If you have generated the cache using the Generate Client Cache tool or you have set the `auto_dump_entity_cache` session property to YES, any changes you make to the entity (schema tables) and class data (including adding of new properties or changing the defaults for the class data) in design-time sessions require that you update the cache for all clients. If you do not regenerate the cache using the Generate Client Cache tool and deploy the regenerated cache appropriately, none of your changes will be visible on the affected clients.

E.7 Toolbar Image Optimization Using PicClip Images

Toolbars support the use of PicClip images. PicClip images are a series of multiple images in a grid format that are stored in a single bitmap (. bmp) file. [Figure E-2](#) shows a PicClip image that contains many of the toolbar images used in the Progress Dynamics toolbars.



Figure E-2: PicClip file, toolclip.bmp

Dynamics maintains this file in the following location:

Dynamics_Install/src/icf/ry/img/toolclip.bmp

E.7.1 Benefits and constraints on using PicClip images

In earlier releases of Dynamics, a separate image file was maintained and loaded for each button in a toolbar. By combining all images into a single PicClip image, the time required for loading these images has decreased substantially over loading them as separate images. Along with name of the image file itself, Dynamics loads an X and Y offset, width, and height that identifies the portion of the PicClip containing a specific image. The performance improvement of using this mechanism is most noticeable the first time a container is launched and the images have not yet been loaded into the operating system cache.

NOTE: Even when not using PicClip images, the Dynamics image-loading process is optimized to minimize the number of searches using the relative path name of the image.

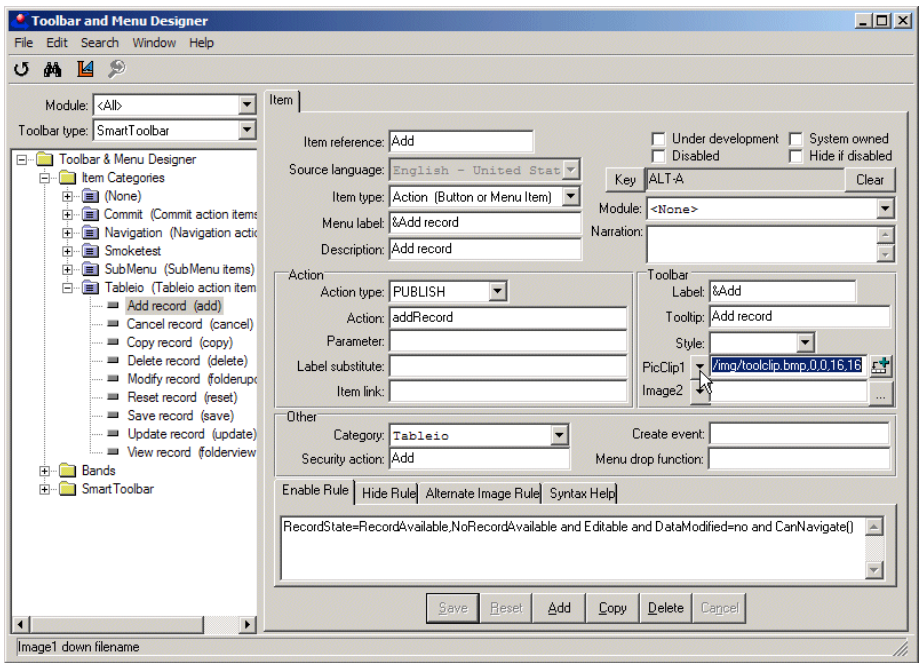
Dynamics only supports the use of PicClip images for the GUI environment.

E.7.2 Using PicClip images for custom toolbars

If you have created your own toolbar images, Progress Software Corporation recommends that you use your own toolbar PicClip image rather than appending your images to the Dynamics-supported PicClip image file (toolclip.bmp). You can specify the PicClip image file, along with the offsets and image size, using the Toolbar and Menu Designer. The Toolbar and Menu Designer allows you to specify images either from a PicClip image file or a single image file, and you can specify primary and alternate versions of each.

To specify a PicClip image:

- 1 ♦ Open the Toolbar and Menu Designer and select an item that requires an image, as shown:



- 2 ♦ Click the down-arrow button next to the Image1 (primary image) field and select **PicClip1** from the pop-up menu.
- 3 ♦ In the **PicClip1** field, specify an image file (relative path name) together with the X and Y offset, the width, and the height (in pixels) in the following comma-delimited format:

SYNTAX

image-pathname,X-offset,Y-offset,width,height

You can also specify the relative path name of a separate image file containing a single image (if provided), using the **Image1** field. At run time, Dynamics attempts to load the PicClip image before loading any separate image specified for Image1.

- 4 ♦ If you have one, you can repeat [Step 3](#) to specify an alternate image using the **Image2** and **PicClip2** fields.

NOTE: The specified PicClip image file information is stored in the fields `gsm_menu_item.image1_down_filename` and `gsm_menu_item.image2_down_filename` for the primary and alternate images, respectively.

E.8 Caching toolbars and container menus at session startup

Dynamics usually extracts toolbar items and container menus when the container they belong to is extracted from the Repository. These items are then cached on the client and reused as necessary. This client caching mechanism caches the bands and actions against the respective toolbar or container. When the toolbars and menus for a new container are cached, Dynamics always extracts all bands and actions because the server cannot determine what information has already been cached on the client. Where the same bands are used across multiple toolbars or menus, this results in a duplication of work if a band has already been cached for another container. Caching toolbars and menus at session startup (*toolbar and menu pre-caching*) reduces this duplication of work by ensuring that any bands reused on multiple toolbars or containers are only processed once.

E.8.1 Benefits and constraints on toolbar and menu pre-caching

When a toolbar or object menu cannot be found in the client cache, the client has to make an AppServer request to retrieve the necessary information. Caching multiple toolbars at session startup results in fewer AppServer requests given that each toolbar cached in the initial request would otherwise have required a later request.

As with any caching at session startup, caching toolbars and container menus at session startup requires a longer Dynamics startup time. An extra AppServer request is also made when the session starts. It is also possible that toolbars and menus cached at session startup might not be used in the session, in which case the caching overhead incurred at startup has been unnecessary.

E.8.2 Configuring toolbar and menu pre-caching

Progress Dynamics provides two session properties to activate this feature. You must specify these session properties in the Progress Dynamics configuration `.xml` file:

- **StartupCacheToolbars** — A comma-delimited list of logical names for toolbars to pre-cache at session startup. Specify the "*" wild card to cache all toolbars.
- **StartupCacheMenusForObjects** — A comma-delimited list of logical names for Repository containers. The menus for these containers will be pre-cached at session startup.

To cache menus for containers against a specific run attribute, specify name and run attribute code for each container in the list using this syntax:

SYNTAX

`container-name [; run-attribute-code]`

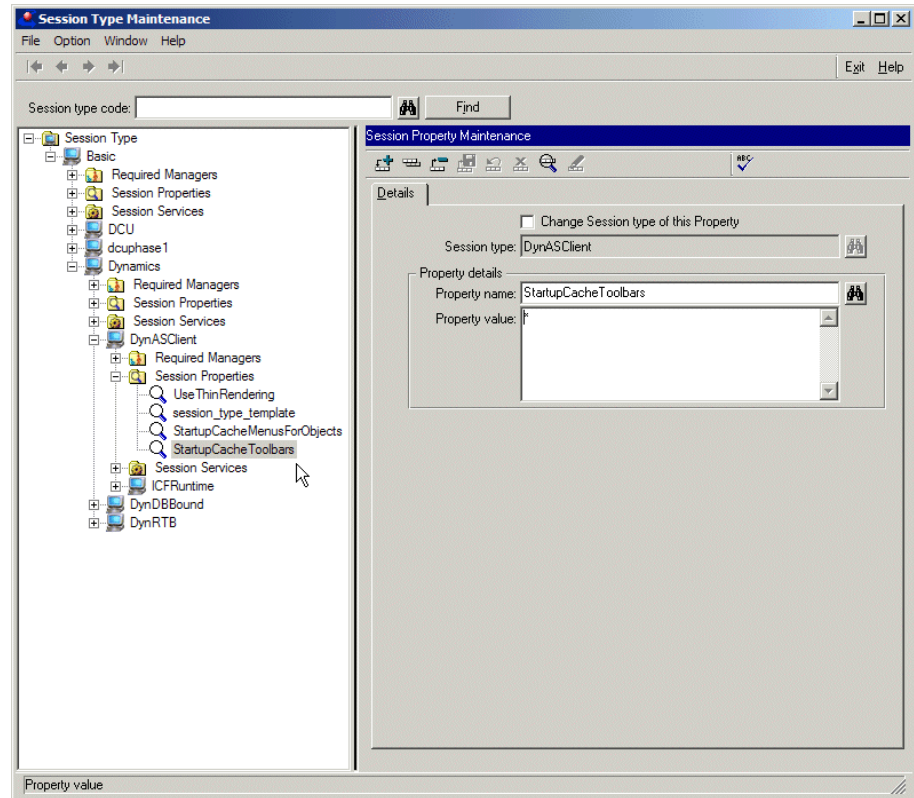
If the run attribute does not apply to a given container, specify *container-name* only for the specified container.

These properties are not specified by default. Therefore, no toolbar or menu pre-caching occurs unless you specify them.

To specify the session properties for pre-caching toolbars and menus:

- 1 ♦ Open the Session Type Maintenance tool by choosing **Session→Session Type Control** in the Administration tool.

- 2 ♦ Add the session properties to appropriate client session types, such as **DynASClient**, as shown:



- 3 ♦ Generate a new Dynamics configuration .xml file by choosing **Session→Generate Configuration File** in the Administration tool. This creates entries in the .xml file for the applicable session types, like the following:

```
<properties>
.
.
<startupCacheToolbars>*/startupCacheToolbars>
<startupCacheMenusForObjects>aFallmencw</startupCacheMenusForObjects>
.
.
</properties>
```

E.8.3 Using toolbar and container menu pre-caching

Note the following guidelines when using this feature:

- The session properties for configuring this feature only have meaning in a client session type; do not set them for AppServer sessions. These properties also do not apply in Dynamics Web sessions.
- Progress Software Corporation recommends that you identify and pre-cache all toolbars that are used extensively in an application.
- Container menus rarely need to be pre-cached. Pre-cache container menus only when the same menu bands are used extensively for different containers.

E.9 Thin SmartObject rendering

SmartObject rendering and initialization are expensive. Running SmartObject instances involves method library include files, prototype include files, property include files, extended class include files and logic to set various properties based on static preprocessor values. The existence of this code adds to the size of the compiled r-code and to the cost of execution.

Thin versions of various Progress SmartObjects™ exist that contain less r-code than their full counterparts. The thin versions exclude logic from their compilations that are not required for most dynamic objects, such as prototype and custom include-file inclusion and logic to create ADMProps.

Thin SmartObject rendering procedures exist for the following dynamic objects:

- Dynamic SmartDataObject
- Dynamic lookup
- Dynamic combo
- Dynamic browser
- Dynamic viewer

These thin rendering procedures include their dynamic object and rely on preprocessor definitions to optimize the rendering information. There are two preprocessor definitions specific to thin SmartObject rendering:

- **ADM-EXCLUDE-PROTOTYPES** — Every ADM property include file contains definition checks to exclude the compile of the ADM prototype include file. For example:

```
&IF DEFINED(ADM-EXCLUDE-PROTOTYPES) = 0 &THEN
&IF "{&ADMSuper}":U EQ "" :U &THEN
{src/adm2/combprto.i}
&ENDIF
&ENDIF
```

- **ADM-EXCLUDE-STATIC** — Every ADM property include and class include file contains definition checks to exclude the compilation of code only required for static objects. For example:

```
&IF DEFINED(ADM-EXCLUDE-STATIC) = 0 &THEN
    {src/adm2/custom/datacustom.i}
&ENDIF

&IF DEFINED(ADM-EXCLUDE-STATIC) = 0 &THEN
    IF NOT {&ADM-LOAD-FROM-REPOSITORY} THEN
        RUN start-super-proc("adm2/appserver.p":U).
    &ENDIF

    &IF DEFINED(ADM-EXCLUDE-STATIC) = 0 &THEN
        IF NOT {&ADM-PROPS-DEFINED} THEN
            DO:
            &IF "{&ADMSuper}":U = "" :U &THEN
                ghADMPProps:ADD-NEW-FIELD('ObjectLayout':U,'CHARACTER':U, 0, ?, '' :U).
                ghADMPProps:ADD-NEW-FIELD('LayoutOptions':U,'CHARACTER':U, 0, ?, '' :U).
                ghADMPProps:ADD-NEW-FIELD('ObjectEnabled':U,'LOGICAL':U, 0, ?, no).
                ghADMPProps:ADD-NEW-FIELD('LayoutVariable':U,'CHARACTER':U, 0, ?,
                '{&LAYOUT-VARIABLE}':U).
                ghADMPProps:ADD-NEW-FIELD('DefaultLayout':U,'CHARACTER':U, 0, ?, '' :U).
                .
                .
                .
            &ENDIF
        &ENDIF
    &ENDIF
```

In addition to defining ADM-EXCLUDE-PROTOTYPES and ADM-EXCLUDE-STATIC, the thin rendering procedures define exclude-start-super-proc to exclude compilation of the start-super-proc internal procedure. Here is an example of a thin rendering procedure:

```
&GLOBAL-DEFINE ADM-EXCLUDE-PROTOTYPES
&GLOBAL-DEFINE ADM-EXCLUDE-STATIC
&SCOPED-DEFINE exclude-start-super-proc
{adm2/dynsdo.w}
```

E.9.1 Configuring thin SmartObject rendering

Progress Dynamics provides the ThinRenderingProcedure attribute to define the thin rendering procedure for a given object. Thin rendering procedures are defined for the following classes:

- **DynSDO** — adm2/thinsdo.w
- **DynLookup** — adm2/thinlookup.w
- **DynCombo** — adm2/thincombo.w
- **DynBrow** — ry/obj/rythinbrowb.w
- **DynView** — ry/obj/rythinvieww.w

If this attribute is blank, Dynamics always uses the procedure specified by the RenderingProcedure attribute to render the object.

Dynamics provides the UseThinRendering session property to specify if Dynamics uses a defined (non-blank) thin rendering procedure to render a given object at run time. If this property is FALSE, Dynamics uses the procedure specified by RenderingProcedure to render the object.

UseThinRendering defaults to TRUE for the DynASClient and Default session types and defaults to FALSE for all other session types.

Thin dynamic SDOs cannot run on the server side because they are initialized on the server based on context passed from the client rather than reading the Repository. Thin SmartObjects must be initialized from the Repository.

NOTE: Even if UseThinRendering is set to TRUE for the DynAppServer session types, the RenderingProcedure attribute is used to set the SDO's ServerFileName instead of the ThinRenderingProcedure attribute.

E.9.2 Using thin rendering

In rare cases where application code relies on logic that has been excluded from the compile of thin objects, re-evaluate the logic and change it, if possible. For example, reliance on prototype definitions requires direct references to functions to change to DYNAMIC-FUNCTION in order to take advantage of thin objects. If dependencies in the excluded logic cannot be resolved, do not use thin rendering.

You can turn off thin rendering for all objects with the UseThinRendering session property, or you can turn it off for a class of objects, for individual object masters, or for object instances by setting the ThinRenderingProcedure attribute to a blank value.

Certain development tools, such as the Container Builder, require the full objects rather than their thin counterparts. You can set UseThinRendering to TRUE in a development session type and thin objects might run in these sessions without a problem. However, some development tools will not always function properly. Therefore, always set UseThinRendering to FALSE for a development session type.

E.10 Dynamic TreeView optimizations

The dynamic TreeView makes use of dynamic frames to construct the object on the right-hand side and this helps it to use the object exactly as it was designed. Unlike previous versions, the dynamic TreeView does not replace the toolbar of your object with a generic toolbar. This change affords far more flexibility than in previous versions. In addition, the dynamic frames used by the TreeView can be kept alive, saving time by not having to destroy and re-create them every time a node associated with the frame is reselected. You can configure how these dynamic frames are kept alive.

The dynamic TreeView also supports a node batching feature that you can configure. If your TreeView object is likely to return a large result set, you can set a maximum number of records to batch, returning subsets of records at one time. If more records are available following the current batch, a '...More' node is added as the last node allowing the user to get the next batch of records.

E.10.1 Keeping dynamic frames alive on the client

When a user selects and deselects (opens and closes) the node associated with a dynamic container, Progress Dynamics manages the frame's resources differently depending on certain session and object settings. So, Dynamics can manage a dynamic frame using one of two mechanisms:

1. Create the dynamic frame each time the user opens it, then destroy the frame each time the user closes it. With this mechanism, Dynamics incurs the performance overhead of creating and destroying the frame with each use, but allocates a minimum of memory to manage all dynamic frames in a session.
2. Create the dynamic frame the first time the user opens it, then hide (keep alive) the frame each time the user closes it. With this mechanism, Dynamics eliminates the performance overhead of creating and destroying the frame with each use, but allocates additional memory with each dynamic frame that it keeps alive.

Properties to manage dynamic frame memory

Dynamics provides the session property, `MaxHiddenContainers`, for you to specify how you want Dynamics to manage the memory resources for dynamic frames used by a given dynamic `TreeView` instance. To manage these resources, Dynamics uses this property together with the `HideOnClose` and `HideChildContainersOnClose` attributes on dynamic frames and dynamic `TreeView`s.

If you set `MaxHiddenContainers` to a value n greater than zero, Dynamics creates a hidden frame list of up to n frames in each `TreeView` that it runs in a session, then displays and hides these frames each time a user opens and closes them. Any additional frames opened by the user replace the least recently used frame in the list.

If you set the value to 0 (zero), to the unknown value (?), or you do not specify the property at all for a session type, Dynamics maintains no hidden dynamic frames, but instead creates each frame when the user opens it (selects a `TreeView` node that displays the frame) and destroys each frame when the user closes it (selects another `TreeView` node to display a different frame).

Configuring dynamic frame memory use

Starting with Progress Dynamics Version 2.1A, Dynamics predefines the `MaxHiddenContainers` property for all run-time client sessions with a default value of 10. You can use Session Type Maintenance tool to change this value for any session type, or add the property to run-time sessions when upgrading from a previous version, then add it to your Dynamics configuration `.xml` file (`icfconfig.xml`) by generating a new configuration file.

In the configuration file, an entry appears for each session type like this (using the default value):

```
<MaxHiddenContainers>10</MaxHiddenContainers>
```

NOTE: A dynamic TreeView can potentially have many dynamic frames running in a session, which can consume large amounts of memory. Before setting a relatively high value for this property, make sure that each client machine that runs a given session type can handle the specified maximum number of hidden frames.

E.10.2 Configuring the node batching feature

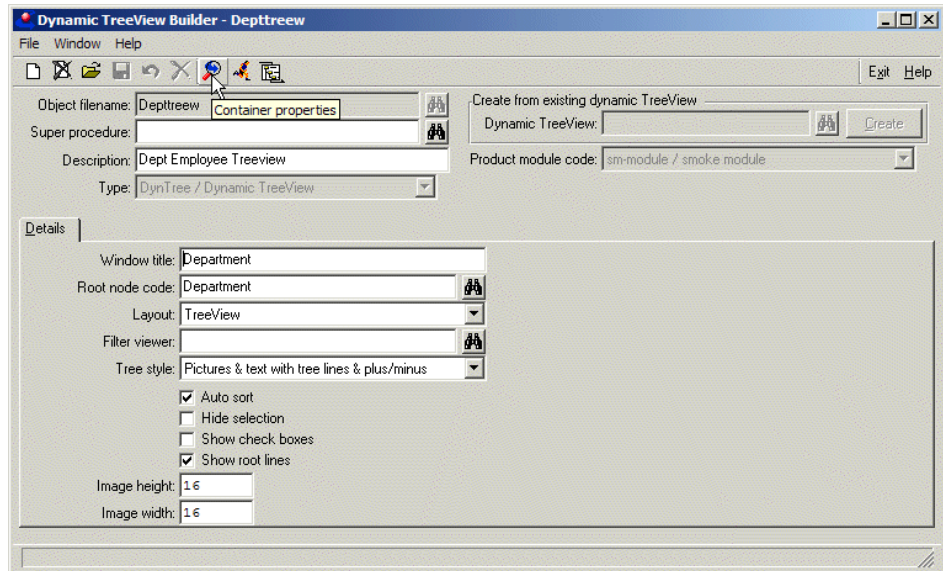
You can configure the number of nodes to batch for a dynamic TreeView using the RowsToBatch attribute at the DynTree class level or on each dynamic TreeView instance. Setting the RowsToBatch attribute to 0 (zero) at the DynTree class level turns off node batching for all dynamic TreeView objects, and at the instance level turns off node batching for the given instance. This forces all nodes to be read at one time.

NOTE: When migrating from a Progress Dynamics version earlier than Version 2.1A, Dynamics automatically enables the node batching feature for all dynamic TreeView objects.

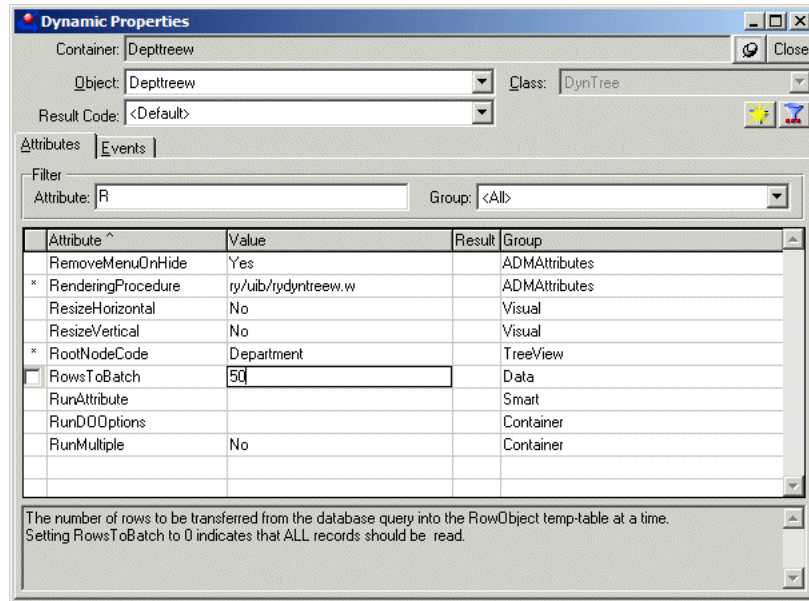
You can configure node batching individually for each TreeView instance by changing the RowsToBatch attribute using the Dynamic TreeView Builder tool.

To change RowsToBatch for a dynamic TreeView instance:

- 1 ♦ Choose **Build→Dynamic TreeView Builder** from the AppBuilder or Development tool to open the **Dynamic TreeView Builder**, as shown:



- 2 ♦ Choose the container properties toolbar button to open the dynamic property sheet (**Dynamic Properties** dialog box), as shown:



- 3 ♦ Scroll to the **RowsToBatch** attribute and set its value as you require.

NOTE: If you use node batching and also have the AutoSort attribute is set to YES for the TreeView object (intending to sort the nodes), you must ensure that your data source (SDO) is sorting on the same field(s) specified as the fields to be displayed as the node label. This is required because when you use node batching, Dynamics ignores the AutoSort setting in order to ensure that '...More' is always positioned as the last node in the TreeView. Otherwise, if the batched data does not come properly sorted from your data source, it will not be sorted in the TreeView. If you later turn off node batching (set RowsToBatch to 0 (zero)), the TreeView again functions according to the AutoSort setting.

E.10.3 Using dynamic TreeViews

Note the following guidelines when using dynamic TreeViews, especially when upgrading from previous versions:

- The optimized dynamic TreeView released as of Version 2.1A eliminates some private and undocumented APIs that are no longer used, but were present in previous versions of the dynamic TreeView.
- If code in previous versions relies on the dynamic TreeView to be the source container of an object, you must review the logic, because for the optimized TreeView, the object's source container is the dynamic frame, and the dynamic frame's source container is the dynamic TreeView.
- When configuring the number of frames to keep alive, you must ensure that the client machine running the given session can handle the maximum number of hidden frames because use of memory increases with an increasing number of running objects.
- If you use node batching and also have the AutoSort attribute is set to YES for the TreeView object (intending to sort the nodes), you must ensure that your data source (SDO) is sorting on the same field(s) specified as the fields to be displayed as the node label. This is required because when you use node batching, Dynamics ignores the AutoSort setting in order to ensure that '...More' is always positioned as the last node in the TreeView. Otherwise, if the batched data does not come properly sorted from your data source, it will not be sorted in the TreeView. If you later turn off node batching (set RowsToBatch to 0 (zero)), the TreeView again functions according to the AutoSort setting.

E.11 SmartDataObject data definition and schema location

Progress Dynamics maintains the field definitions for data objects in the Repository meta schema recorded in Entity and EntityFields, not from the Data Dictionary. This also applies for static SDOs, except that a field's initial value is whatever the code specifies during compilation.

E.11.1 Configuring the location for field definitions

You can modify the location of field definitions by setting the `SchemaLocation` attribute defined for the Data class. These are the possible values and their meanings for where the SDO gets this information:

- **ENT** — Uses the Repository's entity cache for its field definitions (default value).
- **DLP** — Copies its field definitions from the DataLogic procedure buffer.
- **BUF** — Copies its field definitions from the database buffer.

E.11.2 Using the `SchemaLocation` attribute

Change this attribute for performance reasons only very rarely, especially for dynamic SDOs. The entity cache exists to improve performance for all objects that need schema information at runtime, both visual objects and data objects, but the actual caching has to happen somewhere. Setting `SchemaLocation` to `ENT` adds overhead to the SDO startup the first time it is started if the particular entities are not already cached. However, changing the SDO to not use the entity cache, in most cases, moves the responsibility and overhead to some other object. Also note that entities, in most cases, are already cached, either at startup or by startup of another object.

Factors to consider before changing this setting include:

- SDOs that do not use any visual objects and that access entities not used by other objects can get a performance benefit from using the `DLP` setting, because these particular entities do not need to be cached.
- SDOs that do not use the `ENT` setting can be used without importing entities into the Repository.
- The `BUF` setting is currently 50% to 70% slower than the two other settings.
- SDOs that have `OpenOnInit` set to `TRUE` define the temp-table on the first request to the server; SDOs that have `OpenOnInit` set to `FALSE` need to define the temp-table on the client, and thus require the cache to be on the client. You cannot use the `BUF` setting on SDOs with `OpenOnInit` set to `FALSE`.
- The `DLP` setting is static in the sense that the information is not any fresher than the last compilation of the DataLogic procedure.

NOTE: Static SDOs derive a performance benefit from not using the `ENT` setting, because the Entity definition must be added in an additional loop. The `DLP` setting is currently ignored for static SDOs, because both `BUF` and `DLP` make the SDO use the definitions already compiled into them.

E.12 SuperProcedureMode attribute

The SuperProcedureMode attribute indicates a mode for how a class or object super procedure is started. Possible valid values include:

- **STATELESS** — The super procedure starts once per session and stays alive after object shutdown. There is only one instance of the super procedure running in any given session.
- **STATEFUL** — One instance of the super procedure starts per object. The super procedure is shut down when the object is destroyed. Multiple instances of the super procedure might run during a session, depending on the number of concurrently running objects.

The default for classes is STATELESS mode and the default for object (viewer, browser, container, SmartDataField, and so on) super procedures is STATEFUL mode.

E.12.1 Using the attribute

To reduce the size of the client footprint, always try to write super procedures so they can be run using STATELESS mode.

E.13 Dynamic call wrapper(dynlaunch.i)

Avoiding the use of `dynlaunch.i` and the dynamic call wrapper for AppServer calls reduces the time required to retrieve data from the AppServer and reduces the amount of data transmitted in an AppServer request.

Using the dynamic call wrapper to retrieve data from the AppServer can add overhead in an AppServer call because of the need to construct and pass temp-tables as parameters between the client and the server. Given that the overhead might be small for smaller AppServer calls (at an average of 70 ms), the overhead increases as the number of parameters increases so that a call using the dynamic call wrapper can add as much as 500ms to an AppServer request in a worst-case scenario.

As of Progress Dynamics Version 2.1A, all uses of `dynlaunch.i` are removed from Progress Dynamics managers in cases where a call is likely to be made often, for example the `getPropertyList` and `setPropertyList` APIs in the Session Manager.

The dynamic call wrapper is still used by the framework, but not in places where the framework would pay a significant and perceptible overhead.

E.13.1 Using dynlaunch.i

You can continue to use `dynlaunch.i` for most calls without significant impact on performance. Only in cases where there is a significant benefit to be gained, or data on the network must be reduced to an absolute minimum, should you consider replacing `dynlaunch.i` with a static call. Where a static call is used, take care to ensure that a single, non-persistent procedure is called on the server so that no more than one AppServer request is incurred for the call.

E.14 Creating customized login windows

When you create a customized login window, if you use the same layout as the default login window (`aftemlognw.w`) and include "loginw" in the name of the login window, you will avoid one extra AppServer request.

During the startup of a Dynamics session it is necessary to translate the data on the login window. Unfortunately, the controls on the login window are not known until the data is retrieved by the client.

This means that an AppServer request must be made to retrieve the login window information from the AppServer, and a second AppServer request is then made to retrieve translation information for the window. To avoid the extra AppServer request for translation information, code is added to the `cachelogin.p` procedure that retrieves the login window information to automatically retrieve the translation information for the default login window—`aftemlognw.w`.

Many users do customize the login window, though, and to ensure that this is possible, the functionality that retrieves the translation information also supports retrieving the same information for any login window that contains the character string "loginw" in the name of the login window.

E.14.1 Using Dynamics support for customized login windows

If you intend to customize the login window and you want to avoid the extra AppServer request associated with translating the login window, ensure that the customized login window conforms to the following guidelines:

1. The customized login window must include the string "loginw" in the object name of the login window.
2. The customized login window must contain the same objects as the default login window (`aftemlognw.w`). This means that the objects need to be the same type and name as the default login window.

Index

A

ADM-EXCLUDE-PROTOTYPES

preprocessor variable E-33

ADM-EXCLUDE-STATIC preprocessor
variable E-33

ADO *See also Dataset (ADO) files* 5-11

AM handle code C-6

analyzeCase function

Installation User Interface Manager 5-28

analyzeIf function

Installation User Interface Manager 5-29

APP physical sessions C-3

Application

categories 4-25

feature support 4-1

Login dialog box 2-5

menu overview 1-3

user-defined status 4-27

AppServer

connection manager property C-2

AS handle code C-5

AU handle code C-6

Audience xi

Audit Control 1-10

Auditing 4-19

enabling on a table 4-20

triggering an entry 4-21

viewing on a record 4-22

auto_dump_entity_cache session property
E-26

B

Batch-mode DCU 5-47

creating the command script 5-49

dcuphase1.p 5-50

dcuphase2.p 5-50

generating input data 5-48

-icfparam parameters 5-48

sample command script 5-51

Bold typeface

as typographical convention xiii

BTC physical sessions C-3

btnChoose procedure

Installation User Interface Manager 5-29

C

- Caches
 - updating during deployment 5–10
- Cascading security information 3–24
- Categories
 - application 4–25, 4–26
 - user 3–7
- Category Control 1–4
- cConnectParams XML node C–4
- cFileName XML node C–3, C–5
- cHandleName XML node C–5
- checkForDB procedure
 - Installation User Interface Manager 5–30
- Class cache E–21
- client_cache_directory session property
 - E–26
- cManagerName XML node C–3, C–5
- Combo caching E–6
- Comments 4–15
 - accessing 4–18
 - adding 4–16
 - auto-displayed 4–18
 - toolbar button 4–18
- Companies 1–6, 3–9
- Conditional parameters
 - using in DCU 5–19
- Configuration File Import tool 1–7
- Configuration File Manager 2–7
 - Connection Manager interface C–4
 - getPhysicalSessionType method C–3
 - getSessionParam method C–2
 - specifying 2–16
- Configuration files 2–7
 - generating 2–25
 - importing 2–31
 - minimum XML setting 2–27
- Configuration XML file 2–7
 - cConnectParams node C–4
 - cFileName node C–3, C–5
 - cHandleName node C–5
 - cManagerName node C–3, C–5
 - cPhysicalService node C–4
 - cServiceName node C–4
 - cServiceType node C–4
 - generating 2–25
 - importing 2–31
 - ICanRunLocalnode C–4
 - IDefaultService node C–4
 - manager node C–5
 - managers node C–5
 - Properties node C–2
 - service node C–4
 - Services node C–4
 - Sessions node C–2
 - usage and maintenance C–1
- Configurations with DataServers A–6
- connectDatabase procedure
 - Installation User Interface Manager 5–30
- Connecting to databases 2–5
- Connection Manager 2–7, C–4
- Connection parameters 2–9
- Consolidated groups 3–11
 - creating 3–16
- Context Help Control 1–9
- Country Control 1–3
- cPhysicalService XML node C–4
- Creating
 - groups based on users 3–17
 - security groups 3–15
 - session services 2–9
 - session types 2–17

- startup shortcut icon 2–28
- users 3–19
- Cross-version deployment
 - Dynamics tool support D–19
 - recommended procedure D–20
- Crystal Reports
 - Print Preview output 4–30
- cServiceName XML node C–4
- cServiceType XML node C–4
- CUI physical sessions C–3
- Currency Control 1–4
- Custom configuration properties C–4
- Customization
 - DCU pages 5–7
- Customized login window optimizations
 - E–43

D

- Data dump (.d) files
 - DCU stages 5–10
- Database
 - comments 4–15, 4–16
 - connecting 2–5
 - connection manager property C–2
 - stopping 2–6
- Databases
 - creating with DCU 5–38, 5–42
 - upgrading during deployment 5–43
 - version sequence 5–21
- DataServers
 - connecting A–3
 - development considerations A–6
 - Progress documentation A–7
 - Progress Dynamics support A–2
 - schema holder database A–6
 - two-phase commit A–7
 - white papers A–8

- Dataset (ADO) files
 - DCU stages 5–11
- Datasets 1–5
- DCU
 - adding pages 5–40
 - ADO files applied 5–11
 - APIs 5–27
 - available Managers 5–6
 - batch mode 5–47
 - See also* Batch-mode DCU
 - calling manager APIs 5–17
 - calling site data dump and load 5–64
 - capturing data 5–20
 - conditional parameters 5–19
 - configuration file 5–13
 - creating databases 5–42
 - custom database creation upgrade file
 - 5–43
 - custom driver files 5–39
 - custom session type 5–39
 - customization tips 5–45
 - customizing DCU sessions 5–35
 - customizing pages 5–7
 - Data dump files applied 5–10
 - database creation 5–23, 5–38
 - database version sequence 5–21
 - DCU sessions 5–5
 - deployment stages 5–9
 - description 5–3
 - driver files 5–15
 - Action node 5–17
 - ActionParam node 5–18
 - ActionTarget node 5–17
 - API calls 5–17
 - Control node 5–17
 - Database node 5–20
 - Event node 5–17
 - FINISH keyword 5–17
 - MinimumVersion node 5–22
 - Page node 5–16
 - Patch node 5–22
 - Path node 5–15
 - Proc node 5–16
 - QUIT keyword 5–17
 - RegistryKey node 5–15
 - StoreTo node 5–20
 - TableVariable node 5–20

- end conditions 5–12
 - end keywords 5–17
 - interface components 5–7
 - logging 5–11
 - necessary data files 5–35
 - order of upgrades 5–13
 - page descriptions 5–16
 - passing action parameters 5–18
 - patch levels 5–22
 - Phase 1 deployment 5–9
 - Phase 1 stages 5–10
 - Phase 2 deployment 5–9
 - processing sequence 5–12
 - release versioning tools 5–24
 - Schema definition files applied 5–10
 - session property impact 5–36
 - setting minimum version 5–22
 - specifying ADOs 5–24
 - standard pages 5–8
 - starting 5–46
 - template for pages 5–7
 - updating cached data 5–10
 - upgrade
 - retaining site-specific data 5–52
 - upgrade file types 5–22
 - upgrade files 5–22
 - ADO lists 5–24
 - database creation 5–23
 - patch files 5–25
 - upgrade program attributes 5–38
 - upgrade program design 5–36
 - upgrading existing databases 5–43
 - using events and actions 5–18
 - using multiple sessions 5–13
 - wizard interface 5–6
 - XML files 5–13
- dcubatch.bat file 5–51
- dcuphase1.p file 5–50
- dcuphase2.p file 5–50
- DCUSCRIPTFILE parameter
 - batch DCU Phase 1 5–50
 - generating batch data 5–48
- DCUSITEDATAFILE parameter
 - batch DCU Phase 1 5–50
 - generating batch data 5–48
- Default security group 3–16, 3–18
- Defining
 - managers 2–16
 - for a session type 2–19
 - properties
 - for a session type 2–21
 - services
 - for a session type 2–23
- Defining user categories 3–7
- Deploy Static Objects 1–5
- Deployment
 - adding pages to DCU 5–40
 - between Dynamics versions D–18
 - See also* Cross-version deployment
 - central Repository 5–3
 - custom database creation upgrade file 5–43
 - custom DCU session type 5–39
 - custom driver files 5–39
 - customizing DCU 5–35
 - database creation 5–23, 5–38, 5–42
 - database version sequence 5–21
 - Dataset Control 1–5
 - DCU APIs 5–27
 - DCU conditional parameters 5–19
 - DCU configuration file 5–13
 - DCU description 5–3
 - DCU driver files 5–15
 - DCU end conditions 5–12
 - DCU Phase 1 5–9
 - DCU Phase 2 5–9
 - DCU processing sequence 5–12
 - DCU stages 5–9
 - DCU tips 5–45
 - DCU upgrade files 5–22
 - Destinations 1–5
 - dynamic objects as static D–2
 - See also* Static-4GL equivalents of dynamic objects
 - general information D–1
 - impact on design 5–2
 - menu 1–5
 - necessary data files 5–35
 - order of upgrades 5–13
 - page descriptions 5–16

- patch files 5–25
- patch levels 5–22
- Phase 1 stages 5–10
- planning for 5–2
- release versioning tools 5–24
- sequences D–2
- session property impact 5–36
- setting minimum version 5–22
- specifying ADOs 5–24
- specifying databases 5–20
- starting the DCU 5–46
- updating cached data 5–10
- upgrade program attributes 5–38
- upgrade program design 5–36
- upgrading existing databases 5–43
- using multiple DCU sessions 5–13
- XML files 5–13

Design

- deployment impact 5–2

dumpconfig.txt file

- default 5–62
- field options 5–61

Dynamic

- call wrapper optimizations E–42
- lookup mapped fields E–12
- TreeView optimizations E–35

dynlaunch.i file E–42

E

Entity

- cache E–21
- Control 1–9
- Import 1–9
- mnemonic
 - GSCSC 3–2
 - GSCSQ D–2
 - GSMLG 3–9
 - GSMPPF 3–23
 - GSMUC 3–7
 - GSMUS 3–7
 - GSTPH 3–23

- translations 4–14
 - Migrate Widget Translation 4–15
 - Translation Maintenance 4–14

eventProc procedure

- Installation User Interface Manager 5–31

Exporting

- data through Print Preview 4–30

F

Filter Set Maintenance 1–9

FM handle code C–6

4GL Generator 1–6

- window D–3

- See also* Static-4GL equivalents of dynamic objects
- using in Administration tool D–7

G

Gapless sequences

- specifying 4–2

Generate

- Client Cache window 1–6, E–24
- Configuration File window 1–7

Generating

- configuration files 2–25
- data for batch-mode DCU 5–48

Generic

- auditing 4–19
- database comments 4–15, 4–16

getDBFile procedure

- Installation User Interface Manager 5–31

getDirectory procedure

- Installation User Interface Manager 5–31

getPhysicalSessionType method C–3

getSessionParam method C–2

- Global Control 1–8
- GM handle code C–6
- gotoPage procedure
 - Installation User Interface Manager 5–32
- Grant security model 3–5, 3–14
- Group processing 3–17
- Group-based security 3–11
- Groups 1–6
 - based on users 3–17
 - category 4–25
- gsc_logical_service table
 - cServiceName XML node C–4
 - ICanRunLocal XML node C–4
- gsc_manager_type table
 - cManagerName XML node C–5
- gsc_object table
 - cFileName XML node C–5
- gsc_security_control table 3–2
- gsc_service_type table
 - cManagerName XML node C–5
 - cServiceType XML node C–4
- gsm_login_company table 3–9
- gsm_physical_service table
 - cConnectParams XML node C–4
 - cPhysicalService XML node C–4
- gsm_profile_data table 3–23
- gsm_service_type table
 - IDefaultService XML node C–4
- gsm_user table 3–7
- gsm_user_category table 3–7
- gst_password_history table 3–23

- GUI
 - site data dump and load 5–63
- GUI physical sessions C–3

H

- Handles
 - static manager C–5
- History
 - password 3–23
- HTML
 - Print Preview output 4–30

I

- ICFCM_ configuration properties C–2
- ICFCONFIG parameter 2–28
- icfconfig.xml file 2–7
- ICFDB
 - database 2–5, 2–6
 - Install Manager 5–6
 - obtainICFSeqVals procedure 5–32
 - validateSiteNumber procedure 5–34
- ICFDev 2–2
 - default session 2–5
- ICFSSESSTYPE parameter 2–2, 2–28, 2–29
 - SessionType attribute value C–2
- Importing
 - configuration files 2–31
- Indexes
 - DataServer A–6
- Installation User Interface Manager 5–6
 - analyzeCase function 5–28
 - analyzeIf function 5–29
 - btnChoose procedure 5–29
 - checkForDB procedure 5–30
 - connectDatabase procedure 5–30
 - eventProc procedure 5–31

- getDBFile procedure 5–31
- getDirectory procedure 5–31
- gotoPage procedure 5–32
- obtainPatchList procedure 5–32
- processParams procedure 5–33
- restoreProperties procedure 5–33
- screenScrape procedure 5–33
- startUpgradeProcess procedure 5–34
- validateDirectory procedure 5–34
- verifyDBVersion procedure 5–35

Instance Attribute Control 1–3

Italic typeface
as typographical convention xiii

K

Keystrokes xiii

L

Language
Control 1–3, 4–4, 4–5, 4–7
Maintenance 4–5, 4–6, 4–7
translation tools 4–4

lCanRunLocal XML node C–4

lDefaultService XML node C–4

Least restrictive rule 3–11

Levels of categories 4–25

Linking
users and security groups 3–26

Localization
development 4–9
tools 4–4
user run-time 4–11
See also Translation.

Logging
DCU 5–11

Logical
service 2–9
creating 2–11
Service Control 1–8

Login
companies 1–6, 3–9
filename 3–3
linking to security groups 3–19

Lookup caching E–6

M

Management
global options 3–2

Manager
static handles and codes C–5
Type Control 1–8
XML node C–5

Managers
DCU sessions 5–6
defining 2–16
for a session type 2–19
ICFDB Install Manager 5–6
Installation User Interface Manager 5–6

Manual, organization of xi

MaxHiddenContainers session property
E–36

Menu
caching at startup E–29
translations 4–12

Message Control 1–9

Minimum XML files 2–27

Monospaced typeface
as typographical convention xiii

MS SQL Server
Progress DataServer for A–3

Multi Media

Image Control 1–4

Type

Maintenance window 4–24

Type Control 1–4

window 4–23

N

Nationality Control 1–3

NON handle code C–5

O

Objects, securing 3–1

obtainICFSeqVals procedure

ICFDB Install Manager 5–32

obtainPatchList procedure

Installation User Interface Manager 5–32

Oracle

Progress DataServer for A–5

Owning entities 4–25

P

Passwords

history 3–23

setting 3–22

unique 3–3

Performance optimization E–1

Class and entity cache E–21

architecture E–22

configuring E–26

framework usage E–22

generating E–24

usage guidelines E–26

customized login windows E–43

usage guidelines E–43

dynamic call wrapper E–42

usage guidelines E–43

dynamic lookup mapped fields E–12

linking directly to SDO fields E–14

mapping through viewer widgets E–17

setup and usage E–13

when not used E–21

Dynamic TreeView E–35

batching nodes E–37

keeping frames alive E–36

usage guidelines E–40

Lookup/combo cache E–6

clearing E–10

disabling for particular

SmartDataFields E–11

enabling and disabling E–9

operation E–7

when not used E–8

PicClip images E–27

used in custom toolbars E–28

SDOs kept alive on server E–5

usage guidelines E–6

SmartDataObject data definition and

schema location E–40

configuring field definition location

E–41

SchemaLocation attribute E–41

startup

cache for toolbars and menus E–29

configuring pre-caching

E–30

usage guidelines E–32

parameter settings E–3

static-4GL equivalents of dynamic

objects E–5

SuperProcedureMode attribute E–42

thin SmartObject rendering E–32

configuring E–34

usage guidelines E–35

Physical

service

creating 2–13

Service Control 1–8

session types defined C–3

session types supported C–3

Physical service 2–9

- physical_session_list configuration
 - property C-3
 - PicClip images E-27
 - PM handle code C-5
 - Print Preview
 - setting up 4-30
 - print_preview_preference session property 4-30
 - print_preview_stylesheet session property 4-30
 - processParams procedure
 - Installation User Interface Manager 5-33
 - Product Maintenance 1-3
 - Profile
 - code defined 2-32
 - Control 1-8
 - data
 - defining 2-32
 - in a session 2-32
 - defining user 2-33
 - defining users 3-23
 - types and session types 2-32
 - types defined 2-32
 - users 3-21
 - Progress Dynamics
 - application categories 4-25
 - application status 4-27
 - auditing and comments 4-15
 - basic security and management 3-2
 - DataServers A-2
 - See also* DataServers
 - defining profile data 2-32
 - multi-media types 4-23
 - profile data 2-32
 - sequences 4-2
 - user authentication 3-7
 - Progress Dynamics Configuration Utility,
 - See* DCU
 - PROPATH
 - batch-mode DCU 5-49
 - PROPATH setting C-2
 - Properties
 - creating session 2-14
 - custom configuration C-4
 - defining
 - for a session type 2-21
 - ICFCM_ configuration C-2
 - Node C-2
 - physical_session_list configuration C-3
 - run_local configuration C-3
 - session B-1
 - session_ configuration C-2
 - startup_procedure configuration C-2
 - valid_os_list configuration C-3
 - XML node C-2
- ## R
- Redundant ADO Listings 1-5
 - Registering users
 - steps 3-7
 - User Control 3-19
 - Release
 - Version Control 1-5
 - versioning
 - ADO lists in DCU 5-24
 - Repository
 - deployment 5-3
 - starting 2-5
 - See also* ICFDB.
 - Reset Data Modified Status 1-5
 - restoreProperties procedure
 - Installation User Interface Manager 5-33

Revoke security model 3–4, 3–13

RM handle code C–5

run_local configuration property C–3

S

Save Dynamic Object As Static D–3

See also Static-4GL equivalents of
dynamic objects
using in AppBuilder D–4

Schema

definition (.df) files
DCU stages 5–10
holder database A–6

SchemaLocation attribute E–41

screenScrape procedure

Installation User Interface Manager 5–33

Script

batch commands for DCU 5–49
sample DCU batch commands 5–51

Script file

batch-mode DCU 5–48

Security

allocations 3–1
basic settings 3–3
cascading information 3–24
clearing allocations 3–6
Control 1–6
Control Maintenance 3–2
defining groups 3–11
defining model 3–3
enabling 3–3
global options 3–2
grant vs. revoke model 3–4
groups 3–15, 3–16, 3–19
hierarchies 3–11
linking to login companies 3–19

maintenance 1–6

menu overview 1–6

override rule 3–12

turning off 3–3

types 2–10

user authentication 3–7

SEM handle code C–5

Sequence

Control 1–9, 4–2

Maintenance 4–3

Sequences 4–2

database version 5–21

deployment D–2

Service

Type Control 1–8

Type Manager 2–7

types 2–9

XML node C–4

Services

defining

for a session type 2–23

Session

configuring 2–8

creating services 2–9

Manager 2–7

menu overview 1–7

properties 2–2

alphabetical listing B–1

creating 2–14

impact on DCU 5–36

Property Control 1–8

required managers 2–2

service 2–9

service types 2–10

services 2–2

Type Control 1–7

Type Control Data tool 1–8

types

creating 2–17

descriptions 2–4

inheritance 2–3

- predefined 2–3
 - related to profile types 2–32
 - See also* Session type
 - XML node C–2
- SESSION system handle C–2
- Session type
 - customizing for DCU 5–39
 - defining
 - managers 2–19
 - properties 2–21
 - services 2–23
- session_ configuration properties C–2
- SessionType attribute C–2
- Set Site Number dialog box 1–9
- Setting users passwords 3–22
- Shortcut icon for startup 2–28
- Site Data Dump and Load utilities 5–52
 - architecture 5–53
 - calling from the DCU 5–64
 - coding processing directions 5–61
 - default dumpconfig.txt file 5–62
 - dump program
 - customized code 5–57
 - writing 5–56
 - GUI for manual execution 5–63
 - load program
 - customized code 5–60
 - writing 5–58
 - setting up 5–54
- Site data file
 - batch-mode DCU 5–48
- Site-specific data
 - batch-mode DCU 5–48
 - retaining for DCU updates 5–52
- SM handle code C–5
- SmartDataField caching E–6
- SmartDataObject data and schema
 - optimization E–40
- SmartDataObjects
 - kept alive on server E–5
- Starting Dynamics sessions 2–5
 - shortcut icon 2–28
- Startup
 - cache for toolbars and menus E–29
 - parameters
 - performance optimization E–3
- startup_procedure configuration property C–2
- StartupCacheClasses session property E–26
- StartupCacheMenusForObjects E–30
- StartupCacheToolbars session property E–30
- startUpgradeProcess procedure
 - Installation User Interface Manager 5–34
- Static-4GL equivalents of dynamic objects
 - 4GL Generator
 - hook procedure D–11
 - Options tab D–10
 - Paths tab D–11
 - Selection tab D–9
 - using in Administration tool D–7
 - comparing 4GL Generator with Save As Static D–4
 - generated-4GL objects
 - APIs for executing D–17
 - deployment D–14
 - development impact D–16
 - execution and file naming D–12
 - filename rules D–13
 - locating for execution D–14
 - minimum files to deploy D–15
 - when to deploy D–16
 - generation mechanisms D–3
 - using save as static D–4

Status

Control 1–4

window 4–28

Maintenance

window 4–29

tracking internals 4–28

Stopping databases 2–6

Subgroups

category 4–25

SuperProcedureMode attribute E–42

System menu 1–9

T

Thin SmartObject rendering E–32

TM handle code C–5

Toolbar and Menu Designer 1–3

Toolbar caching at startup E–29

Transaction menu

overview 1–10

Translate Window 4–11

Translation

Control 1–3, 4–9

entities 4–14

language 4–4

Maintenance 4–9

menus 4–12

testing 4–10

user run-time 4–11

widget labels 4–9

Two-phase commit

DataServer A–7

Types

category 4–25

of users 3–21

Typographical conventions xiii

U

Upgrade

file types 5–22

programs

attributes 5–38

DCU requirements 5–36

Upgrades

DCU end conditions 5–12

retaining site-specific data 5–52

User

authentication 3–7

categories 1–7, 3–7

linking to security groups 3–19

profile data 2–32

profile settings 2–32

profiles

defining 2–33

registering 3–19

status

Status application 4–27

Users 5–7

defining 1–7

UseThinRendering session property E–35

V

valid_os_list configuration property C–3

validateDirectory procedure

Installation User Interface Manager 5–34

ValidateSiteNumber procedure

ICFDB Install Manager 5–34

verifyDBVersion procedure

Installation User Interface Manager 5–35

W

WBC physical sessions C-3

WBS physical sessions C-3

Widget translations 4-9

WM handle code C-5

X

XML

DCU configuration file 5-13

DCU driver files 5-15

DCU files 5-13

DCU upgrade files 5-22

minimum XML files 2-27

Print Preview output 4-30

