

# Progress AppBuilder Developer's Guide

© 2001 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.

Progress, Progress Results, Provision and WebSpeed are registered trademarks of Progress Software Corporation in the United States and other countries. Apptivity, AppServer, ProVision Plus, SmartObjects, IntelliStream, and other Progress product names are trademarks of Progress Software Corporation.

SonicMQ is a trademark of Sonic Software Corporation in the United States and other countries.

Progress Software Corporation acknowledges the use of Raster Imaging Technology copyrighted by Snowbound Software 1993-1997 and the IBM XML Parser for Java Edition.

© IBM Corporation 1998-1999. All rights reserved. U.S. Government Users Restricted Rights — Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Progress is a registered trademark of Progress Software Corporation and is used by IBM Corporation in the mark Progress/400 under license. Progress/400 AND 400® are trademarks of IBM Corporation and are used by Progress Software Corporation under license.

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

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

.

May 2001




Product Code: 4538
Item Number: 81106;9.1C

# Contents


**Preface** . . . . . **xvi**
Purpose . . . . . xvi
Audience . . . . . xvi
Organization of This Manual . . . . . xvi
How to Use This Manual . . . . . xviii
Typographical Conventions . . . . . xviii
Syntax Notation . . . . . xix
Progress Messages . . . . . xxiii
Other Useful Documentation . . . . . xxiv
Getting Started . . . . . xxiv
Development Tools . . . . . xxvi
Reporting Tools . . . . . xxvii
4GL . . . . . xxvii
Database . . . . . xxviii
DataServers . . . . . xxviii
SQL-89/Open Access . . . . . xxix
SQL-92 . . . . . xxix
Deployment . . . . . xxx
WebSpeed . . . . . xxx
Reference . . . . . xxxi
SQL-92 Reference . . . . . xxxi

**1. Introduction** . . . . . **1–1**
1.1 Progress Application Development Model . . . . . 1–3
1.2 Concepts and Terminology . . . . . 1–4
1.2.1 External Versus Internal Procedures . . . . . 1–4
1.2.2 AppBuilder Objects . . . . . 1–4
1.2.3 What Is Smart About SmartObjects? . . . . . 1–11
1.2.4 Visible Versus Nonvisible Objects . . . . . 1–12

1.3 Widgets . . . . . 1–13
1.4 Reading and Writing 4GL Widgets . . . . . 1–14
1.5 Inter-object Communication: Signatures . . . . . 1–15
1.6 Anatomy of a Database Application . . . . . 1–15
1.7 Working with AppBuilder . . . . . 1–16
1.8 Single-window or Multi-window Design? . . . . . 1–17
1.9 Designing for Character-based Environments . . . . . 1–17
1.10 Building an Application in AppBuilder . . . . . 1–18

**2. AppBuilder Interface . . . . . 2–1**
2.1 Starting and Stopping AppBuilder . . . . . 2–2
2.1.1 AppBuilder for Web-based Applications . . . . . 2–2
2.2 AppBuilder Main Window . . . . . 2–3
2.3 Menu Bar . . . . . 2–4
2.3.1 File Menu . . . . . 2–5
2.3.2 Edit Menu . . . . . 2–7
2.3.3 Compile Menu . . . . . 2–9
2.3.4 Tools Menu . . . . . 2–10
2.3.5 Options Menu . . . . . 2–11
2.3.6 Layout Menu . . . . . 2–12
2.3.7 Window Menu . . . . . 2–14
2.3.8 Help Menu . . . . . 2–15
2.3.9 Tool Bar . . . . . 2–16
2.4 Object Tools Palette . . . . . 2–17
2.4.1 Placing Objects from the Object Palette . . . . . 2–22
2.4.2 Object Palette Options . . . . . 2–23
2.4.3 Customizing the Object Palette . . . . . 2–23
2.5 Setting Your Preferences . . . . . 2–23
2.5.1 AppBuilder Preference Options . . . . . 2–24
2.5.2 Section Editor Preference Options . . . . . 2–28
2.6 Workspaces: Design Windows and Source-code Editors . . . . . 2–35
2.7 Editing Source Code Safely: The Section Editor . . . . . 2–37
2.7.1 Opening the Section Editor . . . . . 2–37
2.7.2 Choosing a Section to Edit . . . . . 2–38
2.7.3 Using the Section Editor . . . . . 2–39
2.7.4 Inserting Text-string Tokens from Lists . . . . . 2–43
2.8 Configuring Look-and-Feel: Object Properties . . . . . 2–52
2.9 Inspecting a Procedure File . . . . . 2–53
2.10 Test-driving Your Design . . . . . 2–55
2.11 Saving Your Work . . . . . 2–55

**3. Organizer Objects . . . . . 3–1**
3.1 Introduction to Organizer Objects . . . . . 3–2
3.2 Organizer Object Roles . . . . . 3–3

3.3 Creating a New Procedure-based Organizer Instance . . . . . 3–4
3.4 SmartWindows . . . . . 3–5
3.4.1 Configuring SmartWindow Visual Properties . . . . . 3–6
3.5 SmartDialogs . . . . . 3–15
3.5.1 Configuring SmartDialog Visual Properties . . . . . 3–16
3.5.2 Configuring SmartDialog Procedure Settings . . . . . 3–18
3.6 SmartFrames . . . . . 3–19
3.6.1 Configuring SmartFrame Visual Properties . . . . . 3–19
3.6.2 Configuring SmartFrame Procedure Settings . . . . . 3–22
3.7 Basic Windows . . . . . 3–22
3.8 Basic Dialogs . . . . . 3–22
3.9 Basic Frames . . . . . 3–22
3.9.1 Creating and Placing a Frame . . . . . 3–23
3.9.2 Configuring Frame Properties . . . . . 3–23
3.10 Rectangles . . . . . 3–23
3.10.1 Creating and Placing a Rectangle . . . . . 3–24
3.10.2 Configuring Rectangle Properties . . . . . 3–24
3.11 Simple SmartContainers . . . . . 3–27
3.11.1 Creating and Populating a Simple SmartContainer . . . . . 3–28
3.11.2 Placing and Configuring a SmartContainer Module . . . . . 3–29
3.12 Creating Paged Layouts . . . . . 3–31
3.12.1 Page Identifiers . . . . . 3–32
3.12.2 Special Pages . . . . . 3–32
3.12.3 Why Only SmartObjects? . . . . . 3–34
3.13 Paging with SmartFolders . . . . . 3–34
3.14 Paging Without SmartFolders . . . . . 3–39
3.15 Paging Whole Windows . . . . . 3–41

**4. Data-access Objects . . . . . 4–1**
4.1 Introduction to Data-access Objects . . . . . 4–2
4.2 SmartDataObjects . . . . . 4–2
4.2.1 Static SmartDataObjects . . . . . 4–2
4.2.2 Dynamic SmartDataObjects . . . . . 4–3
4.2.3 Creating a SmartDataObject Master . . . . . 4–4
4.2.4 Editing a SmartDataObject Master . . . . . 4–8
4.2.5 Creating a SmartDataObject Instance . . . . . 4–9
4.2.6 Configuring a SmartDataObject Instance . . . . . 4–11
4.2.7 Data Validation Limitation . . . . . 4–13
4.2.8 Using the Dynamic SmartDataObject . . . . . 4–15
4.3 SmartBusinessObject . . . . . 4–16
4.3.1 Creating and Placing a SmartBusinessObject . . . . . 4–18
4.3.2 Populating and Configuring a SmartBusinessObject . . . . . 4–19
4.3.3 Synchronizing SmartDataObjects Using DATA Links . . . . . 4–20
4.3.4 Configuring a SmartBusinessObject Instance . . . . . 4–25

4.3.5 How the CascadeOnBrowse Property Works . . . . . 4–26
4.4 SmartFilters . . . . . 4–27
4.4.1 Configuring SmartFilter Properties . . . . . 4–29
4.5 Making Changes to the Data Stream . . . . . 4–33
4.6 Creating a Freestanding Query Object . . . . . 4–34
4.7 Defining and Editing Queries Using Query Builder . . . . . 4–35
4.7.1 Table Mode . . . . . 4–36
4.7.2 Join Mode . . . . . 4–38
4.7.3 Where Mode . . . . . 4–39
4.7.4 Sort Mode . . . . . 4–40
4.7.5 Options Mode . . . . . 4–41
4.8 Selecting Database Fields for Browsing . . . . . 4–42
4.8.1 Creating Calculated Fields . . . . . 4–45

**5. Data-display/Capture Objects . . . . . 5–1**
5.1 Introduction to Data-display Objects . . . . . 5–2
5.2 SmartDataBrowsers . . . . . 5–3
5.3 Dynamic SmartDataBrowser . . . . . 5–3
5.3.1 Creating a Dynamic SmartDataBrowser Instance. . . . . 5–3
5.4 Static SmartDataBrowser . . . . . 5–6
5.4.1 Creating a SmartDataBrowser Master . . . . . 5–6
5.4.2 Configuring SmartDataBrowser Properties . . . . . 5–10
5.4.3 Editing a SmartDataBrowser Master. . . . . 5–16
5.4.4 Creating and Placing a SmartDataBrowser Instance . . . . . 5–18
5.4.5 Configuring a SmartDataBrowser Instance. . . . . 5–18
5.4.6 Scrolling to the Last Record in the Data Stream . . . . . 5–19
5.5 SmartDataViewers . . . . . 5–21
5.5.1 Creating a SmartDataViewer Master . . . . . 5–22
5.5.2 Adding and Removing SmartDataViewer Fields . . . . . 5–27
5.5.3 Configuring SmartDataViewer Properties . . . . . 5–28
5.6 SmartDataFields . . . . . 5–34
5.6.1 Placing and Configuring a SmartDataField Instance . . . . . 5–35
5.7 SmartSelects . . . . . 5–36
5.7.1 Placing a SmartSelect Instance . . . . . 5–37
5.7.2 Configuring a SmartSelect Instance . . . . . 5–38
5.7.3 Example: Assigning a New Sales Rep. . . . . 5–41
5.8 Basic Data Browsers . . . . . 5–44
5.8.1 Creating a Basic Data Browser Instance . . . . . 5–44
5.8.2 Configuring a Basic Data Browser Instance . . . . . 5–45
5.9 Combo Boxes . . . . . 5–50
5.9.1 Creating a Basic Combo Box Instance . . . . . 5–50
5.9.2 Configuring a Basic Combo Box Instance . . . . . 5–51
5.9.3 Operating a Basic Combo Box . . . . . 5–55
5.9.4 Creating an ActiveX Combo Box Instance . . . . . 5–59

5.9.5 Configuring an ActiveX Combo Box Instance . . . . . 5–59
5.10 Basic Data Viewers . . . . . 5–62
5.10.1 Creating a Basic Data Viewer Instance . . . . . 5–63
5.10.2 Configuring a Basic Viewer Instance . . . . . 5–63
5.11 Editors (Multi-line) . . . . . 5–64
5.11.1 Creating an Editor Instance . . . . . 5–64
5.11.2 Configuring an Editor Instance . . . . . 5–64
5.12 Fill–ins (Single-line Editors) . . . . . 5–67
5.12.1 Creating a Fill–in Instance . . . . . 5–68
5.12.2 Configuring a Fill–in Instance . . . . . 5–68
5.13 Radio-button Sets . . . . . 5–72
5.13.1 Creating a Radio-set Instance . . . . . 5–72
5.13.2 Configuring a Radio-set Instance . . . . . 5–73
5.14 Selection Lists . . . . . 5–75
5.14.1 Creating a Selection–List Instance . . . . . 5–76
5.14.2 Configuring a Selection–List Instance . . . . . 5–76
5.15 Slider Controls . . . . . 5–79
5.15.1 Creating a Slider-Control Instance . . . . . 5–79
5.15.2 Configuring a Slider-Control Instance . . . . . 5–80
5.16 Spin Controls (ActiveX) . . . . . 5–82
5.16.1 Creating a Spin-Control Instance . . . . . 5–83
5.16.2 Configuring a Spin-Control Instance . . . . . 5–83
5.17 Toggle Boxes (Check Boxes) . . . . . 5–85
5.17.1 Creating a Toggle–Box Instance . . . . . 5–85
5.17.2 Configuring a Toggle–Box Instance . . . . . 5–86

**6. Data-communication Objects . . . . . 6–1**
6.1 SmartB2BObjects . . . . . 6–2
6.1.1 XML Schema and Mapping Files . . . . . 6–3
6.1.2 Mapping Options . . . . . 6–5
6.1.3 Progress XML Schema Mapping Tool . . . . . 6–9
6.1.4 Starting the XML Mapping Tool . . . . . 6–11
6.1.5 Creating an XML Map File . . . . . 6–11
6.1.6 Creating a Master SmartB2BObject . . . . . 6–16
6.1.7 Creating and Placing a SmartB2BObject Instance . . . . . 6–17
6.1.8 Configuring a SmartB2BObject Instance . . . . . 6–18
6.2 SmartSender and SmartReceiver Objects . . . . . 6–22
6.2.1 Creating and Completing a SmartSender Master . . . . . 6–23
6.2.2 Placing and Configuring a SmartSender Instance . . . . . 6–24
6.3 SmartRouters . . . . . 6–26
6.3.1 Handling Multiple Document Types . . . . . 6–27
6.3.2 Callable Document-handlers . . . . . 6–29
6.3.3 Creating a Document-handling Package . . . . . 6–29
6.3.4 Creating a Document-routing Package . . . . . 6–30

6.4 SmartProducers and SmartConsumers . . . . . 6–32
6.4.1 Creating and Placing a SmartProducer Instance . . . . . 6–33
6.4.2 Configuring a SmartProducer Instance . . . . . 6–34
6.4.3 Creating and Placing a SmartConsumer Instance . . . . . 6–36
6.4.4 Configuring a SmartConsumer Instance . . . . . 6–37
6.4.5 Batch Mode: Starting and Stopping a SmartConsumer . . . . . 6–39
6.5 The Message-handling Relationship: Internal Details . . . . . 6–40
6.5.1 Example of sendHandler Code . . . . . 6–43
6.5.2 Example of sendHandler Override Code . . . . . 6–44
6.5.3 Example of receiveReplyHandler Code . . . . . 6–44
6.5.4 Example of receiveHandler Code . . . . . 6–47
6.5.5 Example of receiveHandler Override Code . . . . . 6–49
6.5.6 Example of sendReplyHandler Code . . . . . 6–49
6.5.7 Example ofsendReplyHandler Override Code . . . . . 6–50

**7. Control Objects . . . . . 7–1**
7.1 Introduction to Control Objects . . . . . 7–2
7.2 SmartToolbars . . . . . 7–2
7.2.1 Placing a SmartToolbar . . . . . 7–3
7.2.2 Configuring a SmartToolbar Instance . . . . . 7–4
7.2.3 Creating a New SmartToolbar . . . . . 7–5
7.2.4 Adding a Filter-control Window . . . . . 7–5
7.3 SmartPanels . . . . . 7–8
7.3.1 Creating and Configuring a SmartPanel Instance . . . . . 7–9
7.4 Basic Pushbuttons . . . . . 7–12
7.4.1 Creating a Pushbutton Instance . . . . . 7–13
7.4.2 Configuring a Pushbutton Instance . . . . . 7–13

**8. Other Objects . . . . . 8–1**
8.1 Simple (Undefined) SmartObjects . . . . . 8–2
8.2 ActiveX (OCX) Objects . . . . . 8–2
8.2.1 Creating an ActiveX Object Instance . . . . . 8–3
8.2.2 Configuring an ActiveX Object Instance . . . . . 8–4
8.2.3 Generic ActiveX Event Handlers . . . . . 8–9
8.2.4 ActiveX (OCX) Properties Data . . . . . 8–10
8.3 Static Images . . . . . 8–10
8.4 Static Text . . . . . 8–14
8.5 Timer Objects (ActiveX) . . . . . 8–15

**A. Frequently Used Dialogs . . . . . A–1**
A.1 ActiveX Control Frame Properties . . . . . A–3
A.2 ADM Pages Manager . . . . . A–4
A.3 Advanced Procedure Settings . . . . . A–7
A.4 Advanced Properties . . . . . A–9

A.5 Colors Selector and Editor . . . . . A–10
A.5.1 Choosing Colors . . . . . A–11
A.5.2 Defining New Colors . . . . . A–12
A.6 Custom Lists Naming . . . . . A–15
A.7 Database Field Selector . . . . . A–16
A.8 Method Library Includes List . . . . . A–18
A.9 Procedure Settings . . . . . A–19
A.10 Properties Window . . . . . A–21
A.11 SmartLinks Editor . . . . . A–22
A.12 SmartInfo . . . . . A–25
A.13 SmartObject Properties . . . . . A–26
A.13.1 Creating a SmartObject Placeholder . . . . . A–28
A.14 String Attributes . . . . . A–28
A.15 Traversal-Path Editor . . . . . A–30
A.16 Temp-Tables Maintenance . . . . . A–31
A.17 Text Style Selector and Editor . . . . . A–34
A.17.1 Choosing a Typestyle . . . . . A–34
A.17.2 Defining a Typestyle . . . . . A–35

**B. Multiple Layouts . . . . . B–1**
B.1 Why Use Multiple Layouts? . . . . . B–2
B.2 What Is a Layout? . . . . . B–2
B.3 Two Types of Layouts . . . . . B–3
B.3.1 Master Layout . . . . . B–3
B.3.2 Alternate Layout . . . . . B–4
B.4 Creating Alternate Layouts . . . . . B–7
B.5 Selecting a Predefined Alternate Layout . . . . . B–8
B.6 Modifying Layouts–Layout Inheritance . . . . . B–8
B.6.1 Layout Inheritance Behavior at Run Time . . . . . B–9
B.6.2 Layout Inheritance Behavior at Design Time . . . . . B–10
B.6.3 Re-establishing Layout Inheritance . . . . . B–10
B.7 How Multiple Layouts Work: A Look at the Code . . . . . B–11
B.7.1 Run-time Properties and AppBuilder Settings . . . . . B–12
B.7.2 Layout Cases Internal Procedure . . . . . B–14
B.7.3 Enable_UI Internal Procedure . . . . . B–15
B.8 Alternate Layout Limitations . . . . . B–16
B.8.1 Modifiable Properties . . . . . B–16
B.8.2 Text Widgets in Alternate Layouts . . . . . B–17
B.8.3 Combo Boxes in Alternate Layouts . . . . . B–17
B.8.4 SIZE–TO–FIT . . . . . B–17
B.9 Switching Between Layouts at Run Time . . . . . B–17
B.10 Simulating Character Applications . . . . . B–18
B.11 Running Character Applications . . . . . B–19
B.11.1 Character Client Connection Issues . . . . . B–20

B.11.2 Double-byte Characters . . . . . B–20
B.12 Multiple Layouts for SmartObjects . . . . . B–21
B.12.1 Creating SmartObject Alternate Layouts . . . . . B–21
B.12.2 Modifying SmartObject Layouts . . . . . B–22
B.12.3 Applying SmartObject Layouts at Run Time . . . . . B–27

**C. Customizing AppBuilder . . . . . C–1**
C.1 Custom Object Files and Extended Features . . . . . C–2
C.2 Creating Custom Object Files . . . . . C–2
C.2.1 General Coding Rules . . . . . C–3
C.2.2 Entry Types . . . . . C–4
C.2.3 Adding a Button to the Objects Palette . . . . . C–4
C.2.4 Adding a SmartObject to a Palette Button Pop-up Menu . . . . . C–6
C.2.5 Adding a New Template . . . . . C–8
C.2.6 Adding a Widget to a Palette Button Pop-up Menu . . . . . C–9
C.3 Creating Extended Features (XFTRs) . . . . . C–19
C.3.1 The &ANALYZE-SUSPEND Directive for XFTRs . . . . . C–21
C.3.2 The Event-mapping Comment . . . . . C–21
C.3.3 The Compilable Code Comment . . . . . C–22
C.3.4 The Ending Comment . . . . . C–22
C.3.5 The &ANALYZE-RESUME Directive for XFTRs . . . . . C–23
C.3.6 How AppBuilder Runs XFTR Event Handlers . . . . . C–23
C.4 AppBuilder's API . . . . . C–23
C.4.1 Context IDs . . . . . C–23
C.4.2 Naming Convention . . . . . C–24
C.4.3 AppBuilder's API Summary . . . . . C–24
C.4.4 Add/Remove SmartLinks (adeuib/_ablink.p) . . . . . C–25
C.4.5 Access Section (adeuib/_accsect.p) . . . . . C–27
C.4.6 AppBuilder Database Table Names (adeuib/_dbtbnam.p) . . . C–32
C.4.7 AppBuilder Information (adeuib/_uibinfo.p) . . . . . C–33
C.4.8 AppBuilder Create (adeuib/_uib_crt.p) . . . . . C–40
C.4.9 AppBuilder Delete (adeuib/_uib_del.p) . . . . . C–42
C.4.10 AppBuilder Dialog (adeuib/_uib_dlg.p) . . . . . C–43
C.4.11 Window Save (adeuib/_winsave.p) . . . . . C–44
C.4.12 Reserved Names (adeuib/_namespc.p) . . . . . C–45
C.5 Example Using AppBuilder's API . . . . . C–47

**Glossary . . . . . Gloss–1**

**Index . . . . . Index–1**

**Figures**


Figure 1–1: Design-time Representation of a Nonvisible SmartObject . . . . . 1–13
Figure 1–2: SmartObject Menu-button Menu . . . . . 1–13
Figure 1–3: Toy Phonebook Application at Design Time . . . . . 1–18
Figure 1–4: Toy Phonebook Application at Run Time . . . . . 1–19
Figure 1–5: SmartLinks Between SmartObjects . . . . . 1–21
Figure 1–6: Toy Phonebook Links in SmartLinks Editor . . . . . 1–22
Figure 2–1: AppBuilder Main Window . . . . . 2–3
Figure 2–2: Current-Object Fields . . . . . 2–3
Figure 2–3: Status Bar . . . . . 2–4
Figure 2–4: AppBuilder Tool Bar . . . . . 2–16
Figure 2–5: Object Tools Palette . . . . . 2–17
Figure 2–6: Workspace for a Visible Object . . . . . 2–36
Figure 2–7: Workspace Element for Nonvisible Object . . . . . 2–36
Figure 2–8: Section Editor Physical Interface . . . . . 2–37
Figure 2–9: Sections Listing Dialog Box . . . . . 2–38
Figure 2–10: Find Dialog Box . . . . . 2–39
Figure 2–11: Replace Dialog Box . . . . . 2–40
Figure 2–12: Field Selector Dialog Box . . . . . 2–43
Figure 2–13: Choose Event Dialog Box . . . . . 2–45
Figure 2–14: Insert Procedure Call Dialog Box . . . . . 2–46
Figure 2–15: Listing Insertable Procedures . . . . . 2–48
Figure 2–16: Preprocessor Names Dialog Box . . . . . 2–49
Figure 2–17: Query Builder Dialog Box . . . . . 2–50
Figure 2–18: Object Names Dialog Box . . . . . 2–51
Figure 2–19: Choose Insert File Name Dialog Box . . . . . 2–52
Figure 2–20: Code Preview Dialog Box . . . . . 2–54
Figure 2–21: Stop Button . . . . . 2–55
Figure 3–1: Window Property Sheet . . . . . 3–8
Figure 3–2: Window Advanced Properties Dialog Box . . . . . 3–14
Figure 3–3: SmartDialog Property Sheet . . . . . 3–16
Figure 3–4: SmartDialog Advanced Properties . . . . . 3–18
Figure 3–5: SmartFrame Property Sheet . . . . . 3–19
Figure 3–6: SmartFrame Advanced Properties Dialog Box . . . . . 3–21
Figure 3–7: Rectangle Property Sheet . . . . . 3–25
Figure 3–8: Rectangle Advanced Properties Dialog Box . . . . . 3–27
Figure 3–9: Paging Using a SmartFolder . . . . . 3–35
Figure 3–10: Paging Using Pushbutton Controls . . . . . 3–39
Figure 3–11: Paging Using Radio-set Controls . . . . . 3–39
Figure 3–12: Paging a Whole Window . . . . . 3–41
Figure 4–1: Compilation of a SmartDataObject . . . . . 4–3
Figure 4–2: SmartBusinessObject in a Web Environment . . . . . 4–16
Figure 4–3: Data Relationship Among Five Tables . . . . . 4–17
Figure 4–4: Resynchronization Hierarchy Defined by DATA SmartLinks . . . . . 4–21

Figure 4–5: Query Builder in Table Mode . . . . . 4–36
Figure 4–6: Sample Query for OF-joinable Tables . . . . . 4–37
Figure 4–7: Query Sample with Incomplete Join Criteria . . . . . 4–37
Figure 4–8: Query Builder in Join Mode . . . . . 4–38
Figure 4–9: Query Builder in Where Mode . . . . . 4–39
Figure 4–10: Query Builder in Sort Mode . . . . . 4–40
Figure 4–11: Query Builder in Options Mode . . . . . 4–41
Figure 4–12: Column Editor Button in Query Builder . . . . . 4–42
Figure 4–13: Column Editor Dialog Box . . . . . 4–43
Figure 6–1: SmartB2BObject in Inter-business Communications . . . . . 6–2
Figure 6–2: Simple One-to-One XML Mapping in Data Communication . . . . . 6–4
Figure 6–3: Mapping an XML Node to a Function . . . . . 6–5
Figure 6–4: XML Mapping Tool Window . . . . . 6–10
Figure 6–5: The SmartSender and SmartReceiver in B2B Communication . . . . . 6–22
Figure 6–6: SmartRouter Distributing Incoming Documents . . . . . 6–26
Figure 6–7: Basic Message-handling Blocks . . . . . 6–27
Figure 6–8: Message Routing Using SmartRouter . . . . . 6–28
Figure 6–9: The Smart Message-interface Objects . . . . . 6–32
Figure 6–10: Outbound Message-handling Process . . . . . 6–41
Figure 6–11: Inbound Message-handling Process . . . . . 6–42
Figure 7–1: SmartToolbar Fully Enabled . . . . . 7–3
Figure 7–2: SmartToolbar in Record-changed State . . . . . 7–3
Figure 7–3: SmartToolbar in Transaction-pending State . . . . . 7–3
Figure A–1: Pages Dialog Box . . . . . A–5
Figure A–2: Advanced Procedure Settings Dialog Box . . . . . A–7
Figure A–3: Custom Lists Naming Dialog Box . . . . . A–16
Figure A–4: SmartWindow Main Procedure Settings Dialog Box . . . . . A–20
Figure A–5: Properties Window . . . . . A–21
Figure A–6: SmartLinks Editor Dialog Box . . . . . A–23
Figure A–7: String Attributes Dialog Box for Static Text . . . . . A–29
Figure A–8: String Attributes Dialog Box for a Combo Box . . . . . A–29
Figure A–9: Tab Editor Dialog Box . . . . . A–30
Figure B–1: Master Layout . . . . . B–5
Figure B–2: Character Alternate Layout . . . . . B–6
Figure B–3: Alternate Layout Dialog Box for SmartObjects . . . . . B–22
Figure B–4: SmartDataViewer Master Layout . . . . . B–23
Figure B–5: SmartDataViewer Address Alternate Layout . . . . . B–23
Figure B–6: SmartDataViewer Balance Alternate Layout . . . . . B–23
Figure B–7: SmartDataViewer Menu Button’s Pop-up Menu . . . . . B–24
Figure B–8: Menu Button’s Pop-up Menu in Alternate Layout . . . . . B–25
Figure B–9: Instance Property Sheet in Alternate Layout . . . . . B–26
Figure B–10: SmartDataViewer Properties with Layout Options . . . . . B–27
Figure C–1: Sample Entry for SmartObject Type . . . . . C–6
Figure C–2: Sample Entries for Adding a SmartObject Master . . . . . C–7
Figure C–3: Sample Entries for New Templates . . . . . C–9

Figure C–4: XFTR in browser.w Template . . . . . C–20
Figure C–5: Event Handler Components . . . . . C–21

**Tables**


Table 1–1: Basic 4GL Object Types . . . . . 1–5
Table 1–2: SmartObject Types . . . . . 1–8
Table 1–3: ActiveX (OCX) Object Types . . . . . 1–11
Table 1–4: Benefits of Using SmartObjects . . . . . 1–12
Table 2–1: File Menu . . . . . 2–5
Table 2–2: Edit Menu . . . . . 2–7
Table 2–3: Compile Menu . . . . . 2–9
Table 2–4: Tools Menu . . . . . 2–10
Table 2–5: Options Menu . . . . . 2–12
Table 2–6: Layout Menu . . . . . 2–12
Table 2–7: Window Menu . . . . . 2–14
Table 2–8: Help Menu . . . . . 2–15
Table 2–9: Options on the Insert Procedure Call Dialog Box . . . . . 2–47
Table 4–1: How to Make Changes to the Data Stream . . . . . 4–33
Table 5–1: Dynamic SmartDataBrowser Display/Update Rules . . . . . 5–5
Table A–1: Progress Color Table . . . . . A–12
Table B–1: Standard Layouts and Run-time Expressions . . . . . B–4
Table B–2: Modifiable Properties in Alternate Layouts . . . . . B–16
Table C–1: Custom Object Files and Their Contents . . . . . C–3
Table C–2: Settable Attributes for Custom Widgets . . . . . C–11
Table C–3: AppBuilder Events . . . . . C–22
Table C–4: Naming Conventions . . . . . C–24
Table C–5: API Procedures . . . . . C–24

# Preface

## Purpose

This guide provides a basic introduction to using AppBuilder for Rapid Application Prototyping/Development (RAP/RAD).

## Audience

Database programmers who appreciate the time and energy advantages of using WYSIWYG layout editors for RAP/RAD. This guide presumes that you understand programming in general, though it does not require that you know the Progress 4GL specifically. You can create simple applications using AppBuilder alone, but you will find it hard to do sophisticated work without at least some programming ability with the 4GL.

## Organization of This Manual

Chapter 1, "Introduction"

Introduces basic terminology and concepts that are important to understand when working with AppBuilder.

Chapter 2, "AppBuilder Interface"

Walks through the user-interface elements apart from the common dialog boxes, which are collected together in Appendix A.

Chapter 3, "Organizer Objects"

Describes objects that integrate smaller objects functionally or visually. Organizer objects include Windows, Dialog Boxes, Frames, and even the Rectangle.

Chapter 4, "Data-access Objects""

Describes objects that help read and write the logical database. These range from the basic Query object through the powerful SmartBusinessObject.

Chapter 5, "Data-display/Capture Objects"

Describes the many objects that you can use to present the user with information from the data stream, and capture both changes and new data. They range from powerful SmartObjects to the simplest boolean Toggle Box.

Chapter 6, "Data-communication Objects"

Describes the transformation and communication objects you can use to create, send, receive, and process data messages.

Chapter 7, "Control Objects"

Describes the few objects that provide a nearly-pure control function: SmartToolbar, SmartPanels, and the lowly Pushbutton.

Chapter 8, "Other Objects"

Describes those few objects that seem not to fit well elsewhere: Static Text and Image objects, for example.

Appendix A, "Frequently Used Dialogs"

Describes the standard dialogs that you will encounter over and over again, nearly unchanged regardless of context.

Appendix B, "Multiple Layouts"

Describes how to create, from a single code base, layouts that vary in appearance across platforms and contexts.

Appendix C, "Customizing AppBuilder"

Describes ways you can change AppBuilder itself so that it will more closely meet your needs.

"Glossary"

## How to Use This Manual

Unless you are familiar with layout or graphics editors, begin by browsing Chapter 1, "Introduction," and studying Chapter 2, "AppBuilder Interface." Then experiment with some trial work of your own. Use the material from Chapter 3, "Organizer Objects," onward mainly as a convenient reference.

## 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
- *Italic typeface* indicates:
  - Progress variable information that the user supplies
  - New terms
  - Titles of complete publications
- `Monospaced typeface` indicates:
  - Code examples
  - System output
  - Operating system filenames and pathnames

The following typographical conventions are used to represent keystrokes:

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

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

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

  **CTRL-X**

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

  **ESCAPE H**
  **ESCAPE CURSOR-LEFT**

## Syntax Notation

The syntax for each component follows a set of conventions:

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

  In this example, ACCUM is a keyword:

  **SYNTAX**

  ```
  ACCUM aggregate expression
  ```

- Italics identify options or arguments that you must supply. These options can be defined as part of the syntax or in a separate syntax identified by the name in italics. In the ACCUM function above, the *aggregate* and *expression* options are defined with the syntax for the ACCUM function in the *Progress Language Reference*.

- You must end all statements (except for DO, FOR, FUNCTION, PROCEDURE, and REPEAT) with a period. DO, FOR, FUNCTION, PROCEDURE, and REPEAT statements can end with either a period or a colon, as in this example:

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

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

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

  **SYNTAX**

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

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

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

  **SYNTAX**

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

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

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

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

  **SYNTAX**

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

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

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

  **SYNTAX**

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

- A vertical bar ( | ) indicates a choice.

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

  **SYNTAX**

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

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

  **SYNTAX**

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

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

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

  **SYNTAX**

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

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

  **SYNTAX**

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

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

  **SYNTAX**

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

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

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

  **SYNTAX**

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

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

  **SYNTAX**

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

## Progress Messages

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

- Execution messages inform you of errors encountered while Progress is running a procedure (for example, if Progress cannot find a record with a specified index field value).
- Compile messages inform you of errors found while Progress is reading and analyzing a procedure prior to running it (for example, if a procedure references a table name that is not defined in the database).
- Startup messages inform you of unusual conditions detected while Progress is getting ready to execute (for example, if you entered an invalid startup parameter).

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

- Continues execution, subject to the error-processing actions that you specify, or that are assumed, as part of the procedure. This is the most common action taken following execution messages.
- Returns to the Progress Procedure Editor so that you can correct an error in a procedure. This is the usual action taken following compiler messages.
- Halts processing of a procedure and returns immediately to the Procedure Editor. This does not happen often.
- Terminates the current session.

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

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

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

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

## Other Useful Documentation

This section lists Progress Software Corporation documentation that you might find useful. Unless otherwise specified, these manuals support both Windows **and** Character platforms and are provided in electronic documentation format on CD-ROM.

### Getting Started

*Progress Electronic Documentation Installation and Configuration Guide* (Hard copy only)

A booklet that describes how to install the Progress EDOC viewer and collection on UNIX and Windows.

*Progress Installation and Configuration Guide Version 9 for UNIX*

A manual that describes how to install and set up Progress Version 9.1 for the UNIX operating system.

*Progress Installation and Configuration Guide Version 9 for Windows*

A manual that describes how to install and set up Progress Version 9.1 for all supported Windows and Citrix MetaFrame operating systems.

*Progress Version 9 Product Update Bulletin*

A guide that provides a brief description of each new feature of the release. The booklet also explains where to find more detailed information in the documentation set about each new feature.

*Progress Application Development Environment — Getting Started* (Windows only)

A practical guide to graphical application development within the Progress Application Development Environment (ADE). This guide includes an overview of the ADE and its tools, an overview of Progress SmartObject technology, and tutorials and exercises that help you better understand SmartObject technology and how to use the ADE to develop applications.

*Progress Language Tutorial for Windows* and *Progress Language Tutorial for Character*

Platform-specific tutorials designed for new Progress users. The tutorials use a step-by-step approach to explore the Progress application development environment using the 4GL.

*Progress Master Glossary for Windows* and *Progress Master Glossary for Character* (EDOC only)

Platform-specific master glossaries for the Progress documentation set. These books are in electronic format only.

*Progress Master Index and Glossary for Windows* and *Progress Master Index and Glossary for Character* (Hard copy only)

Platform-specific master indexes and glossaries for the Progress hard-copy documentation set.

*Progress Startup Command and Parameter Reference*

A reference manual that describes the Progress startup commands and parameters in alphabetical order.

*Welcome to Progress* (Hard copy only)

A booklet that explains how Progress software and media are packaged. An icon-based map groups the documentation by functionality, providing an overall view of the documentation set. *Welcome to Progress* also provides descriptions of the various services Progress Software Corporation offers.

### Development Tools

*Progress ADM 2 Guide*

A guide to using the Application Development Model, Version 2 (ADM 2) application architecture to develop Progress applications. It includes instructions for building and using Progress SmartObjects.

*Progress ADM 2 Reference*

A reference for the Application Development Model, Version 2 (ADM 2) application. It includes descriptions of ADM 2 functions and procedures.

*Progress Basic Database Tools* (Character only; information for Windows is in online help)

A guide for the Progress Database Administration tools, such as the Data Dictionary.

*Progress Basic Development Tools* (Character only; information for Windows is in online help)

A guide for the Progress development toolset, including the Progress Procedure Editor and the Application Compiler.

*Progress Debugger Guide*

A guide for the Progress Application Debugger. The Debugger helps you trace and correct programming errors by allowing you to monitor and modify procedure execution as it happens.

*Progress Help Development Guide* (Windows only)

A guide that describes how to develop and integrate an online help system for a Progress application.

*Progress Translation Manager Guide* (Windows only)

A guide that describes how to use the Progress Translation Manager tool to manage the entire process of translating the text phrases in Progress applications.

*Progress Visual Translator Guide* (Windows only)

A guide that describes how to use the Progress Visual Translator tool to translate text phrases from procedures into one or more spoken languages.

### Reporting Tools

*Progress Report Builder Deployment Guide* (Windows only)

An administration and development guide for generating Report Builder reports using the Progress Report Engine.

*Progress Report Builder Tutorial* (Windows only)

A tutorial that provides step-by-step instructions for creating eight sample Report Builder reports.

*Progress Report Builder User's Guide* (Windows only)

A guide for generating reports with the Progress Report Builder.

*Progress Results Administration and Development Guide* (Windows only)

A guide for system administrators that describes how to set up and maintain the Results product in a graphical environment. This guide also describes how to program, customize, and package Results with your own products. In addition, it describes how to convert character-based Results applications to graphical Results applications.

*Progress Results User's Guide for Windows* and *Progress Results User's Guide for UNIX*

Platform-specific guides for users with little or no programming experience that explain how to query, report, and update information with Results. Each guide also helps advanced users and application developers customize and integrate Results into their own applications.

### 4GL

*Building Distributed Applications Using the Progress AppServer*

A guide that provides comprehensive information about building and implementing distributed applications using the Progress AppServer. Topics include basic product information and terminology, design options and issues, setup and maintenance considerations, 4GL programming details, and remote debugging.

*Progress External Program Interfaces*

A guide to accessing non-Progress applications from Progress. This guide describes how to use system clipboards, UNIX named pipes, Windows dynamic link libraries, Windows dynamic data exchange, Windows ActiveX controls, and the Progress Host Language Call Interface to communicate with non-Progress applications and extend Progress functionality.

*Progress Internationalization Guide*

A guide to developing Progress applications for markets worldwide. The guide covers both internationalization—writing an application so that it adapts readily to different locales (languages, cultures, or regions)—and localization—adapting an application to different locales.

*Progress Language Reference*

A three-volume reference set that contains extensive descriptions and examples for each statement, phrase, function, operator, widget, attribute, method, and event in the Progress language.

*Progress Programming Handbook*

A two-volume handbook that details advanced Progress programming techniques.

### Database

*Progress Database Design Guide*

A guide that uses a sample database and the Progress Data Dictionary to illustrate the fundamental principles of relational database design. Topics include relationships, normalization, indexing, and database triggers.

*Progress Database Administration Guide and Reference*

This guide describes Progress database administration concepts and procedures. The procedures allow you to create and maintain your Progress databases and manage their performance.

### DataServers

Progress DataServer Guides

These guides 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. Each DataServer has its own guide as follows:

- *Progress/400 Product Guide*
- *Progress DataServer for Microsoft SQL Server Guide*
- *Progress DataServer for ODBC Guide*
- *Progress DataServer for ORACLE Guide*

MERANT ODBC Branded Driver Reference

The Enterprise DataServer for ODBC includes MERANT ODBC drivers for all the supported data sources. For configuration information, see the MERANT documentation, which is available as a PDF file in *installation-path*\odbc. To read this file you must have the Adobe Acrobat Reader Version 3.1 or higher installed on your system. If you do not have the Adobe Acrobat Reader, you can download it from the Adobe Web site at: http://www.adobe.com/prodindex/acrobat/readstep.html.

### SQL-89/Open Access

*Progress Embedded SQL-89 Guide and Reference*

A guide to Progress Embedded SQL-89 for C, including step-by-step instructions on building ESQL-89 applications and reference information on all Embedded SQL-89 Preprocessor statements and supporting function calls. This guide also describes the relationship between ESQL-89 and the ANSI standards upon which it is based.

*Progress Open Client Developer's Guide*

A guide that describes how to write and deploy Java and ActiveX applications that run as clients of the Progress AppServer. The guide includes information about how to expose the AppServer as a set of Java classes or as an ActiveX server.

*Progress SQL-89 Guide and Reference*

A user guide and reference for programmers who use interactive Progress/SQL-89. It includes information on all supported SQL-89 statements, SQL-89 Data Manipulation Language components, SQL-89 Data Definition Language components, and supported Progress functions.

### SQL-92

*Progress Embedded SQL-92 Guide and Reference*

A guide to Progress Embedded SQL-92 for C, including step-by-step instructions for building ESQL-92 applications and reference information about all Embedded SQL-92 Preprocessor statements and supporting function calls. This guide also describes the relationship between ESQL-92 and the ANSI standards upon which it is based.

*Progress JDBC Driver Guide*

A guide to the Java Database Connectivity (JDBC) interface and the Progress SQL-92 JDBC driver. It describes how to set up and use the driver and details the driver's support for the JDBC interface.

*Progress ODBC Driver Guide*

A guide to the ODBC interface and the Progress SQL-92 ODBC driver. It describes how to set up and use the driver and details the driver's support for the ODBC interface.

*Progress SQL-92 Guide and Reference*

A user guide and reference for programmers who use Progress SQL-92. It includes information on all supported SQL-92 statements, SQL-92 Data Manipulation Language components, SQL-92 Data Definition Language components, and Progress functions. The guide describes how to use the Progress SQL-92 Java classes and how to create and use Java stored procedures and triggers.

### Deployment

*Progress Client Deployment Guide*

A guide that describes the client deployment process and application administration concepts and procedures.

*Progress Developer's Toolkit*

A guide to using the Developer's Toolkit. This guide describes the advantages and disadvantages of different strategies for deploying Progress applications and explains how you can use the Toolkit to deploy applications with your selected strategy.

*Progress Portability Guide*

A guide that explains how to use the Progress toolset to build applications that are portable across all supported operating systems, user interfaces, and databases, following the Progress programming model.

### WebSpeed

*Getting Started with WebSpeed*

Provides an introduction to the WebSpeed Workshop tools for creating Web applications. It introduces you to all the components of the WebSpeed Workshop and takes you through the process of creating your own Intranet application.

*WebSpeed Installation and Configuration Guide*

Provides instructions for installing WebSpeed on Windows and UNIX systems. It also discusses designing WebSpeed environments, configuring WebSpeed Brokers, WebSpeed Agents, and the NameServer, and connecting to a variety of data sources.

*WebSpeed Developer's Guide*

Provides a complete overview of WebSpeed and the guidance necessary to develop and deploy WebSpeed applications on the Web.

*WebSpeed Version 3 Product Update Bulletin*

A booklet that provides a brief description of each new feature of the release. The booklet also explains where to find more detailed information in the documentation set about each new feature.

*Welcome to WebSpeed* (Hard copy only)

A booklet that explains how WebSpeed software and media are packaged. *Welcome to WebSpeed* also provides descriptions of the various services Progress Software Corporation offers.

### Reference

*Pocket Progress* (Hard copy only)

A reference that lets you quickly look up information about the Progress language or programming environment.

*Pocket WebSpeed* (Hard copy only)

A reference that lets you quickly look up information about the SpeedScript language or the WebSpeed programming environment.

### SQL-92 Reference

(These are non-Progress resources available from your technical bookseller.)

*A Guide to the SQL Standard*

Date, C.J., with Hugh Darwen. 1997. Reading, MA: Addison Wesley.

*Understanding the New SQL: A Complete Guide*

Melton, Jim (Digital Equipment Corporation) and Alan R. Simon. 1993. San Francisco: Morgan Kaufmann Publishers.

# 1

# Introduction

According to many industry estimates, in applications with a graphic user interface, the GUI itself might account for as much as 75% of the total application code.

The Progress AppBuilder tool is a Rapid Application Prototyping/Development (RAP/RAD) tool similar in concept to, for example, Microsoft's Visual development tools. The main component of AppBuilder is a What-You-See-Is-What-You-Get (WYSIWYG) interface layout editor. AppBuilder appreciably simplifies your application programming effort by

- Supporting drag-and-drop layout of interface elements
- Creating the object description and initialization code on your behalf
- Helping you create and manage the integrating code.

This chapter briefly introduces some important concepts, and describes AppBuilder and its place in the Progress application programmer's toolkit:

- The Progress Application Development Model
- Smart Versus Basic Objects
- External Versus Internal Procedures

- Visible Versus Invisible Objects
- g18
- Widgets
- Signatures
- Anatomy of a Database Application
- Working with AppBuilder
- Supported Application Building Blocks

## 1.1 Progress Application Development Model

AppBuilder is part of the Application Development Model (ADM) architecture, currently in Version 2.

Today's desktop computers offer more raw power than even supercomputers did thirty years ago. Programmers are now routinely being tasked to produce applications that were once hardly imaginable, and they are falling behind. The demand for new software overwhelms the capacity of programmers to create it using traditional methodologies.

The ADM addresses this issue by emphasizing object-oriented reusability.

The ADM supports the creation of large-scale, reusable application building blocks that are able to communicate with one another in well-defined ways. Once the blocks have been created, application programmers can assemble them into new applications in a fraction of the time it would require to create the applications from scratch. This is the principle of the run-time library: avoid having to repeat the same or a similar development effort each time. Spend the time and energy only once. Create sophisticated general-purpose modules, and then reuse them over and over again.

The ADM architecture has four basic components:

- A hierarchical set of object classes from which to develop reusable components
- A defined methodology for inter-object communication
- A way to set object properties using public data
- "Super procedures," analogous to dynamically linked libraries, that implement object behaviors

For detailed information about the Application Development Model, see the *Progress ADM 2 Guide*. The related API reference is the *Progress ADM 2 Reference*.

## 1.2 Concepts and Terminology

You will find it easier to use AppBuilder if you understand some related concepts and terminology.

### 1.2.1 External Versus Internal Procedures

In Progress 4GL terminology, *external procedure* is the usual term for a separate subprogram module that exists as a set of related files on disk and can be independently developed and compiled. An external procedure is not a procedure in the sense of being a defined 4GL subprogram that begins with the reserved word PROCEDURE or FUNCTION, but it is callable in much the same way. An external procedure's main filename serves as its identifier, and you can use it in references; for example:

```
RUN example.w.
```

External procedures typically have some number of *internal procedures* and functions defined locally to them. Because these internal procedures and functions implement the external procedure's repertoire of behaviors, they are often known as *methods*. They are defined as normal Progress 4GL subprograms and begin with one of the reserved words PROCEDURE or FUNCTION.

### 1.2.2 AppBuilder Objects

AppBuilder provides for your use a full array of basic 4GL objects, several powerful ADM SmartObjects, and even three ActiveX (OCX) objects.

Basic objects are objects that are available directly as part of the Progress 4GL, for example in a VIEW–AS clause. Some are very basic indeed—static Rectangles, for example—while others, such as Editors, can be quite complex underneath.

Table 1–1 lists the Progress basic 4GL objects available in AppBuilder. The descriptions represent how the objects appear under MS-Windows.

**Table 1–1: Basic 4GL Object Types** *(1 of 3)*

| Role | Object | Description |
|---|---|---|
| Organizer | Window | The outermost organizer object. It is not modal, can have a menu bar and pulldowns, and can parent all other objects. |
| | Dialog Box | Technically a type of Frame with its own dedicated Window. Dialog Boxes are modal, cannot have a menu bar, cannot be resized, and must be parented by a Window. You typically use them for capturing user input on a specific issue. |
| | Frame | An object that delimits a rectangular area and defines a **TAB**-traversal group. Frames can have their own border and title bar. You can use Frames to organize and display other objects. |
| | Rectangle | A static object that visually delimits a rectangular area. You can use a Rectangle to group other objects for emphasis. A Rectangle does not define a traversal group or have any active characteristics; it is purely a visual object. |
| Data Access | Query | A group of related instructions that tell Progress to find a subset of records from one or more database tables. This is a standalone object, but several objects have a query function embedded in them. |
| Data Display and Capture | Data Browser | An object that displays the results of a database query in row/column tabular format. Each row represents a record, and each column a field. Usually called a Browse widget. |

**Table 1–1: Basic 4GL Object Types** *(2 of 3)*

| Role | Object | Description |
|---|---|---|
| (Data Display and Capture) | Combo Box | An object that combines a Fill–in with a Selection List. The Fill–in shows the current value. The user can set the value by typing into the Fill–in, or by picking an item from those in the List. |
| | Data Viewer | An object that displays the results of a database query one record at a time, using one Fill–in per field. Also called a DB Fields widget |
| | Editor | A field-level object that allows editing of large (typically multi-line) character fields. By default, Editors support features such as cut, copy, paste, and word-wrap. |
| | Fill–in (Edit box) | A field-level object that accepts user input, typically a single line of text. The system evaluates that text, possibly performing additional processing such as type conversion and validation, and assigns the result to a database field or a variable. |
| | Radio-button Set | A group of buttons representing a set of values for a variable, only one of which can be valid at a given time. Selecting a button automatically deselects the previous choice. |
| | Selection List (Pick List) | A scrollable list of character strings. The strings are the possible values for an underlying field or variable. You can use a Selection List to allow an end user to select one or several items from a predetermined list of character values. |

**Table 1–1: Basic 4GL Object Types** *(3 of 3)*

| Role | Object | Description |
|---|---|---|
| (Data Display and Capture) | Slider Control | A virtual knob that the user moves along a track. The track represents some subrange of INTEGER; a built-in readout displays the current value. The limiting values are -32,768 and 32,767. Only a value of type INTEGER, without extent, can be displayed by a Slider. |
| | Toggle Box (Check Box) | A small, square box that represents a logical value. You can use groups of Check Boxes to represent sets of properties. |
| Action Control | Button (Pushbutton) | A field-level object that a user typically selects to invoke some action. AppBuilder offers one generic Pushbutton plus others predefined for special purposes. |
| Other | Text Object (Static Text) | Standalone text that is not bound to any other object, though it might appear to be. |
| | Image Object (Static Image) | A static field-level object used to display a bitmap from a graphics file. Progress supports many different graphic formats under MS-Windows. |

SmartObjects are much more complex than the objects listed in Table 1–1. True subprogram modules, SmartObjects are external procedures that may contain hundreds of lines of 4GL code in themselves and their superprocedures. All SmartObjects are implemented as external procedures, but only those external procedures based on the file `smart.i` qualify as SmartObjects.

Table 1–2 lists the SmartObjects available in the current release, and the abbreviations used elsewhere in this document.

**Table 1–2: SmartObject Types** *(1 of 3)*

| Role | SmartObject | Description |
|---|---|---|
| Organizer | SmartWindow (SWin) | Outermost Smart organizer object. Acts as a container and integrator for other objects, including other SmartContainers such as SmartFrames, etc. |
| | SmartDialog (SDialog) | A Smart, special-purpose Frame object supported by a dedicated window. |
| | SmartFrame (SFrame) | Allows you to create reusable Smart layouts not suited to implementation as SmartDialogs. |
| | SmartFolder (SFolder) | Implements the tabbed-file-folder metaphor for managing paged interface displays. |
| | Simple SmartContainer (SSC) | Provides all functionality from the SmartContainer class without the overhead of a visible window. Useful for modules that can operate unattended in the background. |
| Data Access | SmartDataObject (SDO) | A query interface to a database. This object, available in both static and dynamic forms, supplies the data stream used by other objects such as the SmartDataViewer or the SmartDataBrowser. An SDO has no visible representation at run time. |
| | SmartBusinessObject (SBO) | A dedicated organizer object that integrates up to 20 SDOs, providing a single point of contact for other objects. In addition, the SBO allows you to update from multiple SDOs in a single server-side transaction |
| | SmartFilter (SFilter) | A dynamic record-filtering object, visible at run time. Offers the user the ability to focus the associated SmartDataObject's query as tightly as desired. |

**Table 1–2: SmartObject Types** *(2 of 3)*

| Role | SmartObject | Description |
|---|---|---|
| Data Display and Capture | SmartDataBrowser (SDB) | Encapsulates a browse widget to view and possibly update data. Displays in a row/column format, and can present the fields for multiple records simultaneously. |
| | SmartDataViewer (SDV) | Presents a set of fields representing a single record. You can order and arrange the fields visually in any way that meets your needs. |
| | SmartDataField (SDF) | Adds special-purpose functionality to a SDV on a per-field basis. The special functionality can be of any desired sophistication. |
| | SmartSelect (SSelect) | A specialized SmartDataField representing a data-driven pick list. You would use it for updating a field in one table based on the values drawn from a related table. |
| Data Communication | SmartB2BObject (SBB) | Transforms message-body blocks of data between 4GL and XML representations based on a protocol (mapping) file that you create. |
| | SmartSender (SSndr) and SmartReceiver (SRcvr) | Transforms message-body blocks of data in ways that you define. The code you write can be anything from no-ops to the most sophisticated transformations; these are very general tools. |
| | SmartRouter (SRtr) | Determines which SBB should handle some message received by a SmartConsumer, and starts that object. |
| | SmartProducer (SProdr) and SmartConsumer (SConsr) | These objects are the SmartObject interface to the SonicMQ messaging system. The SProdr creates message bodies for use by other objects, and inserts them into the message system when complete. The SConsr watches the inbound piplelines and passes incoming messages upstream for processing. |

**Table 1–2: SmartObject Types** *(3 of 3)*

| Role | SmartObject | Description |
|---|---|---|
| Action Control | SmartToolbar (STB) | A general-purpose control object, optionally combining a menu-bar system with a toolbar object. |
| | SmartPanel (SP) | A cluster of dedicated Pushbuttons. AppBuilder offers versions for Navigation, Update, and Commit. |
| Other | Simple SmartObject (SSO) | An unfinished object that serves as a basis for new SmartObject types that you might wish to define. |

In addition to basic 4GL objects and ADM-compliant SmartObjects, AppBuilder also supports the popular ActiveX object type. ActiveX (OCX) objects resemble SmartObjects in implementing complex functionality, but are defined on Microsoft's ActiveX standard rather than the ADM. ActiveX components are available from many third-party sources.

Table 1–3 lists the ActiveX (OCX) objects AppBuilder provides for your use.

**Table 1–3: ActiveX (OCX) Object Types**

| Role | ActiveX Object | Description |
|---|---|---|
| Data Display and Capture | Combo Box | An object that combines a Fill–in with a hidden Selection List. Once the user makes the Selection List visible by pressing the associated button, choosing from the listed items sets the current value of the Fill–in. |
| | Spin Control | Two opposed pushbuttons and an optional readout field. Operating one of the pushbuttons changes the value of the object either up or down. |
| Other | Timer | Generates an event at some regular interval that you specify. |

### 1.2.3 What Is Smart About SmartObjects?

SmartObjects are not smart in any artificial-intelligence sense. They are smart only in comparison with basic objects. As ADM-compliant objects, they can all communicate with one another through SmartLinks, and as external-procedure objects, they can have any amount of sophistication the developer chooses to provide. If using the Progress 4GL rather than a 3GL corresponds to using large- rather than medium-scale chip integration for building hardware, then using SmartObjects in your applications is like using VLSI chips rather than LSI. The building blocks are bigger and more capable, you can hook them together more easily, and they require less unit testing.

Table 1–4 summarizes some of the advantages of using SmartObjects.

**Table 1–4: Benefits of Using SmartObjects**

| Benefit | Feature |
|---|---|
| Quicker results | Allows you to assemble more sophisticated applications in less time, since each SmartObject master is a predefined application component that can encapsulate more complex functionality than an individual Basic object. |
| Easier integration | Makes it easier to integrate individual object instances into applications because the ADM gives SmartObjects standardized ways of interacting with each other. |
| More reusability | Creates code that can be used repeatedly because a single object can have multiple run-time instances. |
| Easier maintenance | Organizes your development process so that complex applications are easy to maintain. SmartObject code is more maintainable because:<br>• Changes to any SmartObject master files are automatically reflected in all instances of that SmartObject.<br>• Each object can be written without specific knowledge of other objects. Modules that are independent in this way are referred to as being "loosely coupled," a desirable trait. |
| Greater consistency | Establishes a consistent look and feel across multiple applications because different applications can instantiate the same SmartObject master. This consistency makes it easier for you to create a special corporate image for your customers, if you choose. |

### 1.2.4 Visible Versus Nonvisible Objects

Although the majority of building blocks used in a database application might be associated with the GUI presentation, not all are. The modules that read and write the database, for example, generally operate behind the scenes. There is usually no reason to give those operations a visible representation at run time. But at design time, the situation is different. Programmers require easy access to all building blocks. For your convenience, AppBuilder makes every building block visible in some way at design time, even those that are not visible at run time.

The design-time representation of visible objects is nearly identical to their run-time representation. This means that you can usually position and size them in a WYSIWYG manner.

The design-time representation of a nonvisible SmartObject is a small, borderless area containing the object's filename, an icon that identifies the object's class, and a menu button. Figure 1–1 is an example of how a nonvisible SmartObject appears at design time. (The eight small black squares are the resizing handles that appear when the object is selected.)

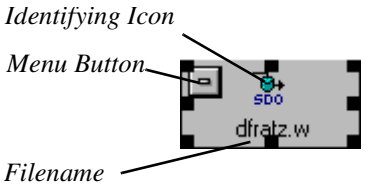



**Figure 1–1: Design-time Representation of a Nonvisible SmartObject**

All SmartObjects display a Menu Button (sometimes called a Ventilator Button) as part of their design-time representation. The menu under the Menu Button, shown in Figure 1–2, provides access to the object's property sheets, SmartLinks, and master definition.

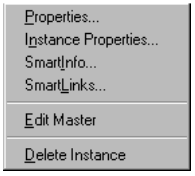



**Figure 1–2: SmartObject Menu-button Menu**

Other nonvisible objects are generally represented by simple, abstract icons. They exhibit little commonality in the details of their appearance.

## 1.3 Widgets

The word *widget* is a synonym for thing or item; it is a generic reference. In recent years, widget has become a widely used term among programmers for a user-interface element. The 4GL follows this convention.

In its role as a WYSIWYG layout editor, AppBuilder provides a design-time user-interface element even for objects that are neither part of the run-time user interface nor even visible at run time. In this guide, therefore, the term widget will be used freely as a synonym for object or building-block, regardless of whether it is a widget in the 4GL sense of the term.

## 1.4 Reading and Writing 4GL Widgets

When you use AppBuilder, you manipulate widgets in a largely WYSIWYG fashion, and AppBuilder generates the 4GL code later to make it all work as you intend. Because of this, it is natural and convenient to think of the widgets themselves as being the variables they represent. AppBuilder encourages this by using the widget identifier as the storage identifier. Thus, when you place into the workspace a Slider widget, for example, identifying it as object iExampleSlider that displays a certain integer subrange, etc., AppBuilder will later generate a 4GL statement such as this one:

```
DEFINE iExampleSlider AS INTEGER INITIAL 0
VIEW-AS SLIDER MIN 0 MAX 16384.
```

The syntax of this statement can give the impression to a programmer familiar with other languages that, in the 4GL environment, basic widgets are mere visual expressions of data (VIEW–AS) handled entirely at the level of the r-code interpreter. If that were so, then an assignment statement such as:

```
iExampleSlider = 8192.
```

would immediately cause (in this example) the knob of the Slider widget to move halfway up the track.

The actual situation is slightly more complicated.

Each widget represents **two** storage locations, not one. There is the storage location that you identify with the widget when you create the instance—iExampleSlider in the example—plus an anonymous storage location conceptually allocated in the system display buffer. These locations—main memory and screen display—are independent of one another, from the standpoint of your code.

User keyboard and mouse actions modify the anonymous location in the display buffer; assignment statements modify the identified location in main memory. A change made to one of the locations is **not** automatically reflected in the other location. You must **explicitly** close that gap by copying changes from one storage location to the other whenever required by the logic of your program.

If you want changes made to a variable to appear on the display, use DISPLAY:

```
iExampleSlider = 8192.
DISPLAY iExampleSlider WITH FRAME fMain.
/* replace fMain with the actual identifier of the frame
that encloses the iExampleSlider widget */
```

If you want changes made to the display to reflect in the variable, use ASSIGN:

```
/* copy the current value from the anonymous location */
ASSIGN iExampleSlider.

/* insert code here that uses the widget value */
```

The benefit of such a disconnected model is speed. Since device i/o is very slow compared to operations on memory, it is better not to do device i/o except when necessary. The Progress i/o model helps you manage that problem.

**NOTE:** The 4GL is much more complex, sophisticated, and adaptable than this brief introduction might imply. DISPLAY and ASSIGN are not the only routines that can move data between the two storage locations. Use this information as a way to make things happen while you are learning more subtle methods.

## 1.5 Inter-object Communication: Signatures

SmartObjects that pass database fields back and forth have embedded in them a string called a *signature*. A signature is a shorthand description of the fields the objects are prepared to supply or accept. Only if signatures match will the SmartObjects transfer data. Signature checking reduces the possibility of confusion and data corruption.

To be sure your SmartObjects can continue to successfully pass data back and forth, always recompile them after making a change to a data stream. For more information, see the section "Making Changes to the Data Stream" in Chapter 4, "Data-access Objects."

## 1.6 Anatomy of a Database Application

Although each application differs greatly in detail, there is an underlying framework that can be represented as

- Finding certain records in a database
- Displaying them
- Modifying them
- Updating the records in the database

Finding records involves opening a database file and searching through it for records that have certain characteristics. The desired characteristics are defined in a *query*. A query might be as simple as "get all records" or highly complex, involving comparisons on many different fields in a multitude of records.

Records returned by a query can be displayed in tabular format for easy browsing, formatted in a more complex way suitable for editing, or used to generate, for example, mailing labels.

Modifying a record is a simple clerical process on one level. But because humans are involved, mistakes are virtually guaranteed to occur. A careful developer will account for this fact and take steps to check the changes being made, insofar as checking is practical.

Once changes have been made, the modified records must be written back to the database. This can be done immediately, for simple updates, or buffered to a staging area as part of a more complex transaction process.

AppBuilder supports objects that address all four steps in this framework.

## 1.7 Working with AppBuilder

Creating an application with the help of AppBuilder is a four-step process:

1. Design your application on paper.
2. Create organizer windows and populate them.
3. Add additional glue and supporting code.
4. Test and refine the result.

AppBuilder can help with steps 2 through 4. Application design is beyond the current scope of this guide. The design process has both a technical and a usability aspect, and those developers who pay attention to both are generally the most successful.

There are a number of formal tools and methodologies that can help with technical design. Use whichever seems best. At present, however, there are no similar tools to help you make your application easy and pleasant to use. The best advice is to test your design assumptions early and often with real users. There are books on usable software design that can help acquaint you with those techniques.

## 1.8 Single-window or Multi-window Design?

A basic part of your application-design process will be to decide whether to use a single window or several. SmartWindows offer you the additional option of multiplexing your space by paging within a single window—a process that resembles using overlays.

There is no simple answer to this single-versus-many design question. A full discussion of the tradeoffs involved is beyond the current scope of this guide. One way to begin thinking about the question, though, is to consider how much natural parallelism is involved in the problem your application will address.

Single-window designs are often best for applications that deal with:

- A single, highly serial problem
- Multiple tasks that are largely unrelated

Multi-window designs might work well for applications that present separable but related clusters of functionality.

Multiplexed (paged, paneled) designs may seem the most natural choice for applications that require a mixed interface, where there is a certain core functionality that remains in view while another part of the window changes in a task-oriented way.

No matter what you choose, you will probably find that you can make legitimate arguments for a different choice. The advantages of one design over another are typically relative, not absolute. For example, AppBuilder itself uses a multi-window design (main, palette, workspace), yet it is not hard to imagine it as a single-window design instead.

Multi-window applications are generally somewhat more difficult to design and develop. You might decide during the design phase of the project that there are advantages to a multi-window design, but that they are outweighed by the practical difficulties. Or you might decide that the advantages you could offer your customer make a multi-window design well worth the extra effort required to create it.

For more information on single-window versus multi-window design issues, see the *Progress Programming Handbook*.

## 1.9 Designing for Character-based Environments

Applications you develop using the Progress 4GL are automatically capable of running in both GUI and character environments. But character environments have a much coarser granularity than GUI environments. It is simply not possible to do as much in a (typically) 80x21 character grid as in the more subtle context of a graphics display, and some widgets cannot be represented at all in a character environment. Moreover, the character environment cannot support

multi-window designs. For that reason, you should pay careful attention to design when creating an application that will run in both GUI and character environments.

AppBuilder will support you whether you choose to design specifically for each environment, or create a single code base for deployment to all environments. Appendix B, "Multiple Layouts" for a discussion of creating multiple layouts from a single code base.

## 1.10 Building an Application in AppBuilder

Before learning more about the underlying components of AppBuilder, consider the toy application shown below. This application uses four basic 4GL objects and five SmartObjects. Figure 1–3 shows the application as it appears at design time, and Figure 1–4 at run time.

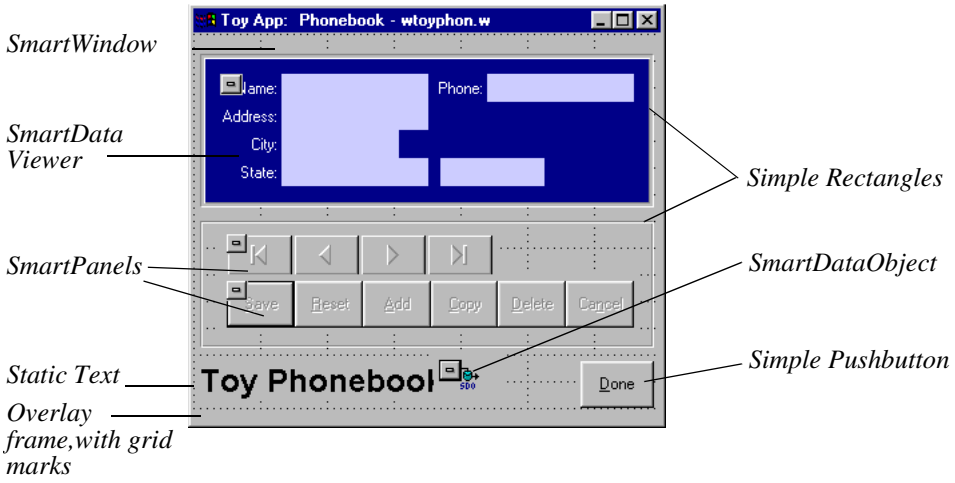



**Figure 1–3: Toy Phonebook Application at Design Time**




**Figure 1–4: Toy Phonebook Application at Run Time**

This application illustrates some important features of AppBuilder:

- The SmartWindow is an organizer object and a member of the class SmartContainer. SmartContainers supply an integrating context for other SmartObjects. Higher-level organizer objects such as Windows, Dialogs, and Frames provide an integrating context for all objects placed in them, but the level of integration depends on the objects involved.

- The SmartDataViewer, like most SmartObjects, is incorporated into the application as an *instance* of a *master* object. You create a SmartObject master in a separate operation before assembling the final application window. SmartObject masters are based on one of several SmartObject *templates* that are provided by AppBuilder.

- The two SmartPanels are supplied by AppBuilder predefined and precompiled. You simply select and place them.

- SmartObjects communicate smoothly with each other behind the scenes. For example, clicking on the navigation buttons in the SmartPanel causes the SmartDataViewer to display a new record. This command and data communication between SmartObjects takes place along SmartLink pathways.

  AppBuilder helps create appropriate SmartLinks during the application assembly process. Each SmartLink represents a type of message to be sent when an event occurs. The messages are sent by the SmartObject experiencing the event to all other SmartObjects that are registered as being interested in events of that kind.

- Although it is not visible at run time, the SmartDataObject is a critical component of this application. It is the object that receives the navigation requests from the SmartPanel and supplies the data to the SmartDataViewer for display and, perhaps, modification.

  Note that in Figure 1–3 the SmartDataObject widget is obscuring part of the large Static Text object, yet the text is completely visible in Figure 1–4. You can place nonvisible objects such as the SmartDataObject anywhere you like in your workspace. They are totally invisible at run time.

- The two static Rectangle widgets serve to visually pull together and regularize the edges of the controls and the Viewer object. Enclosing related widgets is one way to make them seem more related and reduce the sense of clutter. Even purely visual elements, used wisely, can greatly enhance the usability of your applications. Here, widening the border and filling the upper Rectangle with color (modifying the colors in the Viewer to agree) creates a bit of additional visual interest, though the deep blue color might be less appropriate in a real application than in this toy one.

- The basic, Static Text object is an example of a special font definition. Used carefully, special fonts can add to the usability and the visual appeal of your applications.

- The Done Pushbutton is another basic object. At run time, choosing the Done button ends execution and releases resources in an orderly way. AppBuilder supplies a number of predefined Pushbuttons as well as a generic one. When you add such a dedicated button to the layout, AppBuilder automatically generates appropriate trigger code. When you create your own applications, you have the option of choosing from the set of predefined buttons, or using the generic Pushbutton and writing your own trigger code.

  The Done button is an example of a custom object definition. It is defined in the `progress.cst` file. For information about creating and making available your own custom objects, Appendix C, "Customizing AppBuilder"."

Figure 1–5 shows the SmartLinks connecting the SmartObjects in the toy application. Conceptually, SmartLinks are dedicated, directional message pathways between two SmartObjects. The *event*-Source object experiences an event, and reports that event to the *event*-Target object using the 4GL Publish/Subscribe mechanism.

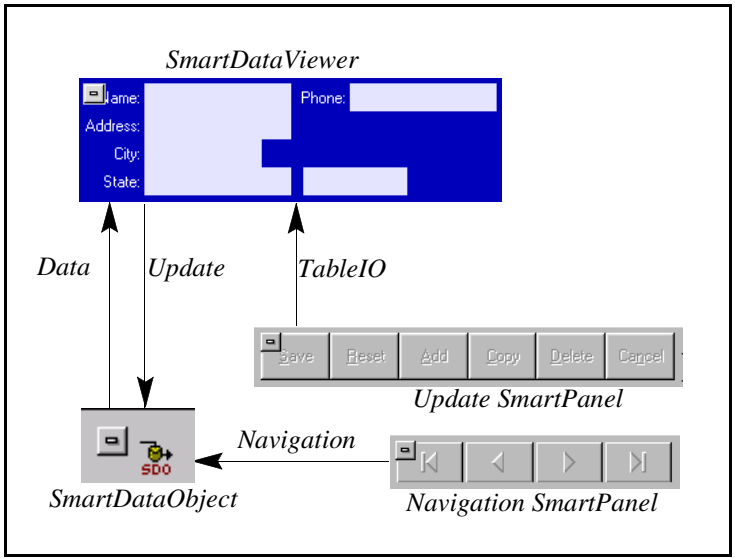



**Figure 1–5: SmartLinks Between SmartObjects**

Figure 1–5 shows four SmartLinks. The first is a Navigation SmartLink between the Navigation SmartPanel and the SmartDataObject. When the user chooses one of the Pushbuttons in the SmartPanel array, SmartPanel sends a Navigation message to the SmartDataObject, asking it to change the position of its current-record pointer.

Whether this will mean a new read from the storage device is a matter for the SmartDataObject to decide, based on factors including the size of the RowObject temp-table you declared when creating it. The SmartPanel knows nothing about such matters; SmartObjects are only loosely coupled to one another.

When the SmartDataObject has changed its pointer in response to the request from the SmartPanel, it sends a Data message to the consumer of the data stream it supplies: the SmartDataViewer. When the user changes the data, and confirms the change using the Update Panel, the Update Panel reports that confirmation to the Viewer, and the Viewer sends an Update message back to the SmartDataObject.

Not shown in Figure 1–5 are links between the SmartContainer (the SmartWindow) and the SmartObjects that it contains, because those links are automatically created and removed by AppBuilder as you add and remove SmartObjects from the SmartContainer's workspace. You never have to deal with them explicitly.

Figure 1–6 shows how these four links appear in the SmartLinks Editor dialog. For further information about the editor, see the section "SmartLinks Editor" in Appendix A, "Frequently Used Dialogs."

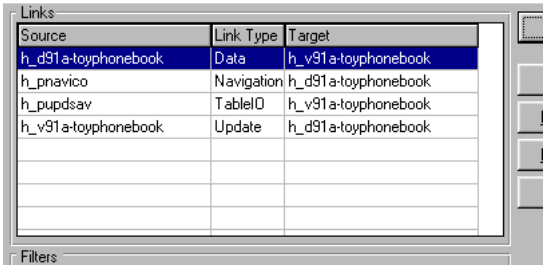



**Figure 1–6: Toy Phonebook Links in SmartLinks Editor**

# 2

# AppBuilder Interface

The AppBuilder interface has a number of components:

- Main window
- Object Tool Palette
- Workspace windows
- Source-code Section Editor
- Various dialog boxes for object properties, etc.

This chapter describes the AppBuilder user interface in detail, and explains how to use it.

## 2.1 Starting and Stopping AppBuilder

Depending on the product you purchased, you might be using AppBuilder to develop standard client/server applications for GUI or character mode environments, applications for the Web, or some combination. Regardless of your development goal, you start and stop a development session the same way.

You can start AppBuilder from the Application Development Environment (ADE) Desktop or from any tool that has a Tools menu, such as the Data Dictionary or the Procedure Editor. You can also start AppBuilder directly from the operating system.

### Starting from the ADE Desktop

To start AppBuilder from the ADE Desktop, choose the AppBuilder button, File→ New→ Window/Dialog, or Tools→ AppBuilder.

### Starting from Progress Tools

To start AppBuilder from the Progress Procedure Editor, Data Dictionary, or other tool, choose Tools→ AppBuilder from the tools main menu.

### Starting from the Operating System

To start AppBuilder from the operating system, enter the following command:

```
PROWIN32.EXE -p _ab.p
```

You can also choose AppBuilder from the MS-Windows Start menu.

### Stopping AppBuilder

You can end your development session and exit AppBuilder by any of these actions:

- Choose File→ Exit from the AppBuilder main menu
- Click on the close icon in AppBuilder's main window
- Select Close from the pop-up menu of the small AppBuilder icon on the main window

Progress returns you to where you started AppBuilder.

### 2.1.1 AppBuilder for Web-based Applications

This manual does not attempt to cover features and techniques specific to developing for the Web. For information on WebSpeed-specific features, see the *Getting Started with WebSpeed* manual or the AppBuilder online help.

## 2.2 AppBuilder Main Window

From the AppBuilder main window, shown in Figure 2–1, you can access all of AppBuilder's supporting functionality. The main window appears whenever you start AppBuilder.

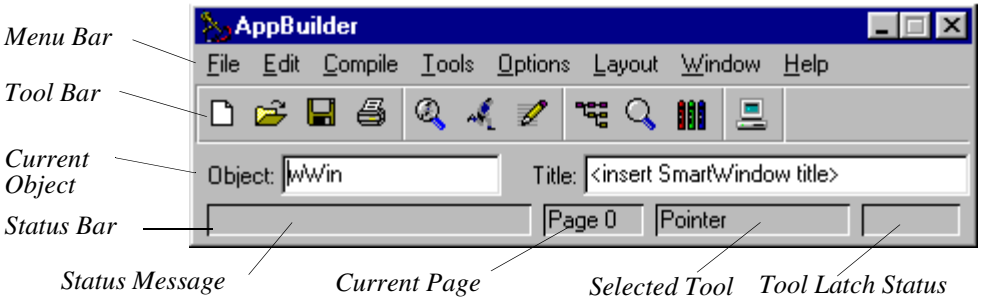



**Figure 2–1: AppBuilder Main Window**

The AppBuilder main window is divided into four areas:

- Menu bar
- Tool bar
- Current object fields
- Status bar fields

See the"Menu Bar" section for a description of the menu bar and the related toolbar.

### Current-Object Fields

The current-object fields display the Object identifier and, often, additional information such as a text or label string. Both fields are generally editable. Figure 2–2 shows current object fields for a SmartWindow.




**Figure 2–2: Current-Object Fields**

The current object is a single object that is part of the design window. Only one object can be current at a time. Thus, if you select more than one object in a design window, nothing will be listed in these fields.

The current object fields display the following information:

- **Object** — Specifies the name (identifier) of the current basic 4GL widget, SmartObject instance, structured procedure file, method library, or structured include file. For character windows, this field displays the words TERMINAL-SIMULATION when the window is selected. No window name displays because in character mode there is never more than one window open. If some object in the window is selected instead, that object's identifier displays here.
- **Title/Label/Text/Master/File** — For basic widgets other than Text, specifies the title or label, if one is available. For Text objects, which have no label or title, this field displays the Text widget's string value. For structured procedure files, method libraries, and structured include files, it specifies the pathname of the procedure file.

### Status Bar

Figure 2–3 shows the status bar.




**Figure 2–3: Status Bar**

From left to right, these fields display:

- Status information about current AppBuilder activity.
- The Design Page in the current Smart organizer object. If the current organizer/container is not Smart, this field is blank.
- The name of the currently selected object type in the Palette. For example: SmartDataBrowser.
- The text "LOCK" if the current object icon is latched (locked) down for multiple placements. It is blank otherwise.

## 2.3 Menu Bar

The menu bar provides access to nearly all the AppBuilder functionality, apart from the component objects. Note that some menu options are not available when you are working in WebSpeed development mode.

### 2.3.1 File Menu

The File menu allows you to create, open, save, and print procedure files, as shown in Table 2–1.

**Table 2–1: File Menu**

| File | |
|---|---|
| New... | Creates a new external procedure or component file. |
| Open... | Opens an existing external procedure or component file. |
| Close | Closes the current file. |
| Close Windows... | Closes some or all of the active windows. |
| Save | Saves the current object's files. |
| Save As... | Saves the current object's files, possibly under a different name. |
| Save All... | Saves all open files. |
| Print | Prints the source for the current procedure-based object. |
| (List of the most-recently-used structured-procedure objects.) | |
| Exit | Exits the current AppBuilder session. |

**File→ New**

Opens the New dialog box. After you choose the object or component you want to create, AppBuilder reads in the appropriate template. A *template* is an incomplete procedure file for an object. After loading the template, AppBuilder will take some further, template-specific action:

- Windows, dialog boxes, and outer-level frames: AppBuilder opens a *design window* or *workspace* where you assemble and arrange the widgets that will make up your application's user interface. AppBuilder also opens a workspace if you choose Simple SmartObject, since the majority of SmartObjects will have a visible representation.

  When designing WebSpeed applications, you do not work with a design window. See the *Getting Started with WebSpeed* tutorial for an overview.

- SmartObjects based on a master: AppBuilder starts the design wizard for that class of object. After the design wizard runs to completion, you will have a fully defined SmartObject master that you will normally save to disk.

- Procedure, include file, or methods library: AppBuilder opens two windows. One shows a tree view of the editable sections of that file type, and the other displays the built-in source-code section editor with a code stub for the current section type.

### File→ Open

Opens an existing object file and displays its associated design window or tree view. If AppBuilder cannot determine how to present the file in a better way, it will use a procedure window.

You can also drag and drop a file from the Windows Explorer or desktop to the AppBuilder main window. AppBuilder opens the file according to type, and prompts you to connect to any necessary databases.

### File→ Save

Saves the current procedure file locally or remotely, according to your current development mode. AppBuilder prompts you for a filename only if you have not saved this file before.

### File→ Save As

Opens the Save As dialog box for the current procedure file. Normally you would Save As in order to change the filename, but you can Save As to the original filename if you wish.

### File→ Save All

Immediately performs a Save on all open files being maintained by AppBuilder.

Newly-created files are saved locally or remotely according to the Development Mode setting. Previously-existing files are saved to the locations from which they were opened, regardless of the Development Mode setting.

**CAUTION:** You must perform individual Saves on any files that AppBuilder opens in a Procedure Editor window rather than a Section Editor window. Use of the Procedure Editor means that AppBuilder does **not** maintain the file.

### File→ Print

Prints the source code of the current structured-procedure file, not the currently selected object specifically. If you clear the Use Print Dialog Box option in the Preferences dialog box, the file is printed to the default printer immediately, without further prompting.

### 2.3.2 Edit Menu

The Edit menu allows you to perform a number of coarse-grain, generic editing operations such as cutting and pasting, as shown in Table 2–2.

**Table 2–2: Edit Menu**

| **Edit** | |
|---|---|
| Undo | Undoes the most recent operation, if possible. Item temporarily changes to Redo after an Undo. |
| Cut | Moves the current selection to the system Clipboard. |
| Copy | Copies the current selection to the system Clipboard. |
| Paste | Copies the contents of the system Clipboard to the current cursor location. |
| Duplicate | Duplicates the selection. |
| Delete | Removes the selection permanently, unless Undone. |
| Copy to File... | Exports the selection to a file on disk. |
| Insert from File... | Imports the objects defined in a file on disk. |
| Tab Order... | Sets the traversal order for the current frame. |
| Goto Page... | Changes the current ADM page. |

#### Edit→ Undo

Shortcut: **CTRL–Z** — Undoes the most recent operation, including a Delete, if possible. The menu item text shows what that operation was. Item temporarily changes to Redo after an Undo. If undoing the most recent operation is not possible, this item is greyed out.

#### Edit→ Cut

Shortcut: **CTRL–X** — Cuts the currently selected object, if any, to the clipboard.

#### Edit→ Copy

Shortcut: **CTRL–C** — Copies the currently selected object, if any, to the clipboard.

### Edit→ Paste

Shortcut: **CTRL–V** — Pastes a copy of the current clipboard contents to the current workspace, at the current cursor position.

### Edit→ Duplicate

Creates a copy of the currently selected object without using the Clipboard. The new copy is pasted directly on top of the original. To see the original, the duplicate must first be moved out of the way.

### Edit→ Delete

Removes the currently selected object, if any. You can recover a deleted object if you can Undo the operation, but you cannot recover by pasting because the Clipboard is not involved.

### Edit→ Copy to FIle

Opens a dialog box to copy the source code for the currently selected object to an export (.wx) file.

### Edit→ Insert from File

Opens a dialog box to insert the contents of some file into the procedure editor at the current insertion point.

### Edit→ Tab Order

Only enabled when a frame is selected, this item opens the editor that allows you to define the **TAB** traversal order for objects in the frame. The editor offers five predefined orderings plus a custom ordering that you can define on a per-frame basis.

### Edit→ Goto Page

Only enabled when editing a SmartContainer. Opens the dialog box that lets you set the ADM Design Page. See the "Simple SmartContainers"section in Chapter 3, "Organizer Objects" for information about the meaning of the Design Page.

### 2.3.3 Compile Menu

The Compile menu allows you to invoke the Progress interpreter to inspect, test, and debug your work, as shown in Table 2–3.

**Table 2–3: Compile Menu**

| **Compile** | |
|---|---|
| Run | Compiles and runs the current workspace. |
| Check Syntax | Compiles but does not run the current workspace. |
| Debug | Starts the debugger on the current workspace. |
| Code Preview | Opens a read-only window on the current source file. |
| Close Character Run Window | Closes the current character-mode window. |

#### Compile→ Run

Shortcut: **F2** — Compiles and runs the current workspace. While the workspace is running, no AppBuilder control is available other than the Stop button, which appears in place of the Run button in the toolbar.

#### Compile→ Check Syntax

Shortcut: **SHIFT-F2** — Does a syntax check on the current workspace without running it.

#### Compile→ Debug

Shortcut: **SHIFT-F4** — Compiles the current workspace and starts the Progress Debugger.

#### Compile→ Code Preview

Shortcut: **F5** — Opens a source-code viewer on the currently selected SmartObject master. If the currently selected SmartObject is an instance rather than a master, the viewer window will instead display the source of the current container object (workspace).

#### Compile→ Close Character Run Window

Closes the currently open character-mode test window, if any.

### 2.3.4 Tools Menu

The Tools menu, as shown in Table 2–4, gives you access to several of the important AppBuilder dialog boxes as well as the other Progress tools installed on your system.

**Table 2–4: Tools Menu**

| **Tools** | |
|---|---|
| New Procedure Window | Opens a version of the Procedure Editor in a new window. |
| New ADM Class... | Creates all files for a new class under the ADM standard. |
| Property Sheet... | Opens the base property sheet for the current object. |
| Procedure Settings... | Opens the base procedure-settings dialog box for the current procedural object. |
| Color... | Sets colors for the current selection. |
| Database Connections... | Connects and disconnects databases. |
| (List of all Progress tools installed on your system.) | |

#### Tools→ New Procedure Window

Shortcut: **CTRL–F3** — Opens a version of the Procedure Editor, with a reduced number of menu items. The Procedure Editor is not AppBuilder's built-in Section Editor, and should not be used to edit files that were created by AppBuilder. (Note that the behavior of this Procedure Editor may be slightly different to that of the standard Progress editor of the same name—the two do not use the same underlying code.)

#### Tools→ New ADM Class

Opens a dialog box to allow basic definition of a new class, generating all the relevant files. Entering the classname automatically fills in the blanks for the other fields.

#### Tools→ Property Sheet

Opens the appropriate base-properties dialog box for the current object.

### Tools→ Procedure Settings

Opens the file-level properties dialog box for the current procedure file. Procedure settings include the:

- Type of procedure file
- Filename
- Description
- Location of the properties for ActiveX objects, if any, in this context
- Directory in which AppBuilder compiles the procedure file
- External tables (if any) required for the procedure file
- Settings for running from AppBuilder
- Custom lists available for the procedure file
- Relevant information about partition and pages, if the file represents a SmartObject

Additionally, if you open a template, you can access the template's advanced procedure settings. These settings determine, among other things, what types of objects you can place in a design window. For more information, see the section "Procedure Settings" in Appendix A, "Frequently Used Dialogs."

### Tools→ Color

Opens the color selector for the current object. Not available in Remote (WebSpeed) mode.

### Tools→ Database Connections

Opens a window to display, add to, or remove current database connections.

## 2.3.5 Options Menu

The Options menu, as shown in Table 2–5, allows you to set both general and editor-specific preference items, and change the visibility and magnetism of the layout grid.

**Table 2–5: Options Menu**

| **Options** | |
|---|---|
| Preferences... | Opens the Preferences dialog box. |
| Editing Options... | Opens the dialog box for editor-specific preferences. |
| √ Snap to Grid | Turns the layout-grid magnetism on and off. |
| √ Display Grid | Turns the visibility of the layout grid on and off. |

#### Options→ Preferences

Opens the dialog box that allows you to set various preferences. See the "Setting Your Preferences" section for detailed information.

#### Options→ Editing Options

Opens the dialog box that allows you to set editing-environment preferences such as color-coding, tab size, etc. See the section"Setting Your Preferences" for detailed information.

#### Options→ Snap to Grid

Determines whether objects will snap to the nearest gridline(s) in frames. The gridlines need not be visible. This switch affects all layout workspaces, not just the current one.

#### Options→ Display Grid

Determines whether layout gridlines will be visible. This switch affects all layout workspaces, not just the current one.

### 2.3.6 Layout Menu

The Layout menu allows you to define additional layouts and to arrange selected objects in regular ways, as shown in Table 2–6.

**Table 2–6: Layout Menu** *(1 of 2)*

| **Layout** | |
|---|---|
| Alternate Layout... | Opens the Alternative Layout dialog box. |
| Center Left-to-Right in Frame | Centers selected objects horizontally. |

**Table 2–6: Layout Menu** *(2 of 2)*

| Layout | |
|---|---|
| Center Top-to-Bottom in Frame | Centers selected objects vertically. |
| Even Spacing Left-to-Right | Evens up the horizontal spacing between selected objects. |
| Even Spacing Top-to-Bottom | Evens up the vertical spacing between selected objects. |
| Move-to-Top | Moves the selected object all the way forward. |
| Move-to-Bottom | Moves the selected object all the way back. |
| Align | Allows you to align multiple objects along a selected feature. |

### Layout→ Alternate Layout

Opens the Alternate Layout dialog box. You can define the conditions for additional layouts using this dialog box. For more information, see Appendix B, "Multiple Layouts".

### Layout→ Center Left-to-Right in Frame
### Layout→ Center Top-to-Bottom in Frame

Centers all selected objects along one axis (X or Y) within the frame, without changing their positions on the other axis.

### Layout→ Even Spacing Left-to-right
### Layout→ Even Spacing Top-to-Bottom

Evenly spaces the objects from one another along one axis (X or Y) without changing the overall space they take up, and without changing their positions on the other axis.

### Layout→ Move-to-Top
### Layout→ Move-to-Bottom

Changes the Z–axis layering order of the selected object. Moving it to the top makes it seem to overlay all other objects (that is, it makes the object seem closest to the viewer); moving it to the bottom makes it seem to be overlaid by all other objects (makes it seem furthest away). Some objects do not respond to relayering requests.

#### Layout→ Align

Opens a submenu where you can choose the feature on which to align multiple objects. Objects can be aligned on their:

- Label colons
- Left or right edges, or horizontal center (vertical centerline)
- Top or bottom edges, or vertical center (horizontal centerline)

### 2.3.7 Window Menu

The Window menu provides access to the built-in Section Editor, the common-properties editor window, the OCX properties editor, and a list of current objects, as shown in Table 2–7.

**Table 2–7: Window Menu**

| **Window** | |
|---|---|
| Code Section Editor | Invokes built-in source-code editor |
| Hide Object Palette | Determines whether the objects palette is visible. |
| Properties Window | Opens common-properties window. |
| OCX Property Editor | Opens OCX properties window. |
| (List of procedure objects currently open) | |

#### Window→ Code Section Editor

Shortcut: **CTRL–S** — Invokes the built-in source-code editor on the current object. Unlike other editors, including the standard Progress Procedure Editor, the built-in Section Editor will not make object source code unmanageable by AppBuilder.

#### Window→ Hide Object Palette

Determines whether the Objects Palette is visible. The menu item changes to "Show Object Palette" when the Palette is hidden.

#### Window→ Properties Window

Opens the window that lists certain of the common properties of all selected objects. For example, all objects have an origin and a size. Where property values differ between objects, as

is typically the case, the differences are represented by a range in the Properties Window. You can set substitute a single value for the range, and this will affect all selected objects.

#### Window→ OCX Property Editor

If the current object is an ActiveX object, opens an editor window for the object's properties. If the current object is not an ActiveX object, selecting this item opens a window with no content.

### 2.3.8 Help Menu

The Help menu provides access to useful information of various kinds, as shown in Table 2–8.

**Table 2–8: Help Menu**

| **Help** | |
|---|---|
| Help Topics | Opens the Online Help window. |
| Messages... | Displays text associated with a message number. |
| Recent Messages... | Allows you to review recent messages. |
| Cue Cards... | Allows you to review recently dismissed cue cards. |
| About AppBuilder | Provides information about the current AppBuilder version. |

#### Help→ Help Topics

Opens the online help window.

#### Help→ Messages

Opens a window that displays the text associated with the message number you enter.

#### Help→ Recent Messages

Opens a window that allows you to review recent messages that AppBuilder generated, if any.

#### Help→ Cue Cards

Opens a window that displays recently dismissed cue cards, if any.

#### Help→ About AppBuilder

Opens a window that displays the version of AppBuilder you are running, the current version of the operating system, and the total and free hardware memory.

### 2.3.9 Tool Bar

Most buttons on the tool bar correspond to menu options. The exceptions are the List Objects button, and the Development Mode Switch. The Development Mode Switch is a special option not present in all versions of AppBuilder. See Figure 2–4.

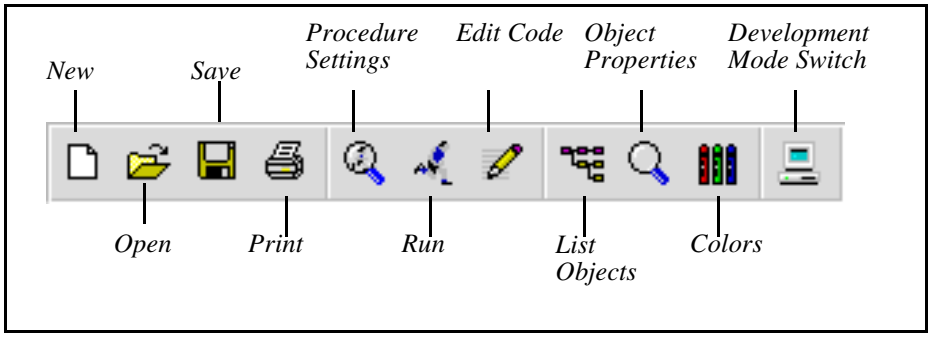



**Figure 2–4: AppBuilder Tool Bar**

You can use the Windows Tooltip feature to view the action labels for each button. For a more complete description of the button's function, see the description for the corresponding menu item, as noted.

Click on the tool bar buttons to invoke the following AppBuilder actions:

- **New** — Corresponds to File→ New.
- **Open** — Corresponds to File→ Open.
- **Save** — Corresponds to File→ Save.
- **Print** — Corresponds to File→ Print.
- **Procedure Settings** — Corresponds to Tools→ Procedure Settings.
- **Run** — Corresponds to Compile→ Run.
- **Edit Code** — Corresponds to Window→ Code Section Editor.
- **List Objects** — Invokes the List Objects dialog box, which lists all the objects in use in the AppBuilder session, allowing you to access their Property Sheets. There is no other way to open this dialog box.
- **Object Properties** — Corresponds to Tools→ Property Sheet.
- **Colors** — Corresponds to Tools→ Color.

- **Development Mode Switch** — If present, alternates the development mode between local and remote. These modes affect where AppBuilder compiles, runs, and saves objects. For more information on using remote (WebSpeed) development mode, see the *Getting Started with WebSpeed* manual.

## 2.4 Object Tools Palette

For your convenience, the Object Tools Palette makes available all the component objects supplied with AppBuilder. Only the major organizers (Windows, Dialogs, SmartFrame) are not on the Palette. Create them from the list in the File→ New dialog box.

Figure 2–5 shows the Object Tools Palette.




**Figure 2–5: Object Tools Palette**

Each icon has a Windows Tooltip that identifies it. Or, if you would prefer to work with a pulldown menu of object names instead of the icons, select the Options→ Show Menu Only item on the Palette's menu.

By dragging the edge of the Palette appropriately, you can change the shape of the Palette through a broad range, from one (long) row to two columns. You can also choose where to position the Palette within the MS-Windows display area.

The default set of object tools is:



**Pointer** — This tool does not create any object; it is part of AppBuilder itself. Use this tool when selecting, rearranging, or resizing objects in the visual-layout workspace. Also choose this tool if you wish to change your mind after selecting another tool but before placing that object.



**Basic Data Viewer** — (DB Fields) Prompts you to connect a database, if necessary, and then to identify first a table and then fields in the table. Creates a query based on your selection, and adds to the workspace one labelled Fill-in per field, arranged in a single column.



**Query** — Prompts you to connect a database, if necessary, and then starts the Query Builder dialog box. This widget is visible only at design time.



**Basic Data Browser** — Prompts you to connect a database, if necessary, and then starts the Query Builder dialog box. The results of the query are automatically displayed by the widget.



**Frame** — Adds a Frame to the workspace. Use a Frame to visually group other objects or, generally only in character mode, to serve as a window (viewport) into a data display.



**Rectangle** — Adds a static Rectangle to the workspace. Rectangles have no associated functionality. Use Rectangles to visually group other objects.



**Image** — Adds a static, bitmapped Image to the workspace. AppBuilder supports many different image formats under MS-Windows. This object is, of course, meaningless in a character environment.



**Radio Button Set** — Adds a set of three Radio Buttons to the workspace, with generic labels. You can change the number of buttons in the set after placement. Radio-button sets represent any small set of discrete values where only one can be valid at a given time.



**Toggle Box** — Adds a single Toggle Box (Check Box) to the workspace, with a generic label. Toggle Boxes represent a single LOGICAL (Boolean) value. Generally, you should set up the box so that the unchecked condition represents No, False, Empty, Not set, etc.



**Slider Control** — Adds a Slider Control widget to the workspace, with optional readout. You can choose to orient this widget vertically or horizontally. A Slider Control can represent INTEGER values from –32768 to 32767.




**Pushbutton** — Adds a Pushbutton to the workspace, with a generic label. You would typically use a Pushbutton for activating some process rather than selecting a static condition. Pushbuttons are represented as momentary-on (nonlatching) devices: when released, they return to the up or inactive state.



**Selection List** — Adds a Selection List to the workspace. A Selection List presents a scrollable set of items displayed as character strings.



**Multi-line Editor** — Adds a multi-line text Editor widget to the workspace. You can set the size of the edit buffer, as well as the availability of features such as wordwrap.



**Combo Box** — Adds to the workspace a Drop-down List, or a Combo Box with your choice of drop-down or visible list element. A true Combo Box allows the user to type a value into the fill–in component or select from the items displayed by the list component.



**Fill–in** — Adds a single-line Editor to the workspace. Although a Fill–in presents its data at run-time as a character string, you can specify automatic type conversion such that, for example, the value is actually LOGICAL with the user's only choice being between some two string literals you define such as "True" and "False."



**Static Text** — Adds a string of static text to the workspace. You control font, color, and size.



**ActiveX (OCX) Control** — Opens a list of ActiveX controls for your selection. ActiveX controls can be broadly similar in capability to Progress SmartObjects, but they follow Microsoft's Component Object Model (COM) standard, not the ADM.



**Spin Buttons (ActiveX)** — Adds a Progress ActiveX Spin Button control to the workspace. Spin Buttons allow setting an INTEGER value. They are functionally similar to a hardware thumbwheel device. You can choose the upper and lower bounds of the range and the increment/decrement value.



**Combo Box (ActiveX)** — Adds a Progress ActiveX Combo Box to the workspace. See the description of the 4GL Combo Box for general details about this type of object.



**Timer (ActiveX)** — Adds a Progress ActiveX Timer object to the workspace. Timers are used as countdown devices that endlessly generate an event at an interval you determine. A timer has no run-time representation or associated data value.



**SmartDataObject** — Adds an ADM database interface to the workspace. A SmartDataObject manages a data stream to and from disk, filtering records and responding to requests from other ADM controls residing upstream.



**SmartObject** — Adds some SmartObject of your choosing to the workspace.



**SmartFolder** — Adds an ADM tabs widget to the workspace. A SmartFolder provides a well-defined context within which you can divide your total interface into functional units.



**SmartPanel** — Adds one of the predefined ADM button arrays to the workspace. SmartPanels are dedicated toolbars, providing upstream control of other SmartObjects.



**SmartDataBrowser** — Adds an ADM tabular data display to the workspace. SmartDataBrowsers provide a multi-record, row/column interface to the data stream.



**SmartDataViewer** — Adds an ADM data display to the workspace. SmartDataViewers provide a single-record interface to the data stream. You can arrange their layout to suit your design.



**SmartDataField** — Adds special capabilities of your choosing to a SmartDataViewer on a per-field basis.



**SmartFilter** — Adds an ADM Smart Query-By-Form capability to the workspace.



**SmartToolbar** — Adds an ADM menu and toolbar to the workspace. The default SmartToolbar offers the same navigation and update functions that are provided by the corresponding SmartPanels.



**SmartContainer** — Allows you to add an outer-level container object, such as a SmartWindow, to another container. This creates a child-parent relationship.

### 2.4.1 Placing Objects from the Object Palette

To select an object for placement, click on its tool icon (tool button):

- All objects have a default version. Some also have special-purpose variants. To list any variants available for the object under a tool icon, right-click that icon. To select one of the listed variants, left-click it. If there is only one choice, it will be listed as Default.
- To pick up a basic 4GL object for placement, left-click the icon. The icon will stay depressed to indicate that it remains selected. When you bring the cursor over a clear spot in the workspace, the cursor will change to indicate an object of the type you are placing.

  Left-click over the spot where you wish to release the object. The object will appear on the workspace at that position, and the icon and cursor will return to their usual states.
- To pick up a SmartObject for placement, left-click the icon:
  - For SmartObjects based on a master, a dialog box opens that allows you to select a master that you previously defined, or begin the process of defining a new master.
  - SmartObjects such as the SmartFolder, which is not based on a master, can be placed immediately, as though they were basic objects.

#### Multiple Placements

To place more than one object of the same kind without having to reselect the tool, left-click the icon a second time. For all SmartObjects based on a master, this second click must occur after you have selected the exact master you intend to use.

The image of a padlock will appear on the icon. This indicates that the object tool is latched (locked) down and will allow you to make as many placements as you like.

When you wish to release the lock on that tool, select the Pointer icon. The image of the lock goes away, and the button returns to the up position.

Locking a tool is easy to do inadvertently. If you find your cursor seems to be stuck, check for the lock icon. Clear it by selecting the pointer tool, and simply delete from the workspace any extra objects that the accidental lock caused you to add.

#### Changing Your Mind

If you change your mind in the middle of the placement process, you can cancel by selecting the Pointer tool. If you change your mind after placing the object, simply delete it from the workspace. There will be no residue left behind.

### 2.4.2 Object Palette Options

The Object Palette menu allows you to control the following options:

- **Top-Only Window** — Keeps the Object Palette on top of other AppBuilder windows.
- **Show Menu Only** — When checked, removes all buttons from the Object Palette. You must then use the Object Palette Menu for all interactions with the Object Palette.
- **Save Palette** — Saves the position, shape, and size of the Object Palette for future AppBuilder sessions.

### 2.4.3 Customizing the Object Palette

By editing or creating a *custom object file*, you can control how three AppBuilder interface components—the Object Palette, New dialog box, and the Choose dialog box—present objects for selection. You can specify which objects are included and how they are presented. You can select a custom object file by choosing Menu→ Use Custom. For more information about custom object files, see Appendix C, “Customizing AppBuilder”.

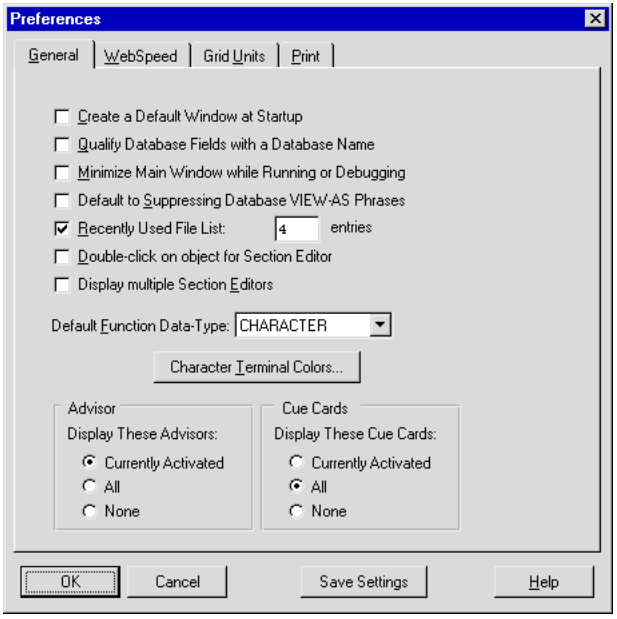
## 2.5 Setting Your Preferences

AppBuilder offers a number of properties that you can set to better suit your personal style and preferences without going through the effort of total customization. See Appendix C, “Customizing AppBuilder,” for information on more extensive customization.

You can configure both AppBuilder’s behavior and that of the built-in Section Editor.

### 2.5.1 AppBuilder Preference Options

To begin setting AppBuilder-specific preferences, choose Options→ Preferences. The dialog box opens at the General options tab:




You can set or clear as many of these options as you like:

- **Create Default Window at Startup** — Normally cleared. Setting this box causes AppBuilder to offer a brand new SmartWindow workspace for your use whenever it starts up. This might be especially convenient if you typically complete work on a particular window during an edit session and are ready for a fresh workspace next time. It is less useful if you normally resume work on an existing, partly completed window.

- **Qualify Database Fields with a Database Name** — Normally cleared. AppBuilder's output is application source code, including references to database fields. By default, such references are not fully qualified—any database that can supply the fields will be presumed to be the correct one. This can cause subtle difficulties in some circumstances. Setting this box causes AppBuilder to write out more complete references that are less prone to such errors.

- **Minimize Main Window when Running/Debugging** — Normally cleared. Setting this box causes AppBuilder's main window to go to an iconic state rather than stay open, when you start a test run or a debug session.

- **Default to Suppressing VIEW–AS Phrases** — Normally cleared. Setting this box causes AppBuilder to generate code without including widget specifications from the Data Dictionary. If you then need to make changes in representation, you need make them in only one place (the Dictionary) rather than two (Dictionary and source).

- **Recently Used File List** — Normally set with a default value of 4. As a convenience, the File pull-down menu includes menu items that point to the 4 most recently opened structured-procedure files. You can open one of the files again by selecting its menu item. If you would like more or fewer than 4 files to be listed here, change the number. If you prefer that no files be listed, clear the box.

- **Double-click for Section Editor** — Normally cleared. By default, double-clicking an object opens its Property Sheet. You can choose to open the Section Editor instead, by setting this box.

- **Display Multiple Section Editors** — Normally cleared. Setting this box allows you to have a separate Section Editor window open for each open workspace. By default, a single Section Editor window switches context according to which workspace has focus.

- **Default Function Data Type** — Normally CHARACTER. Unlike procedures, functions return a value upon termination and a call to them can, for example, appear on the right-hand side of an assignment operator. You can choose here the default data type of the value they return. It will apply to all functions defined thereafter for which you do not specify some other data type. The other choices are COM–HANDLE, DATE, DECIMAL, HANDLE, INTEGER, LOGICAL, MEMPTR, RAW, RECID, ROW–ID, and WIDGET–HANDLE.

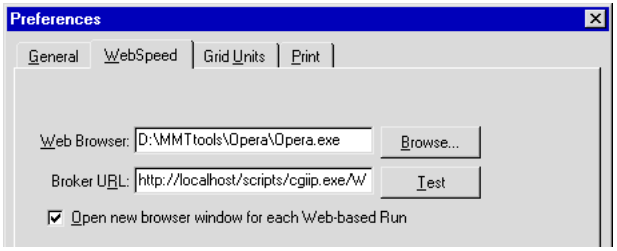
- **Character-Terminal Colors** — Choose this button to open the Color Selector dialog box. Use it to set the foreground and background colors for your character-window workspaces. See the section "Colors Selector and Editor" in Appendix A, "Frequently Used Dialogs."

- **Advisors** — Advisors are dialog boxes that appear from time to time to offer advice about how to complete certain processes. Working with AppBuilder, you will most often see Advisors appear to suggest—and offer to help with—creation of SmartLinks between SmartObjects.

- **Cue Cards** — Cue cards appear when you create SmartObjects. They offer a brief definition of the object, and hints on how to use it.

### WebSpeed Options Tab

If you have WebSpeed installed, you can use AppBuilder as your layout editor for both standard and Web applications. If you do not have WebSpeed installed, you will not see this tab. Your Preferences dialog box will only have the General, Grid Units, and Print tabs:




You can change any or all of the settings here, if the default values do not meet your needs:

- **Web Browser** — Enter the path to the browser you wish to use for WebSpeed development. By default, AppBuilder uses the path to the browser identified in the Registry. Choose the Browse button to open the standard Windows file system navigation tool, if you want to locate your preferred browser interactively.

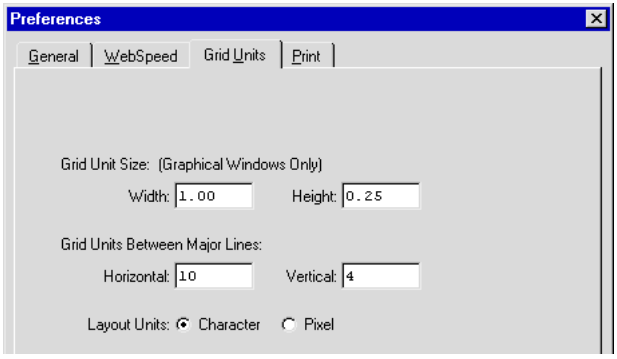
- **Broker URL** — Enter the URL of the WebSpeed broker you will use to run your Web objects. Choosing the Test button will display a test page of broker information if the URL you enter is a valid one.

- **Open new Browser Window** — Normally set. Clear this box to re-use a single browser window rather than generate new windows for each Web object. PSC recommends that you do **not** clear this box if using Netscape Navigator.

### Grid Options Tab

One of the hallmarks of good design is that things line up neatly. Graphic designers solved this problem by lining things up against a grid, originally printed on their workspace in a pale, cool-blue ink that was visible to the human designer but not to the platemaking camera.

AppBuilder provides a similar grid to help you lay out your widgets neatly in the application workspace. You can make it visible to you, but it is never visible to the users of your application:



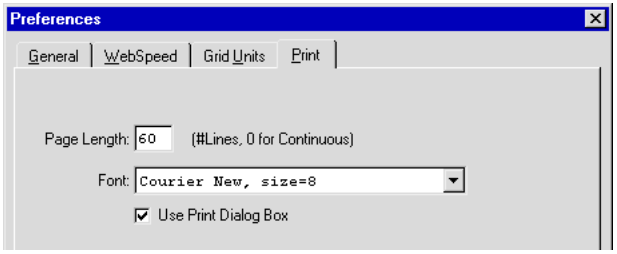

You can choose a grid spacing in this tab. It will apply only to graphical Frames; the character Window has a fixed size. If you plan to run your application in both GUI and character-cell contexts, you might wish to define your grid to be 1.0 x 1.0 characters in size, and take care not to consume more than 80 x 21 cells for any single layout.

Once you have a grid defined, choose Options→ Snap to Grid to cause your widgets to automatically jump to the closest grid intersection, if you release the widget nearby. Choose Options→ Display Grid to make the grid lines visible to you as a series of dots. The two options are independent of one another.

Note that any changes you make to the grid spacing here will apply only to Frame objects you create afterward, not to objects that you have already created. The Snap and Display menu item settings apply immediately to all existing and future workspaces, however.

### Print Options Tab

The Print options tab gives you some control over the way AppBuilder will print the source code it generates for you:




The Font and Page Length items are self-explanatory. If you clear the Use Print Dialog Box setting, AppBuilder will queue all subsequent print requests to the default printer immediately, without opening the Print dialog box.

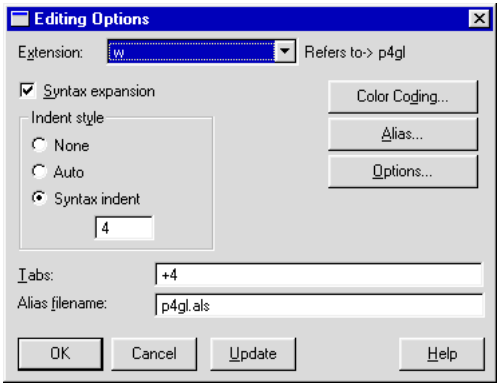
## 2.5.2 Section Editor Preference Options

The Section Editor allows you to set preferences on a per-language basis in three broad areas:

- Color coding of tokens
- Token completion and macro expansion
- Block formatting and indentation style

In certain cases, Section Editor applies your settings to individual blocks of language statements within a file, not merely to the whole file indiscriminately. Thus, you can (for example) embed a block of JavaScript code in a file that is otherwise HTML code, and Section Editor correctly color-codes and formats the embedded block and the surrounding blocks differently, each in the way you have specified for the particular language.

To begin configuring your Section Editor preferences, choose Options→ Editing Options. The Editing Options dialog box opens:




The Section Editor determines what settings to use by looking at the filename extension. The example shows default settings for Progress 4GL source file (.w extension). The notation "Refers to→ " to the right of the drop-down list indicates that changes will be written to a file common to several extensions, in this example to all extensions associated with the 4GL.

- **Syntax Expansion** — Normally set. Clearing this box prevents the Section Editor from completing partial tokens or expanding syntax macros. Example: when this box is set, if you type in **IF** and press **CTRL–SPACE**, the Section Editor inserts the rest of an IF template conforming to 4GL syntax.

- **Indent Style** — Determines the column where the new line starts after you press **RETURN**:

  – **None** — Line starts in left-most column.

  – **Auto** — Line starts in same column as preceding line.

  – **Syntax indent** — The Section Editor tries to determine where to start the line by taking current language syntax into account. The absolute integer field (default: 4) determines the amount of change for new levels of indenting.

- **Tabs** — You can specify either:

  – A tab interval by typing in an integer preceded by a + sign. For example, **+3**.

  – Actual tab stops by typing them in as space-separated, ascending integers. For example, **3 5 8 11 16 20 30**.

- **Alias Filename** — Enter the fully qualified filename where AppBuilder is to store alias (macro) definitions for the particular extension type. You can have as many alias files as you have languages defined. For Progress file extensions, the default aliases file is `p4gl.als`

- **Color Coding Button** — Choosing this button opens the Color Coding Setup dialog box, described in the "Color Coding Setup" section.

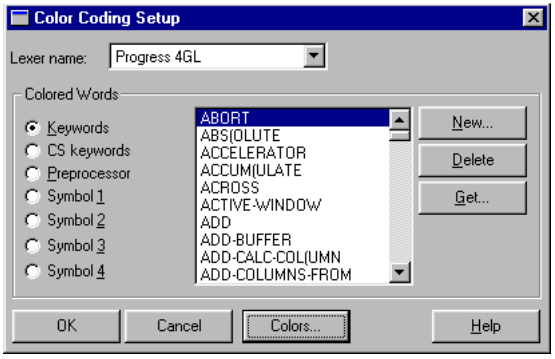
- **Alias Button** — Choosing this button opens the Alias Setup dialog box, described in the "Alias Setup" section.

- **Options Button** — Choosing this button opens a syntax-specific-formatting dialog box, described in the "Options" section.

Choose the Update button to make your changes to these settings permanent.

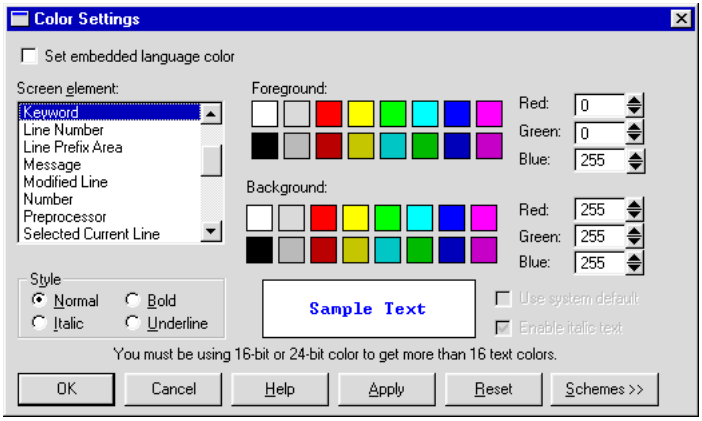
### Color Coding Setup

The Color Coding setup dialog box allows you to set the color for a particular class of token within a particular language. You can add new tokens to the most appropriate of several lists that AppBuilder maintains for the use of the recognizer:



Follow these steps to set color and style for some class of token:

1 ♦ Choose the Colors button. The Color Settings dialog box opens:




2 ♦ Check the Set Embedded language Color box, if appropriate.

3 ♦ Select the class of token in the Screen Element list for which you wish to make changes.

The meaning of two of the listed elements might not be immediately clear to you:

- Window Text is any text not otherwise defined.
- Attributes (HTML only) are recognized modifiers within an HTML tag. For example, in the tag <img src="somelocation">, “src” is an attribute.

Note that there are a few categories that, while still listed, are not settable in this version of the editor: Current Line, Cursor, No Save Line, Inserted Line, Line Number, and Message.

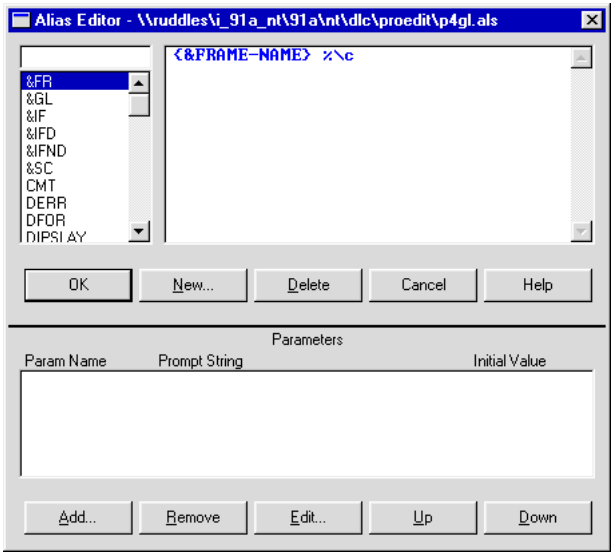
4 ♦ Click on color swatches to set foreground and background colors, and select the radio button for style. The sample text immediately reflects your choices.

5 ♦ Choose Apply to save your choices for the selected token class without also dismissing the dialog box.

6 ♦ When you have finished making changes, choose OK.

### Alias Setup

The Section Editor has a powerful macro-expansion capability. AppBuilder supplies predefined tokens that expand to common 4GL syntactic constructs, and using these tokens can reduce your typing effort considerably. The Alias Setup dialog box provides you the means to define additional tokens that the Section Editor will later expand in the same way:

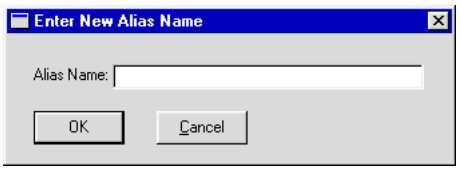



The expansion process is straightforward. For example, when you type **IF** and press **CTRL–SPACE**, the editor adds two spaces and the keyword THEN, and places the cursor between the two embedded spaces in exactly the right position for you to enter the IF test expression. Similarly, you can type in the predefined macro **&FR**, press **CTRL–SPACE**, and the editor will expand that to be {&FRAME–NAME}.

You can add additional items for automatic expansion, or even correction. If there is some typing mistake that seems to be a favorite of yours, you can enter it together with the relevant correction, and not have to worry about it any more. The transposition DIPSLAY for DISPLAY is an example of this—that error is such a popular one that PSC has already included it in the list.

Follow these steps to add a new expansion:

1 ♦ Choose New. The Enter New Alias Name dialog box opens:




2 ♦ Type in the token to be expanded and choose OK. The dialog box closes and the token appears in the list of tokens.

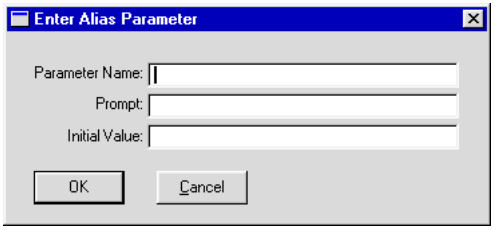
3 ♦ Type the expansion value into the editor. The expansion value may have multiple lines. Use **%\c** (percent sign, backslash, c) to represent the final position of the edit cursor, after expansion. Choose OK.

In addition to being able to define an alias that will place the cursor properly, you can define aliases that prompt for completion. For example, if you still do old-style debugging, using disclosure code, you might wish to define a disclosure message that would reveal the value of some variable at a certain point in your program. To do that follow these steps:

1 ♦ Follow Step 1and Step 2, above, calling the new alias PEEK.

2 ♦ Type in this expansion string: **MESSAGE "At %(loc) %(var) is %(val)".**

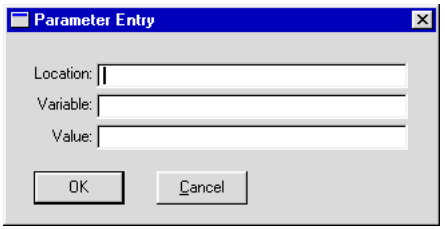
3 ♦ Choose the Add button in the bottom row. The Enter Alias Parameter dialog box opens:




4 ♦ Type in **loc** as the parameter name, and **Location** as the actual prompt string. You can leave Initial Value blank for this example, unless you do want some text as the initial value.

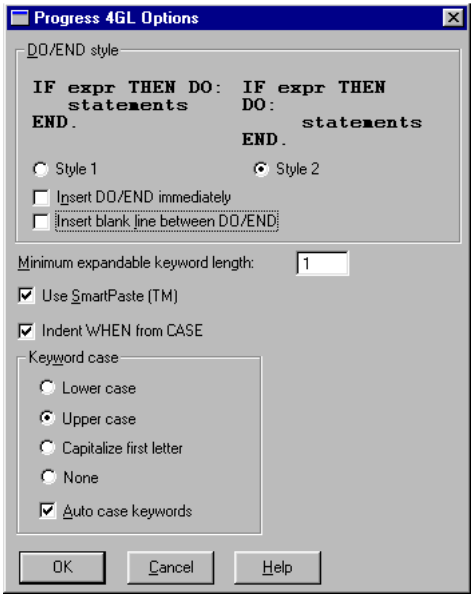
5 ♦ Repeat Step 3 and Step 4 for **var** and **val**, using **Variable** and **Value**, respectively, as the prompts. Choose OK.

6 ♦ Open the Section Editor in a context suitable for testing your new alias. Type in **PEEK** and press **CTRL–SPACE** The editor immediately opens a dialog box to prompt you for the three parameters, afterward inserting the fully expanded call to MESSAGE in place of PEEK:




### Options

The Options dialog box allows you to customize the appearance of your source code in various ways:




You can change as many of these values as you like:

- **DO/END Style** — Choose between having the DO block begin on the same line as the THEN token, or on the next line at the same indent level as the IF (default).
- **Insert DO/END Immediately** — Normally cleared. Setting this box makes the editor create every IF template with an attached DO/END block.

- **Insert blank line between DO/END** — Normally cleared. Setting this box the editor include a blank line when creating a DO/END block.
- **Minimum Keyword Length** — Choose the number of characters you must type (default: 1 character) before the editor will consider it a possible alias token. The editor will ignore **CTRL–SPACE** for any string with fewer characters than this.
- **Smart Paste** — Normally set. Clearing this box makes the editor ignore indentation when pasting. When Smart Paste is turned on, the editor attempts to correctly indent any text that you paste.
- **Indent WHEN from CASE** — Normally set. Clearing this box makes the editor put the WHEN clause at the same indent level as the enclosing CASE. The editor will normally indent WHENs by one additional level.
- **Keyword Case** — Normally all-caps. Choose the style the editor uses when expanding aliases.
- **Auto-case Keywords** — Normally set. Clearing this box makes the editor ignore differences between the Keyword Case setting and how you type keywords manually. When set, the editor changes your manual style to conform to your Keyword Case choice; for example, changing a lowercase keyword to all caps.

## 2.6 Workspaces: Design Windows and Source-code Editors

Designing an application using AppBuilder involves creating building blocks and then gluing them together with appropriate 4GL code. In Progress applications that are event-driven rather than procedure-driven, many of the building blocks will be user-interface related, and involve visual layout. Other building blocks have an internal and supporting role, and do not require you to lay out a visible representation.

The workspace for laying out visible objects is a design window. Figure 2–6 shows a design window for a SmartWindow—it is the SmartWindow itself. SmartWindows are outer-level organizers and members of the class SmartContainer.

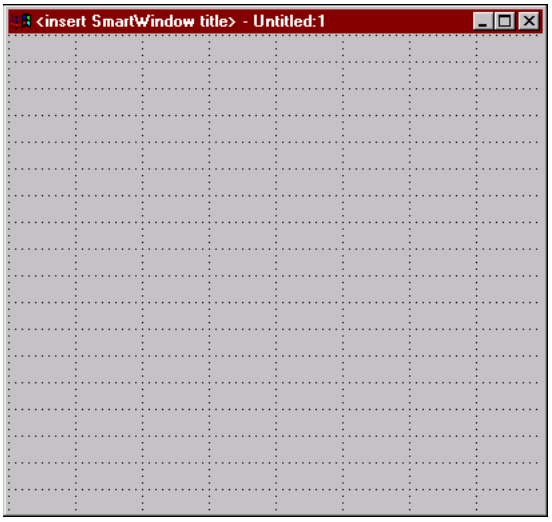



**Figure 2–6: Workspace for a Visible Object**

The workspace for non-visible objects is the built-in source-code Section Editor, together with a window that shows a tree view of the editable components. Figure 2–7 shows a tree-view window.

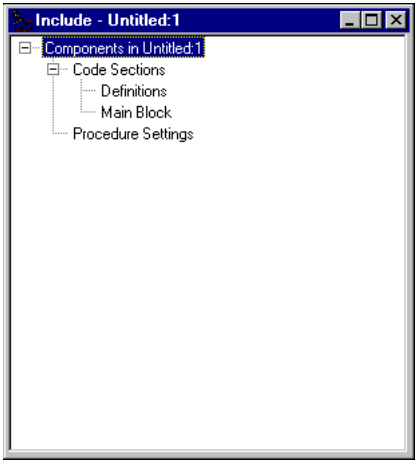



**Figure 2–7: Workspace Element for Nonvisible Object**

It is important that you use only the built-in Section Editor to maintain your AppBuilder objects.

All AppBuilder-generated procedure files conform to a specific format, which allows AppBuilder to read and update them. AppBuilder cannot reliably read and understand a file that does not conform to this format. Even apparently trivial changes—in some cases, adding a blank character—can render a file unopenable. For more information about editing AppBuilder-generated procedure files, see the "Editing Source Code Safely: The Section Editor" section in this chapter.

If you open in AppBuilder a procedure file that it did not generate, AppBuilder does not display a design window or tree view for it. You must maintain such files manually, so AppBuilder displays the file in a standard Progress Procedure Window.

## 2.7 Editing Source Code Safely: The Section Editor

The Section Editor allows you to edit individual code sections of procedure files without risk to later readability by AppBuilder. This is not true of any other editor, including the editor in the Progress Procedure Window.

### 2.7.1 Opening the Section Editor

To open the Section Editor, select the object you wish to edit and do one of the following:

- Choose the Window→ Code Section Editor menu item.
- Click the Code toolbar icon.
- Enter **CTRL–S** from the keyboard.

You can also double-click on the object if you have that preference item checked in the Options→ Preferences dialog box. Figure 2–8 shows the user interface elements visible when editing the Triggers section. The visible elements vary from section to section.

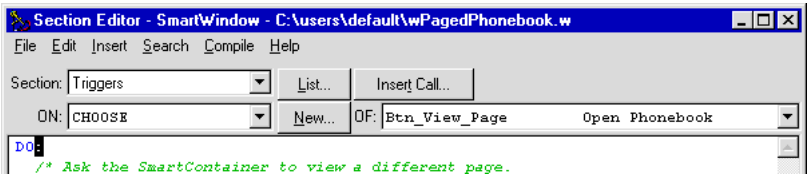
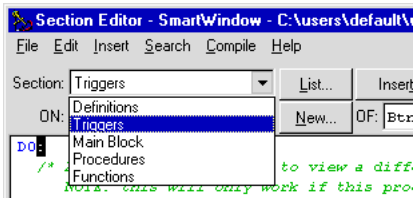



**Figure 2–8: Section Editor Physical Interface**

### 2.7.2 Choosing a Section to Edit

There are two ways to choose a particular code section for editing:

- Use the Section combo box to choose a section (Definitions, Triggers, Main Block, Procedures, or Functions). The figure below shows the section list opened:




If you select Triggers, Procedures, or Functions as the main category, you must also select an individual Trigger, etc., or choose the New button. Existing Triggers, etc. are listed in the combo box below the Sections list.

- Choose the List button to the right of the Sections combo box. The dialog box shown in Figure 2–9 will appear.

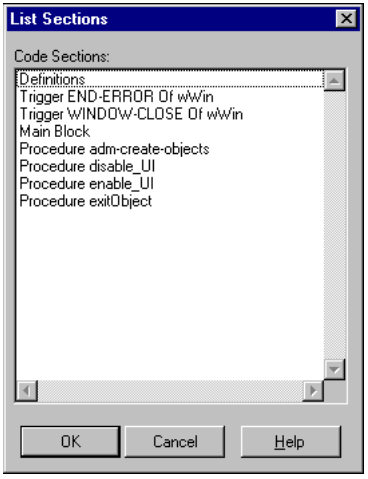



**Figure 2–9: Sections Listing Dialog Box**

This dialog box lists all the sections and subsections that you can edit, to the level of internal procedures and individual triggers. To make a selection, highlight the name and choose OK.

## 2.7.3 Using the Section Editor

The AppBuilder Section Editor is a capable editing utility that offers modeless text insertion, standard Windows-style cut-and-paste, plus:

- Text search-and-replace
- Transformation of selected text (indent/unindent, comment/uncomment)
- Token insertion by picking from lists
- Whole-file insertion
- Undo of the most recent action, for most actions
- The ability to delete whole subsections

### Cutting, Copying, and Pasting Text

To cut a block of text, highlight the block of text, then choose Edit→ Cut (shortcut: **CTRL–X**). The Section Editor removes the block of text from the current editing area and places it in the system clipboard.

To copy a block of text, highlight the block of text, then choose Edit→ Copy (shortcut: **CTRL–C**). The Section Editor copies the block of text and places it in the system clipboard.

Once a block of text has been cut or copied into the system clipboard, you can paste it at any current cursor location by choosing Edit→ Paste (shortcut: **CTRL–V**).

### Finding and Replacing Text

Within the current editing area, you can search for and replace text strings. To search for a text string, choose Search→ Find (shortcut: **CTRL–F**). The Find dialog box appears, as shown in Figure 2–10.

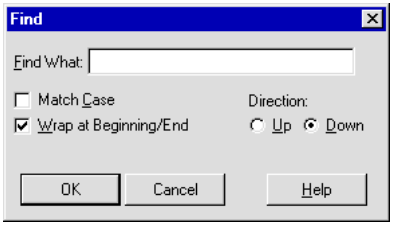



**Figure 2–10: Find Dialog Box**

In the Find What field, type the text string that you want to find. Set search options as needed:

- Activate the Match Case toggle box to make the search case-sensitive.
- Activate the Down radio button to search forward; activate the Up radio button to search backward.
- Activate the Wrap at Beginning/End toggle box to specify that upon reaching the beginning or end of the current editing area, you return to the top or bottom and begin to search again.

When you choose OK, the Section Editor searches for the first instance of the text string.

To find the next instance of the text string, choose Search→ Find Next (shortcut: **F9**). To find the previous instance of the text string, choose Search→ Find Previous (shortcut: **SHIFT-F9**).

To replace instances of the text string with a different string, choose Search→ Replace (shortcut: **CTRL–R**). The Replace dialog box appears, as shown in Figure 2–11.

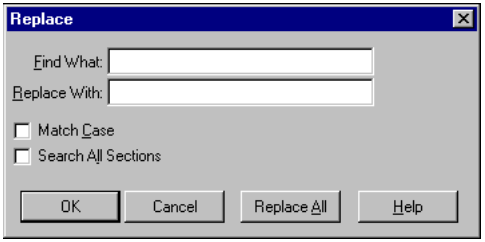



**Figure 2–11: Replace Dialog Box**

The following search and replace features are available:

- **Find What** — Enter the substring to be searched for.
- **Replace With** — Enter the replacement substring.
- **Match Case** — Activate this toggle box to match the search target's case.
- **Search All Sections** — The default search-and-replace is local to the section being edited. To search globally within the current file, check this box.
- **Replace All** — Click on this button to replace all occurrences of the target substring without further confirmation.

  When you use the Replace All option, a message box displays indicating the Replace All is complete and the number of occurrences that were replaced.

### Transforming Text

A block of selected text can be indented, unindented, commented out, or uncommented by choosing Edit→ Format Selection, and then choosing the operation desired.

### Inserting Text from Lists

The process of creating code involves entering many reserved words, identifiers, filenames, and similar tokens. The possibilities for error are numerous. To reduce error and, in most cases, save keystrokes, AppBuilder helps you insert text strings for these token types:

- Database fields
- Event names
- Internal procedure calls
- Preprocessor names
- Queries
- Object names
- Filenames

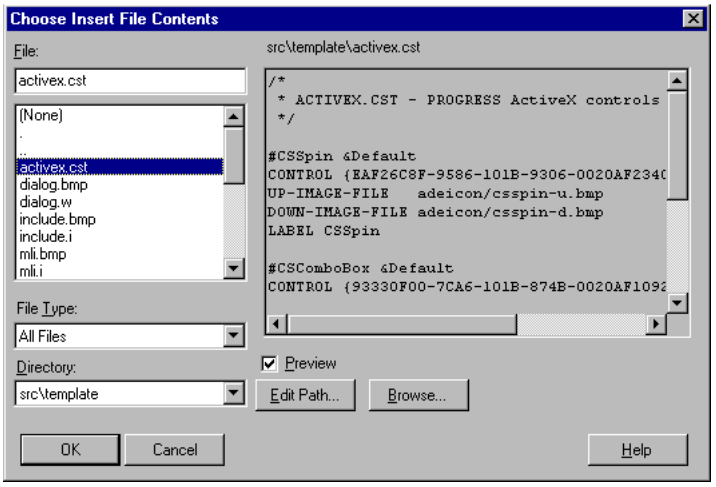
For each of these items, AppBuilder provides a way for you to choose or specify a text string that AppBuilder inserts into the edit buffer at your cursor location. You can access this capability by right-clicking in the editing area, or by choosing from the Insert menu.

For detailed instructions, see the "Inserting Text-string Tokens from Lists" section in this chapter.

### Inserting Whole Files

You can insert (merge) the contents of a whole text file into the edit buffer at the current cursor location. This is convenient if you have used another editor to write some code that you now wish to integrate into your AppBuilder file.

To insert the contents of a text file, choose Insert→ File Contents. The following dialog box opens:




**NOTE:** Apart from the dialog box title shown by the title bar, and the end result, this is the same dialog box that appears when you choose Insert→ Filename. Make sure of what you are inserting.

By default, this dialog box points to the `src/template` directory and lists the header and superprocedure files found there. You can list different files, or files in a different directory, by selecting from the choices in the respective drop-down lists. You can edit the list of directories by choosing Edit Path, and you can add a directory without editing by selecting a file using the Browse button, which invokes the standard Open File dialog box.

### Saving Your Changes

Since the Section Editor is a feature of AppBuilder rather than a separate utility, it has no Save option of its own. To save your changes, use the File→ Save or File→ Save As menu items in the AppBuilder main window. This will save all changes made to the current object, not just those you made using the Section Editor.

### Undoing Your Changes

To undo all unsaved changes made to the current editing area, choose Edit→ Undo All. This restores the file to the state it was in when you entered this editing area.

To undo only your most recent unsaved change, choose Edit→ Undo. This option undoes your last edit. If you use Section Editor features such as Insert or Paste, Undo undoes the most recent action only. Successive Undo operations switch between redoing and undoing the previous edit.

### Deleting Whole Sections

To delete all the non-AppBuilder code in a particular subsection (an individual Trigger, Procedure, etc.), choose Edit→ Delete Trigger (Procedure, etc.) The menu item text varies according to the element you are currently editing.

**CAUTION:** You cannot Undo the deletion of a whole subsection, such as a whole Trigger. If you use the Edit→ Delete *subsection type* function, and then change your mind, you will have to re-enter all the deleted code. It cannot be recovered by using Undo or Undo All.

## 2.7.4 Inserting Text-string Tokens from Lists

### Database Field Names

To insert the name of a database field into an editing area, choose Insert→ Database Fields from the Section Editor menu or choose the Section Editor's pop-up menu by right-clicking in the editing area. The Field Selector dialog box appears, as shown in Figure 2–12.

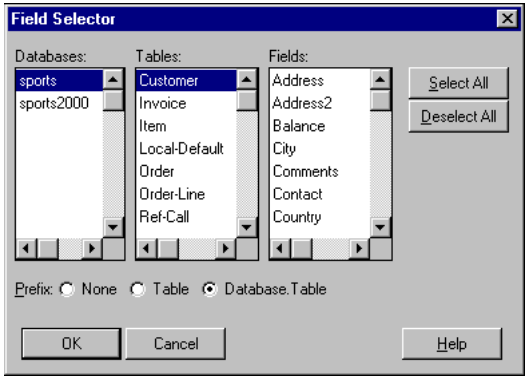



**Figure 2–12: Field Selector Dialog Box**

This dialog box lets you choose any field for any currently connected database. Follow these steps to specify a database field:

1 ♦ Select a database. The Tables and Fields lists change.

2 ♦ Select a table. The Fields list changes again.

3 ♦ Select the field names that you want to insert in the current editing area:

- The Select All button selects all of the fields for the current table.
- The Deselect All button deselects all of the currently selected fields.
- To select multiple fields but not all of them, hold down the **CTRL** key while selecting.

4 ♦ Specify a prefix (qualifier) for the field name:

- The None radio button prefixes nothing to the field name.
- The Table radio button prefixes the table name to the field name.
- The Database.Table radio button prefixes both the database and table names to the field name.

5 ♦ Choose OK to insert the fields.

AppBuilder inserts the field names at the cursor location in the current editing area or replaces selected text.

### Event Names

To insert event names in an editing area, choose Insert→ Event Name. The Choose Event dialog box appears, as shown in Figure 2–13.

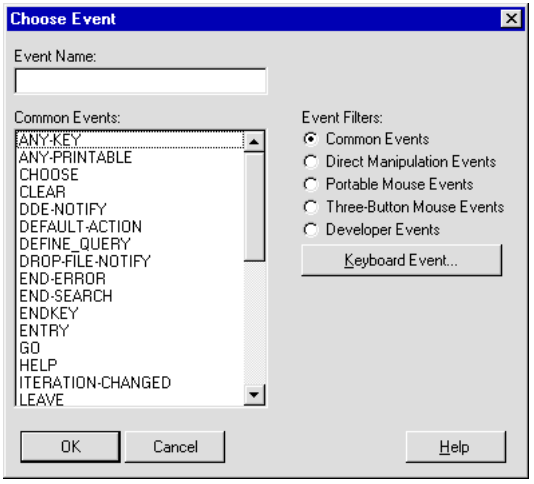



**Figure 2–13: Choose Event Dialog Box**

This dialog box displays a list of all events recognized by the Progress 4GL. To choose a listed event, select it and click on OK:

- Use the Event Filters radio buttons to view different types of events.
- To specify a keyboard event, choose the Keyboard Event button.

### Procedure/Function Calls

To insert a call to an internal procedure or function or to a SUPER procedure's internal procedure or function in an editing area, choose Insert→ Procedure Call. The Insert Procedure Call dialog box appears, as shown in Figure 2–14.

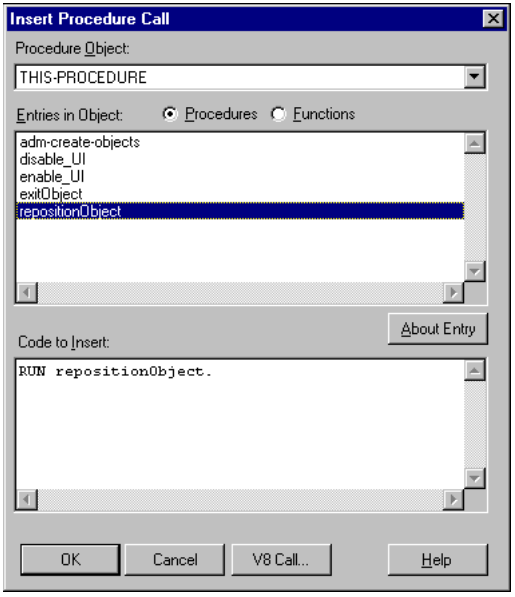



**Figure 2–14: Insert Procedure Call Dialog Box**

Table 2–9 identifies and describes the options available on the Insert Procedure Call dialog box.

**Table 2–9: Options on the Insert Procedure Call Dialog Box** *(1 of 2)*

| Option | Description |
|---|---|
| Procedure Object | Displays the name of the procedure object that is active in the Code Section Editor as indicated by the entry THIS–PROCEDURE, any SmartObjects, or super procedures defined for the object. A drop-down list is available from which you can choose a different procedure object.<br><br>To edit a Web object, choose web–utilities–hdl from the drop-down list to access the internal entries of the Web utility. The web–utilities–hdl is a procedure handle defined for all Web objects that reference event procedures, method procedures, and API functions of the Web utility object, `web-util.p`. |
| Entries in Object | When either the Procedures option or the Functions option is selected, this option displays a list of names of the current Procedure Object's procedures or functions, respectively. The procedure call code that you can insert into the Code Section Editor from this list displays in the Code to Insert field.<br><br>If procedures or functions are for actively running SmartObjects and super procedures, the insert code includes any run-time parameters for that procedure or function. You might have to replace each parameter name with the appropriate expression (that is, a constant, field name, variable name, or expression) for the specific call in your code. The data type of each parameter displays in comments.<br><br>For procedures and functions in the THIS–PROCEDURE object, the dialog box does not provide the run-time parameters. You can determine the appropriate run-time parameters by switching to that procedure or function in the Section Editor and reviewing any parameter definitions. |
| About Entry button | If clicked, displays additional information about the procedure or function currently highlighted.<br><br>If the current procedure object is a SmartObject and it has an internal procedure named object-help, choose About to run that procedure in the object, passing the current procedure name as an INPUT parameter. The object can provide procedure-specific help.<br><br>If the object has no object-help procedure, the Insert Procedure Call dialog box runs AppBuilder online help for the selected procedure. |

**Table 2–9: Options on the Insert Procedure Call Dialog Box** *(2 of 2)*

| Option | Description |
|---|---|
| Code to Insert | Displays the code that you can insert into the Code Section Editor's edit area. You can edit this code before inserting it into the Section Editor. |
| V8 Call button | Allows you to access the Insert Procedure Call - Version 8 dialog box to insert Version 8 style ADM RUN, DISPATCH, and NOTIFY procedure calls into code. You might need to use this dialog box if you maintain Version 8 style SmartObjects in the Version 9 AppBuilder.<br><br>For more information about the Version 8 functionality associated with the Insert Procedure Call - Version 8 dialog box, see the Progress Version 8 documentation. |

Use the Procedure Object drop-down list to select the appropriate ADM object. This allows you to call the procedures and functions for that super procedure. Figure 2–15 shows the procedures for `containr.p`.

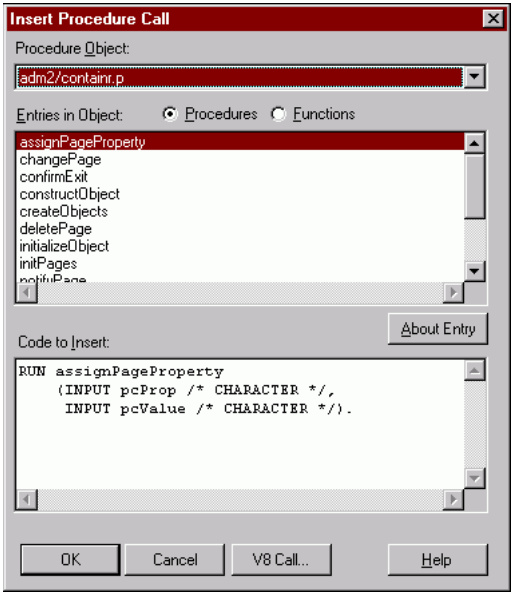



**Figure 2–15: Listing Insertable Procedures**

The Procedure Object field lists all of the procedure handles (THIS–PROCEDURE or SmartObject instances) referred to in the current procedure file. For a standard window procedure file, THIS–PROCEDURE is the only procedure handle. For a SmartContainer, each SmartObject instance has a handle that is listed, in addition to THIS–PROCEDURE, which is listed for the SmartContainer itself. When you choose a procedure handle in the Procedure Object field, all of the internal procedures or functions listed pertain to the procedure associated with the object.

**NOTE:** AppBuilder does not keep track of parameters needed for internal procedures in THIS–PROCEDURE. It does this only for procedures and functions in SmartObject instances used by THIS–PROCEDURE. If you insert a RUN statement for a procedure defined in THIS–PROCEDURE, you must either check to see if the correct parameters are passed or wait for the Progress Compiler to do so.

### Preprocessor Names

To insert Preprocessor names in an editing area, choose Insert→ Preprocessor Name. The Preprocessor names are listed under its tab in the Code References window, as shown in Figure 2–16.

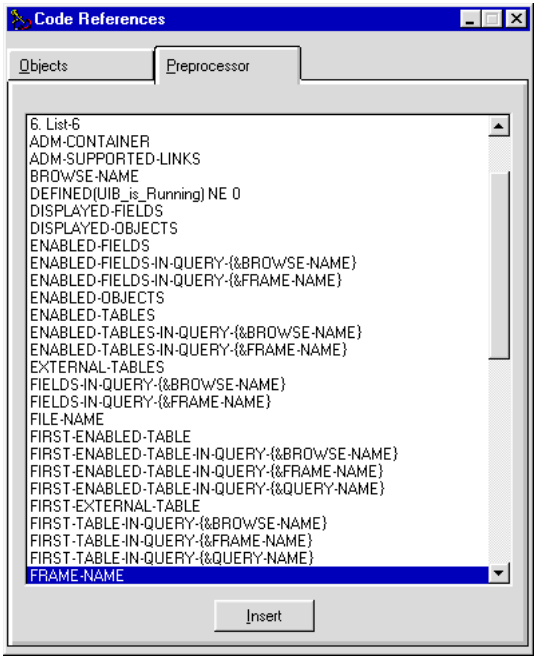



**Figure 2–16: Preprocessor Names Dialog Box**

To select a Preprocessor name, click on the name and choose OK. For more information about Preprocessor names, see the online help.

### Queries

You can use AppBuilder's query-building dialog boxes to generate a query definition rather than typing in the query. To do this, choose Insert→ Query. The Query Builder dialog box appears, as shown in Figure 2–17.

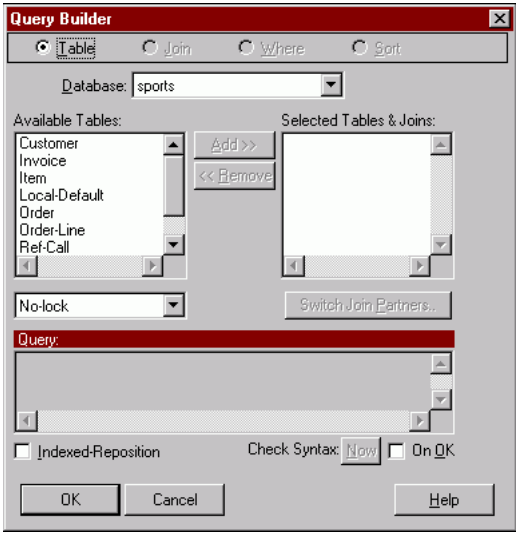



**Figure 2–17: Query Builder Dialog Box**

This dialog box is the first in a series of dialog boxes that help you generate a query. For more information about building queries in AppBuilder, see the section "Defining and Editing Queries Using Query Builder" in Chapter 4, "Data-access Objects."

When you finish generating the query, choose OK. The Section Editor inserts the entire text of the query at the cursor location in the current editing area.

### Object Names

To insert object names in the current editing area, choose Insert→ Object Name. The Code References window appears, open to the Object Names tab, as shown in Figure 2–18.

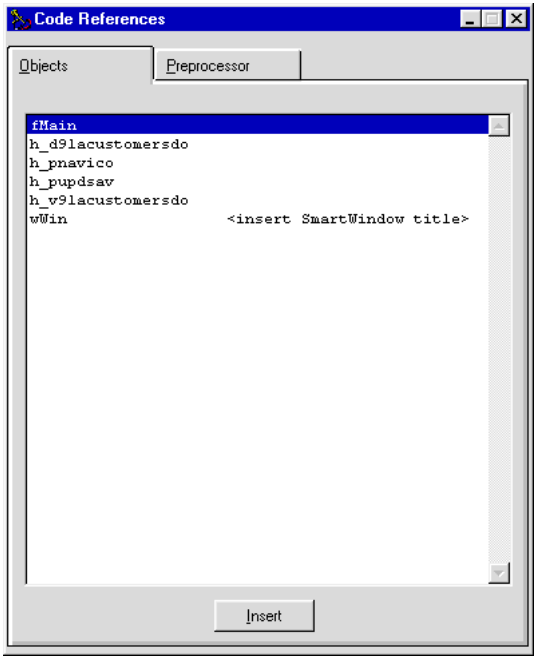



**Figure 2–18: Object Names Dialog Box**

This dialog box displays the 4GL object names of all widgets and SmartObject instances available in the procedure file:

- The left column contains the name of each widget and SmartObject instance.
- The right column contains the label or title of each widget.

To insert an object, click on it and choose OK. To insert more than one, hold down the **CTRL** key while selecting. The Section Editor will insert your choice at the cursor.

### Filenames

To insert a file's pathname in the current editing area, choose Insert→ File Name. The Choose Insert File Name dialog box appears, as shown in Figure 2–19.

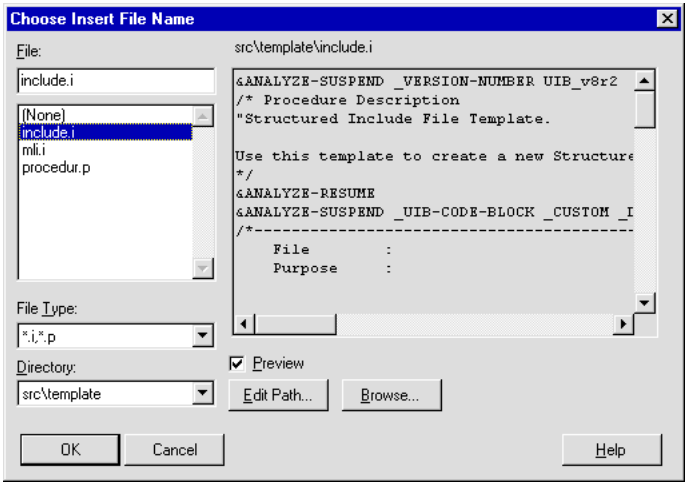



**Figure 2–19: Choose Insert File Name Dialog Box**

**NOTE:** This is the same dialog box that appears when you wish to insert the contents of a file rather than only a filename. Make sure of what you are doing

To insert the file's pathname, select the filename and choose OK. The Section Editor inserts the file's pathname at the current cursor location in the editing area. By default, this dialog box displays all of the .i and .p files in the `src/template` directory.

## 2.8 Configuring Look-and-Feel: Object Properties

AppBuilder supports application building blocks that differ significantly in their style of construction, sophistication, configurability, and functional role. Because of this, it is not possible for AppBuilder to present their properties to you in a single, unified way. You can find properties aggregated in several different ways:

- File-level property sheets, available by choosing Tools→ Procedure Settings or the corresponding toolbar icon. Building blocks that exist as external procedures—such as container objects and ADM SmartObjects—will have file-level properties.
- Property sheets for SmartObjects, available by choosing Properties on the menu-button menu. To reach the Instance Properties sheet, select that menu item instead, or click on the Edit button in the main property sheet.

- SmartLinks sheet, available by choosing the button with the chain-link icon in the property sheet for a SmartObject. The sheet lists all current links to or from that object.
- Property sheets for basic widget instances, available by selecting the object and choosing Tools→ Property Sheet.
- Certain object properties, such as size and position, can be examined and set by choosing Window→ Properties Window. The exact properties visible in this window vary with the object. If you have more than one object selected, only properties common to all are displayed, and any changes affect all selected objects as well.
- Property sheets for ActiveX (OCX) objects, available by double-clicking the object representation.

For the properties that apply to a given type of object, see the discussion of that object type.

## 2.9 Inspecting a Procedure File

When you read a procedure file into AppBuilder, it compiles the file in order to analyze its contents, and stores relevant information about the file in memory. This represents a snapshot of the procedure as it appears at compilation time. As you make changes to the procedure, AppBuilder updates this snapshot.

At any time in an AppBuilder session, you can examine the current state of a procedure file as it exists in AppBuilder's memory. To do this, choose Compile→ Code Preview from the AppBuilder main menu. AppBuilder builds a readable version of the procedure file and places it into a content browser, the Code Preview dialog box, as shown in Figure 2–20.

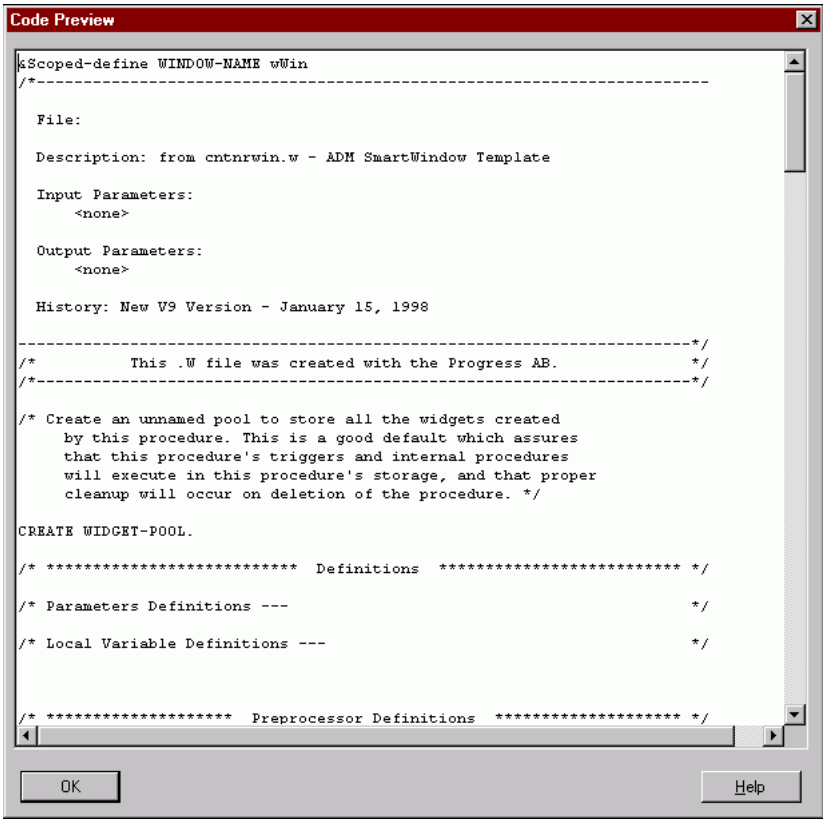



**Figure 2–20: Code Preview Dialog Box**

The Code Preview dialog box shows the current state of the design window as 4GL code. You can copy code from the Code Preview dialog box and paste it into the Section Editor.

Follow these steps to copy and paste text:

1 ♦ In the Code Preview dialog box, highlight the text you want to copy.

2 ♦ Press **CTRL–C** to copy the highlighted text.

3 ♦ In the Section Editor, move the text cursor to where you want to insert the text.

4 ♦ Press **CTRL–V** to paste the text.

You can also search for text strings in the file. To do this, press **CTRL–F**. AppBuilder displays the Find dialog box, where you can specify the string you want to search for. To search forward, press **F9**; to search backward, press **SHIFT-F9**.

## 2.10 Test-driving Your Design

At any time during an AppBuilder session, you can test the current design window workspace. When you run a design window, you are in fact running a temporary procedure file. If you read a procedure file into AppBuilder and make changes to the design window, the temporary file reflects those changes. However, until you save the procedure file, AppBuilder does not store the changes permanently. When you save your changes, AppBuilder overwrites the procedure file and preserves the changes. If you prefer to preserve the original state, do a Save As instead, or discard your changes by exiting without saving.

To run a design window, choose Compile→ Run, or choose the Run button, or press **F2**. Running the design window hides all windows except AppBuilder's main window, and disables all of AppBuilder's functionality except for a Stop button. AppBuilder displays the Stop button (eight-sided, red; it resembles the traditional US 'STOP' road-traffic sign) in place of the Run button, as shown in Figure 2–21.

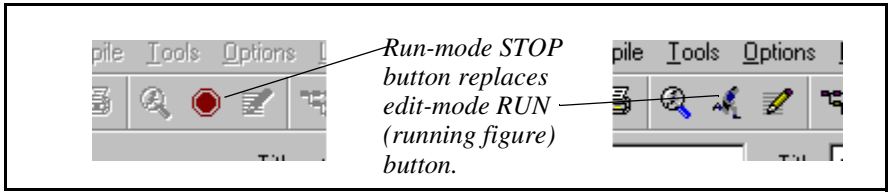



**Figure 2–21: Stop Button**

Choosing the Stop button stops the design window even if the window is, for example, caught in an endless loop. Choosing the Stop button always closes the run window and re-enables AppBuilder's editing functionality.

## 2.11 Saving Your Work

The development mode in which you are using AppBuilder affects how it saves objects. If you are using the remote mode, AppBuilder saves files to your Web server or transaction server machine. If you are using the local mode, AppBuilder saves files to local or networked drives of your choice. For more information on remote development mode, see the *Getting Started with WebSpeed* manual.

To save a procedure file, choose the Save button or File→ Save from the AppBuilder main menu. If the file is new, the Save As dialog box appears. This dialog box allows you to specify a name for the file. If you have previously saved the file, AppBuilder automatically saves the file using its current name. If you want to save the file using a different name, choose File→ Save As.

# 3

# Organizer Objects

Organizer objects provide the visual and sometimes the execution context for other objects. Organizer objects include Windows, Dialog Boxes, Frames, and even Rectangles.

This chapter discusses the following topics:

- Basic Window and SmartWindow
- Basic Dialog and SmartDialog
- Basic Frame and SmartFrame
- Simple SmartContainer
- SmartFolder
- Character Window
- Character Dialog Box
- Rectangle
- SmartContainer tool

## 3.1 Introduction to Organizer Objects

Whenever you develop a Progress application, you use organizer objects to provide a context for other objects. When you create a GUI Window and populate it, the widgets you use seem more related than they would without the Window around and under them. The Window is providing that sense of visual and purposeful organization. If you set off part of the Window's *client area* with a Frame or Rectangle, the widgets you put inside the Frame/Rectangle will then seem more closely related to one another than to other widgets in the same overall Window. The Frame/Rectangle is acting as another level of organizer.

Besides providing visual/psychological integration, some organizer objects have a functional role as well. SmartWindows, for example, have built-in code to create and initialize the SmartObjects they contain, and more code to maintain the SmartLink communication pathways between the contained SmartObjects.

Part of AppBuilder's advantage is that it allows you to treat organizer objects (except for basic Frames and Rectangles) as design-window workspaces. Using techniques similar to those of an interactive graphics editor, you can assemble a complete window's worth of building blocks in the workspace and then execute the workspace.

AppBuilder currently offers seven types of organizer in the File→ New list:

- SmartWindow
- SmartDialog
- SmartFrame
- Simple SmartContainer
- Basic Window
- Basic Dialog
- Character Window
- Character Dialog

In addition, AppBuilder offers three types of component-level organizer in the Object Palette:

- SmartFolder
- Basic Frame
- Basic Rectangle

## 3.2 Organizer Object Roles

Each type of organizer object has different characteristics and capabilities. They are only superficially interchangeable, if at all, and you should choose accordingly.

The SmartObjects are the most capable, and you must use them if you are building ADM-compliant applications. They all incorporate the logic needed to integrate other SmartObjects, and offer considerable savings in programming time. They also are the only organizer objects that support paging. The major differences between them are:

- SmartWindows are general-purpose outer-level Windows, and include a default Frame. They are not modal, they can have a menu bar, and they can be used as the main window of your application.
- SmartDialogs appear to be specialized Windows, but technically are specialized Frames, instead. They are modal, they cannot have a menu bar, and they cannot be used as the main window of your application. You typically open them to capture specific information and then close them again.
- SmartFrames are Frame objects that can support component SmartObjects and act as a communication pathway. To display them at run time, you must always place them in a SmartWindow.
- SmartFolders are a special type of component objects. Conceptually, they are most closely related to Frames, in that they are component objects with an organizing role. They use ADM paging to implement the tabbed-file-folder metaphor. You can place them in SmartWindows, SmartDialogs, or SmartFrames.
- Simple SmartContainers are general-purpose Smart organizers that offer the functionality of a SmartWindow without the actual Window part. They are invisible at run time. Because they are low-overhead objects, they can be very useful for modules that can run unattended.

The basic Window and Dialog objects do not have ADM Smart logic. Because they lack this supporting logic, you cannot place SmartObjects in them. You can place other objects in them, such as basic Frames and field-level objects, and ActiveX objects. Apart from the differences in how Smart they are, Windows and SmartWindows serve comparable functional purposes, as do Dialogs and SmartDialogs.

Basic Frame objects are essentially static organizers. Their role is to provide a visual context for the field-level objects they contain. Additionally, Frame objects allow you to define the order in which pressing the **TAB** key will traverse their contents.

Rectangles are also static organizers, used to provide a visual context for other objects. Unlike Frames, Rectangles do not create a traversal group; their role is entirely visual.

The SmartContainer tool—not to be confused with the Simple SmartContainer—creates no object. AppBuilder provides the SmartContainer tool as a convenient way for you to associate independent organizer objects of the class SmartContainer, such as SmartWindows, with one another in parent/child relationships.

The balance of this chapter discusses the available organizer objects in more detail.

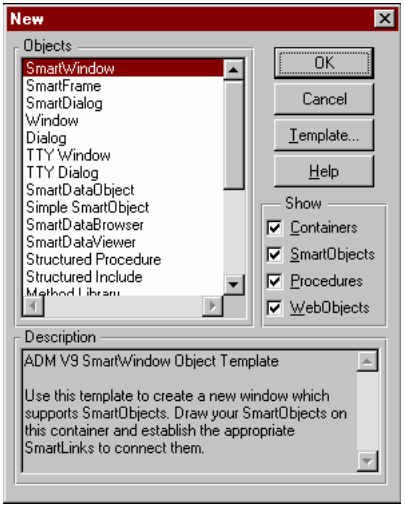
## 3.3 Creating a New Procedure-based Organizer Instance

This section uses the SmartWindow as the example, but the process is the same for most types of procedure-based organizers. To create a different kind of object, select that kind from the list instead of selecting a SmartWindow. To create a component-level organizer—a basic Frame or Rectangle, or a SmartFolder—follow the steps in the appropriate section.

To create a SmartWindow workspace, follow these steps:

1 ♦ In the AppBuilder main window, choose File→ New or the New button on the toolbar.

The New dialog box appears:




This dialog box contains the following user-interface elements:

- **Objects** — Lists the available types of files. The objects in this list all exist as external procedures on disk. Note that there also might be file types listed, such as Structured Include, that do not represent objects.

- **Template** — Invokes the Template dialog box, which allows you to preview a procedure template file. When you choose a template, AppBuilder adds it to the Objects selection list. Adding a procedure to this selection list allows you to open it as an untitled procedure.
- **Show** — Specifies which procedure types you want to see in the Object selection list. Depending on your license, the options in this area might be different than those shown here.
- **Description** — A description of the currently highlighted object type.

2 ♦ Select SmartWindow from the Object selection list, then choose OK.

AppBuilder loads a template as an untitled procedure and opens a workspace on your screen. For visible objects such as a SmartWindow, the workspace is the actual window you will populate. In other cases, such as when you select a Structured Procedure file, Structured Include file, or Method Library, AppBuilder opens the Section Editor as the workspace.

3 ♦ Adjust the size and position of the new SmartWindow as needed.

Click on the title bar to select the SmartWindow, and move it to a convenient place on your display. Resize the SmartWindow by dragging its edges.

## 3.4 SmartWindows

SmartWindows are general-purpose, outer-level organizer objects. They have ADM Smart technology embedded in them, and can fully integrate other SmartObjects such as SmartDataObjects, SmartToolbars, and so forth. SmartWindows are members of the SmartContainer class. Unlike the other SmartContainer objects, a SmartWindow can serve as your application's main window or as one of the windows in a multi-window design.

Note that a SmartWindow comes fitted with a Frame that covers the client area of the Window. The Frame should automatically resize as you resize the SmartWindow. You can also resize it separately, if you choose. You might wish to do that in order to have room for additional Frames. Besides providing the required context for widget placement, Frames also group widgets for traversal using the **TAB** key, and you might wish to define more than one such group. Note that, rather than re-sizing the default frame, you could instead add a new Frame for each separate traversal group, overlaying the default frame.

SmartWindows are file-level (external procedure) objects. Unlike the objects embedded in the Progress 4GL itself, objects defined by external procedures typically are complex. Such objects have a set of properties associated with their nature as a subprogram module, and additional sets—if they have a visible run-time representation—associated with whatever widgets they put up as their user interface. The sections below discuss these different properties and their meanings.

### 3.4.1 Configuring SmartWindow Visual Properties

Configuring the visual properties for your new SmartWindow can be simple or complex, depending on how well the default settings suit your purpose.

Whenever you create a SmartWindows object using AppBuilder, you actually create two visible elements: a Window and a Frame. The Frame completely covers the client area of the Window, and resizes itself when you resize the Window. Because of that, it may not be at all obvious to you at first that you are dealing with two different elements or how to open the Properties dialog box for the underlying Window rather than for the Frame that covers it.

Both the Window and Frame elements have properties you may wish to set. This section describes the properties that apply to the Window part. See the "Configuring SmartFrame Visual Properties" section in this chapter for a discussion of properties unique to Frames.

#### Opening the Property Sheet for the Underlying Window

Perhaps the most common way to open the property sheet for any object is to double-click on its client area, if you have your preferences so set. But the overlying Frame will prevent you from doing that here: double-clicking will open the property sheet for the Frame, not the Window. To open the Properties dialog box for the Window part of the SmartWindow object, use one of these methods:

- While holding down the **CTRL** key, click in the client area of the SmartWindow until you see that you have the Window selected (its identifier will display in the Object field of AppBuilder's main window). Then choose Tools→ Property Sheet.

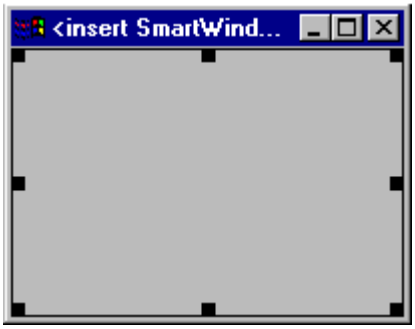
  Although this method is to be preferred, you may find it requires several clicks if the window is not empty.

- Expose some of the underlying Window. Then double-click on the area you exposed.

**CAUTION:** You should not leave bare Window surface exposed unnecessarily, since there is a slight possibility that you might then unintentionally place an object partly outside the Frame, causing it to disappear.

To expose some of the underlying Window, follow these two steps:

1 ♦ Select the Frame part of the SmartWindow by clicking in the client area (the area below the title bar). A border with handles will appear, as shown in this figure:




2 ♦ Drag the bottom-center handle upward a bit. This exposes the surface of the underlying Window widget, though there might be little or no visible difference.

Once you have the Window selected, opening the property sheet will cause the dialog box shown in Figure 3–1 to appear.

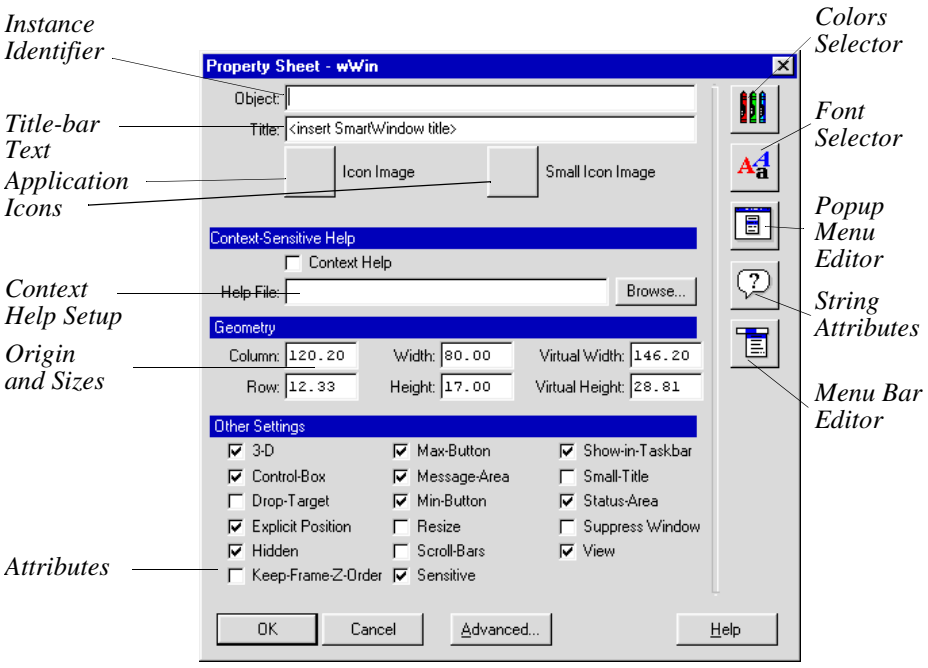



**Figure 3–1: Window Property Sheet**

### Minimal Configuration Changes

Although you can choose to accept many default settings, you should at least ensure that each SmartWindow you create is uniquely identifiable. To individualize a SmartWindow, make these changes:

- Change the generic instance identifier to one that more closely describes the window instance you are creating. The instance identifier is the token that will appear in the source code, and is the default value for the filename. Note that Progress identifiers for Window objects are conventionally prefixed with a w.
- Change the generic title-bar text to a string that identifies the particular Window, or the application if the Window you are creating is its main window.
- Click each of the icon image buttons and identify the icon bitmap (.ico) files for the large (32x32) and small (16x16) icons that will be used to represent the Window.

### Miscellaneous Properties (Other Settings)

There are a number of changes to a Window's appearance or behavior that you can make, if the default settings do not suit your purposes. They are listed here:

- **3D** — Normally set. Clearing this box changes the color of the Window's client area, but has no other obvious effect.
- **Control-Box** — Normally set. Clearing this box clears and disables the Max-Button and Min-Button check boxes. The run-time effect is to remove all three buttons from the right end of the Window's title bar and the small icon from the left end. Only the title string remains.
- **Drop-Target** — Normally cleared. Setting this box causes the Window to generate an event when some object is dragged and dropped onto it. You can write code to intercept and respond to such events under program control.
- **Explicit Position** — Normally cleared. Setting this box enables the Column and Row settings in the Geometry section, and initializes them to the current location of the upper-left corner of this Window. You can then freely reposition the Window during design time, and the Window will start up in the location you choose rather than in some location determined by the operating system.
- **Hidden** — Normally cleared. Setting this box prevents this Window from displaying itself in response to implicit requests. Explicit code must be used to display a HIDDEN Window. If initializing this SmartWindow will take a noticeable amount of time, keeping the window hidden meanwhile might be less worrisome to your customer, as long as you display a progress indicator.
- **Keep Frame Z-Order** — Normally set by AppBuilder when generating code. Prevents nested frames from inappropriately changing their layering in response to getting the input focus.
- **Max Button, Min Button** — Normally set. Clearing these boxes eliminates the buttons that appear at the right end of the title bar. Those buttons allow the user to make the window go to a full-screen or iconic state, respectively.
- **Message Area** — Normally cleared. Setting this box adds an area for messages at the bottom of the Window, but above any Status Area. Character windows always have a Message Area.

- **Resize** — Normally cleared. Setting this box allows the user to resize the Window by dragging its edges. Widgets in the client area do not respond automatically to this resizing in any way.

- **Scroll Bars** — Normally cleared. Setting this box allows scroll bars to be automatically activated as needed.

  Setting the Scroll Bars option gives meaning to the Virtual Width/Height values shown in the Geometry section. Scroll bars exist to move different portions of some notional space into and out of view as desired. The size of the area currently in view is reflected in the Width and Height numbers, shown in character units in the main property sheet and pixel units in the Advanced Properties. The total size of the logical area, both visible and scrolled-off, is shown by the Virtual numbers. Those dimensions are referred to as Virtual rather than Total because they could conceivably represent a much larger area than could be displayed whole by any physical device. Unless the window can be scrolled, however, the total and visible areas are effectively identical, making the Virtual numbers meaningless for practical purposes.

- **Sensitive** — Normally set. Clearing this box causes the Window to decline the input focus when offered.

- **Show in Taskbar** — Normally set. Clearing this box causes the Window to iconize to a position above the taskbar rather than within the taskbar itself.

- **Small Title** — Normally cleared. Setting this box has the same effect as clearing Control Box, plus it reduces the height of the title bar to the minimum needed for the title.

- **Status Area** — Normally cleared. Setting this box adds a status area at the bottom of the Window.

- **Suppress Window** — Normally cleared. Setting this box effectively removes the Window widget, leaving only the Frame. Frames are displayed by default in the Current Window, which might be a scratch window created for the purpose.

- **View** — Normally set. Useful only when the Frame is HIDDEN.

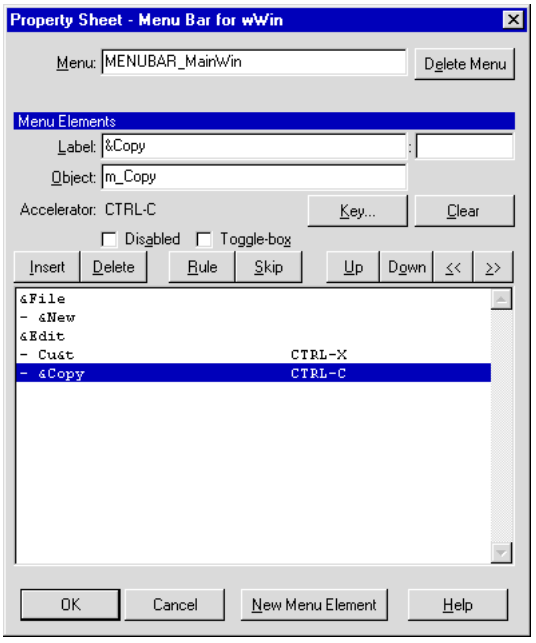
### Menus

Unlike SmartDialogs and SmartFrames, a SmartWindow can have a Menu Bar and pull-down menus. It can also have a pop-up menu. You create both types of menu using the same editor and the same procedure.

**NOTE:** You cannot use this menu if you plan to place a SmartToolbar, even if you only intend to use the toolbar element of that object. The built-in menu and the SmartToolbar are incompatible. See the description of the SmartToolbar object in the *Progress ADM 2 Guide*.

To create a Menu Bar and pull-down menus, follow these steps. The procedure for creating a pop-up menu is identical, apart from the button you choose at the start:

1 ♦ Choose the Menu Bar button. The editor shown here opens:




2 ♦ Change the default identifier for the menu object to reflect your naming conventions, if desired.

3 ♦ Enter the label for the first menu item. A corresponding identifier will immediately be generated in the Object field. Modify that identifier, if desired:

- You must precede one of the letters in each label with an ampersand (&) so the user can choose the menu item from the keyboard. The choice of letter must uniquely identify the menu item within its group (bar or pull-down). If you have not determined where this item will reside, you can defer selecting a character until later.
- By convention, a menu item on a pull-down ends in an ellipsis (...) if choosing it will open another level of pull-down or invoke a dialog box. Example: Save As...
- If you wish to define string attributes for the label, enter them in the area to the right of the label. For further information, see the "String Attributes" section in Appendix A, "Frequently Used Dialogs."

4 ♦ Repeat Step 3 until you have entered all the items that will appear anywhere in the menu system you are creating.

5 ♦ Select each menu item that is to appear in a pull-down rather than on the bar. Mark it by choosing the >> button. You will see a dash appear in front of the item. The >> button can be used more than once, and each use will prepend another dash. The number of dashes indicates the nesting level of the pull-down on which the item will appear:

- No dashes means the item will be on the menu bar itself.
- One dash means the item will be on the main pull-down menu.
- Two dashes means the item will be on a cascade/flyout menu off the pull-down.
- Three dashes means the item will be on a second level of cascade menu.

If you change your mind, remove unwanted dashes by choosing the << button.

6 ♦ Select each menu item and use the Up and Down buttons to move it into the correct position relative to the other items. The first item on a pull-down menu should appear immediately below the item that causes the pull-down to open. Example: in the figure shown in Step 1, the Cut item would be the first item in the Edit pull-down.

If the item is a pull-down item, and is to appear with a check mark in front of it whenever selected, check the Toggle-box option.

7 ♦ Arrange pull-down items into logical groups by inserting rule lines (Rule button) and nonselectable blank items (Skip button) as appropriate.

8 ♦ Select each item that will be frequently used and define an accelerator (shortcut) key for it. Choose the Key button and, when the capture dialog appears, **press the actual key** or combination (such as **CTRL-X** for Cut) that is to become the shortcut.

**NOTE:** To create a shortcut for an item on the menu bar, you must first change it to a pull-down item using the >> button. After you have defined the shortcut, move it back to the menu bar using the << button. The shortcut will remain defined.

9 ♦ Select each item that will not be active on startup and check the Disabled box. Save your work.

### Context-sensitive Help

If you plan to provide context-sensitive help, check the Context Help box and specify the full pathname of the relevant help file.

### String Attributes, Window Color, Status Area Font

You can set several other characteristics of the Window, if you choose:

- String attributes.
- Font used for text written to the Status Area, if any.
- Colors used for the Window's foreground and background. By default, MS-Windows applications leave their window colors undefined so that the end user can set them as a preference item.

See the online help system for further information.

### Advanced Properties

Choose the Advanced button to set additional properties. The Advanced Properties dialog box shown in Figure 3–2 will appear.

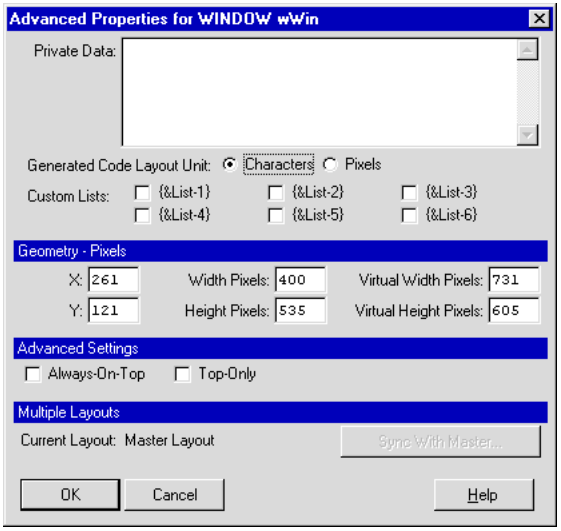



**Figure 3–2: Window Advanced Properties Dialog Box**

The Advanced Properties shown here are common to most objects. You can modify any that do not meet your needs:

- **Private Data** — Not used by AppBuilder. AppBuilder assigns the contents of this field to the Window's PRIVATE–DATA attribute. You can write code to read and operate on that value in any way that meets your needs.

- **Generated Code Layout Unit** — Determines whether AppBuilder uses character or pixel units when generating source code for this object.

- **Custom Lists** — Custom Lists would be called macros in other languages such as C. They provide a convenient way to avoid typing a large number of identifiers when many objects are to be treated identically. For example, if you check the box for List 1, the identifier for this Window is associated with List 1. Later, any time a reference to this procedure's List 1 is encountered by the preprocessor, the identifier for this Window—together with the identifiers for all other objects you associated with List 1 in this procedure—will be substituted in the source code in place of the reference. You can choose to include this Window in up to six such macros.

  You can also give the macros themselves more meaningful identifiers; see the "Custom Lists Naming" section in Appendix A, "Frequently Used Dialogs."

- **Geometry** — You can set the exact startup size and position for this window, in pixel units. Startup position is only meaningful when you set the Explicit Position box.
- **Always on Top, Top Only** — You can set either of these options, but not both. Neither is set by default:
  - Top Only prevents this Window from being occluded by other Windows belonging to this Progress session.
  - Always on Top prevents this Window from being occluded by **any** other Window, even those belonging to other applications.
- **Sync With Master** — Forces this layout to take on the attributes of the master layout. This pushbutton is disabled unless this layout is an alternate layout. For more information on alternate layouts, see Appendix B, "Multiple Layouts"."

## 3.5 SmartDialogs

SmartDialogs are special-purpose, outer-level organizer objects, technically a type of Frame with a dedicated, inseparable window. They are members of the SmartContainer class, and can integrate other SmartObjects. They differ from SmartWindows in that they:

- Cannot serve as the main window of your application.
- Cannot have a menu bar.
- Cannot be resized at run time or scrolled.
- Cannot generate or accept SmartLinks of type Filter.
- Can accept other widgets placed directly on them (that is, without your having to interpose a separate Frame).

Because they are always modal, they must be dismissed before any other window can gain focus, so the number of roles they can play is not large. You will typically want to use them to capture user input without which your program cannot continue.

### 3.5.1 Configuring SmartDialog Visual Properties

Many of the properties of a SmartDialog are the same as those of a SmartWindow. The principal differences are:

- There is nothing related to resizing, scrolling, or virtual versus actual size, since the size of a SmartDialog is fixed at design time.
- There is no provision for defining a menu bar, since a SmartDialog is limited to pop-up menus.
- You can invoke the Tab Editor if you wish to define a special **TAB**-traversal order, since a SmartDialog is technically a Frame and maintains a traversal list. For information about setting traversal order, see the "Traversal-Path Editor" section in Appendix A, "Frequently Used Dialogs".

To open the property sheet, shown in Figure 3–3, double-click on the client area of the SmartDialog, or choose Tools→ Property Sheet.

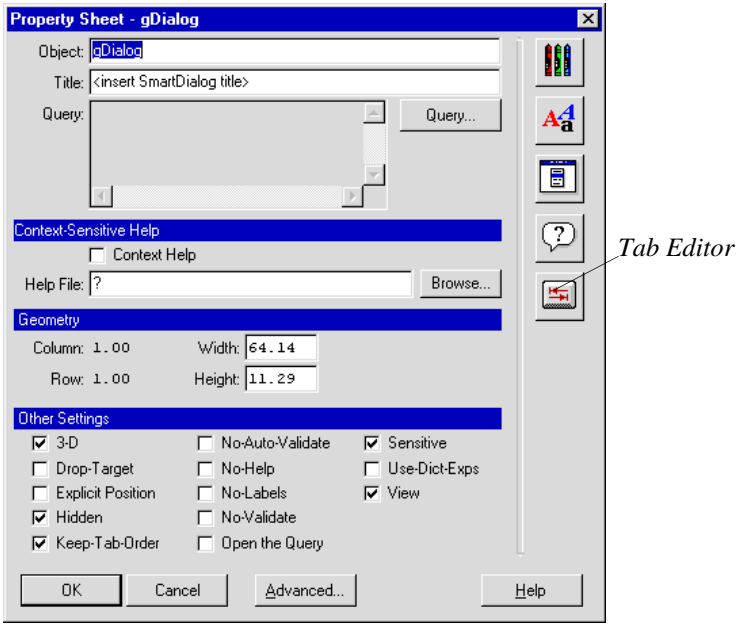



**Figure 3–3: SmartDialog Property Sheet**

### Minimal Configuration Changes

Although you can choose to accept many default settings, you should at least ensure that each SmartDialog you create is uniquely identifiable. To individualize this SmartDialog, make these changes:

- Replace the generic instance identifier with one that more closely describes the Dialog instance you are creating here. The instance identifier is the token that will appear in the source code, and is the default value for the filename. Note that Progress identifiers for Dialog objects are conventionally prefixed with a g.
- Change the generic title-bar text to a string that identifies this particular Dialog.

### Miscellaneous Properties (Other Settings)

Dialogs have fewer miscellaneous properties than Windows do, but those they have in common have the same meaning in both contexts. Dialogs also have a few properties in common with Frames:

- **No Auto Validate** — Normally cleared. Setting this box turns off, for all field-level objects in the dialog box, the automatic validation defined in the Data Dictionary.
- **Open the Query** — Normally cleared. Setting this box automatically populates the fields in this dialog box, if there is a query defined.
- **Use Dict Exps** — Normally cleared. Setting this box makes all the validation expressions and help strings defined in the Data Dictionary available to all the widgets assigned to this SmartDialog.

### Advanced Properties

To open the Advanced Properties dialog box shown in Figure 3–4, choose the Advanced button.

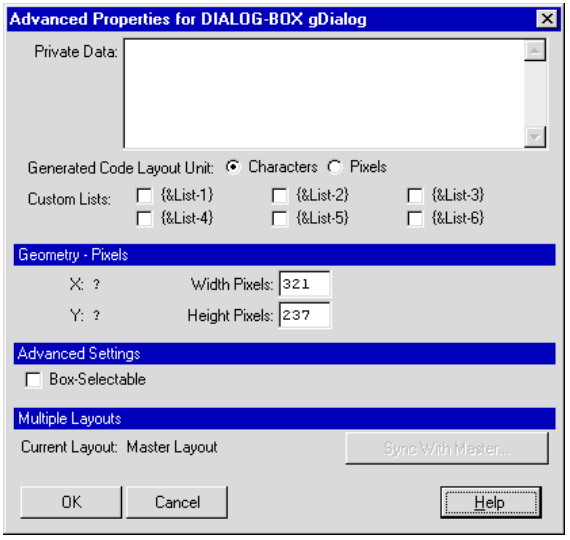



**Figure 3–4: SmartDialog Advanced Properties**

The only differences in Advanced Properties between a Window and a Dialog are:

- Dialog boxes are always modal, so there is no option to make them always appear in front of other objects. Modal objects always appear in front of other objects.
- Dialog boxes have an option to allow box-selection, normally cleared. Box-selection is sometimes called marquee-selection or lasso-selection. It refers to selecting more than one object by dragging a dynamically generated box (the marquee) around them.

## 3.5.2 Configuring SmartDialog Procedure Settings

The SmartDialog uses the standard Procedure Settings dialogs. See the "Procedure Settings" and "Advanced Procedure Settings" sections in Appendix A, "Frequently Used Dialogs" for more information.

## 3.6 SmartFrames

SmartFrames are special-purpose organizer objects. They are members of the SmartContainer class and can integrate other SmartObjects.

Every SmartFrame must be supported by a SmartWindow or SmartDialog; they are not independent or outer-level objects. The principal benefit SmartFrames offer is reusability. If you have a self-contained layout that includes SmartObjects, and will be reused repeatedly but would not be appropriate to implement as a SmartDialog, consider implementing it on a SmartFrame.

### 3.6.1 Configuring SmartFrame Visual Properties

SmartFrames are based on the Frame widget. To open the property sheet for the Frame widget component, double-click the client area of the SmartFrame, or choose Tools→ Property Sheet. The dialog box shown in Figure 3–5 will open.

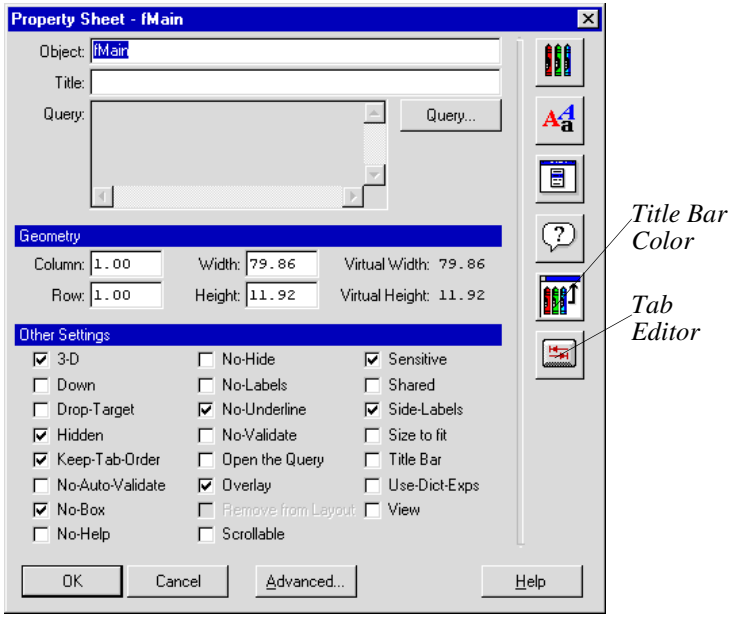



**Figure 3–5: SmartFrame Property Sheet**

### Minimal Configuration Changes

Although you can choose to accept many default settings, you should at least ensure that each SmartFrame instance you create is uniquely identifiable. To individualize a SmartFrame, make these changes:

- Replace the generic instance identifier with one that more closely describes the Frame instance you are creating. The instance identifier is the token that will appear in the source code, and is the default value for the filename. Note that Progress identifiers for Frame objects are conventionally prefixed with an f.
- If your new Frame will have a visible title bar, change the generic text to a string that identifies the purpose of the Frame.

### Miscellaneous Properties (Other Settings)

You can make a number of changes to a Frame's appearance and behavior, if the default settings do not suit your purposes. Some properties are discussed in the sections on SmartWindows and SmartDialogs. Listed here are the properties unique to Frames:

- **Down** — Normally cleared. Setting this box allows as many frame iterations (logical lines of output) as will fit to be written to the Frame before pausing. This option has no meaning if the Frame is not being used for direct output.
- **No Box** — Normally set. Clearing this option causes the Frame to display with a visible border.
- **Title Bar** — Normally cleared. Setting this box clears the No Box option and causes the Frame to display with a border and a prominent, visible title bar. You can set the background and foreground colors of the title bar (choose the Title Bar Color button), though the settings have no effect when the application is running under MS-Windows.

### Advanced Properties

You can make several changes to a Frame's behavior or appearance in the Advanced Properties dialog box. To open the dialog box, shown in Figure 3–6, choose the Advanced button.

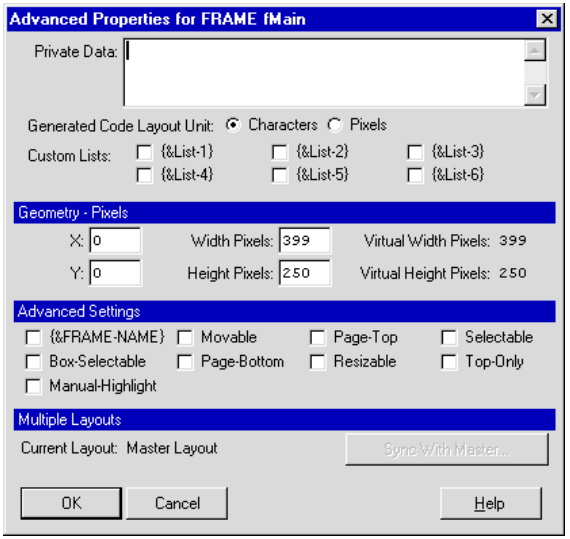



**Figure 3–6: SmartFrame Advanced Properties Dialog Box**

Some properties are discussed in the sections on SmartWindows and SmartDialogs. Listed here are the properties unique to Frames:

- **{&FRAME-NAME}** — Normally cleared.
- **Manual Highlight** — Normally cleared. Setting this box allows you to use special highlighting for certain objects.
- **Movable** — Normally cleared. Setting this box allows the user to move the Frame within the enclosing space (Window, Dialog, or other Frame).
- **Page Top, Page Bottom** — Normally cleared.
- **Resizable** — Normally cleared. Setting both this box and the Selectable box allows the user to resize the Frame, within the boundaries of the enclosing object.
- **Selectable** — Normally cleared. Setting this box allows the user to select and, if Resizable is also set, resize the Frame.

### 3.6.2 Configuring SmartFrame Procedure Settings

The SmartFrame uses the standard Procedure Settings dialogs. See the "Procedure Settings" and "Advanced Procedure Settings" sections in Appendix A, "Frequently Used Dialogs" for more information.

## 3.7 Basic Windows

Basic Windows are general-purpose, outer-level organizer objects. Basic Windows are SmartWindows without the ADM Smart technology. They are external-procedure objects, and can be used as the main window in your non-ADM applications, but they cannot support SmartObjects.

For information about configuring your Basic Window objects, see the "SmartWindows" section in this chapter. There are only two differences in default properties:

- Basic Windows have the Keep Frame Z-Order property set; SmartWindows do not.
- Basic Windows do not initialize in a Hidden state; SmartWindows do.

## 3.8 Basic Dialogs

Basic Dialogs are special-purpose, modal organizer objects. Basic Dialogs are SmartDialogs without the ADM technology; they cannot support SmartObjects.

For information about configuring your basic Dialog objects, see the "SmartDialogs" section in this chapter. There is only a single difference in default properties:

- Basic Dialogs have the Open Query property set; SmartDialogs do not.

## 3.9 Basic Frames

Basic Frames are nonwindow, component organizer objects. You can place SmartObjects on a basic 4GL Frame, even if it is nested in other basic Frames, as long as the outer-level object that supports them all is a SmartWindow or SmartDialog.

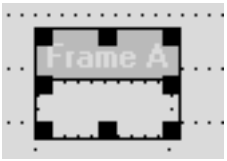
### 3.9.1 Creating and Placing a Frame

Because Frames are basic 4GL objects, creating and placing one is quite easy. Follow these steps:

1 ♦ Click on the Frame tool icon in the Object Palette:



2 ♦ Move the mouse cursor over a bare spot in your workspace and click to place the new object:




3 ♦ Configure it, and adjust its size and position appropriately.

### 3.9.2 Configuring Frame Properties

For information about configuring your basic Frame objects, see the "SmartFrames" section in this chapter. Basic Frames are SmartFrames without the ADM Smart technology.

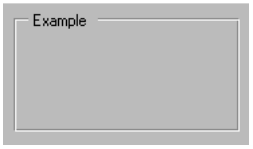
## 3.10 Rectangles

Rectangles are the simplest organizer objects. Their role is entirely visual. Rectangles can be nested within Frames and can be placed within or partly within (overlapping) other Rectangles. You can place SmartObjects on a Rectangle as long as there is an appropriate Smart organizer (SmartWindow, SmartDialog, or SmartFrame) to support them.

By default, a Rectangle appears as a figure with a closed, sunken border 2 pixels thick:



By combining a Text widget with a Rectangle, you can create an object similar in appearance to an MS-Windows Frame widget:




There is no line object in the 4GL, but by carefully sizing, positioning, and coloring one or two Rectangles, you can create the illusion of a line object. This can be convenient as a visual spacer.

### 3.10.1 Creating and Placing a Rectangle

Because Rectangles are basic 4GL objects, creating and placing one is quite easy. Follow these steps:

1 ♦ Click on the Rectangle tool icon in the Object Palette:



2 ♦ Move the mouse cursor over a bare spot in your workspace and click to place the new object:



3 ♦ Configure it, and adjust its size and position appropriately.

### 3.10.2 Configuring Rectangle Properties

Rectangles are basic objects, defined in the 4GL itself. Not being objects defined by external procedures, they do not have Procedure Settings. Their properties are all contained in property sheets.

To open the property sheet for a Rectangle, select the Rectangle and choose Tools→ property sheet. The dialog box shown in Figure 3–7 will open.

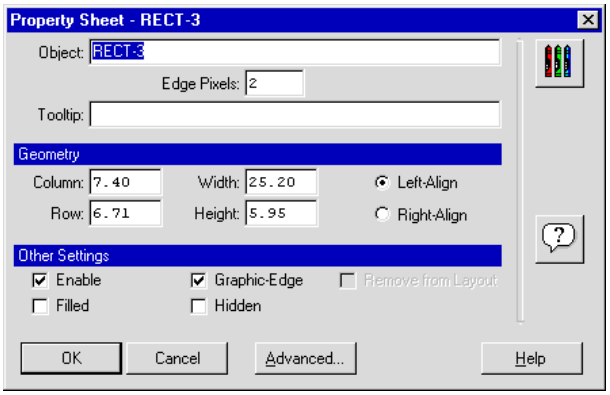



**Figure 3–7: Rectangle Property Sheet**

### Minimal Configuration Changes

You should change the default identifier, shown in the Object field, to a token that more accurately identifies the role of this Rectangle in your design.

### Border Width (Edge Pixels)

Rectangles can be very useful as a way to visually define a special area, for example one whose appearance will change because of paging. You might wish to make such an area even more visually special by changing the thickness of the border. This Rectangle shows a distinctive, eight-pixel border:



To change the thickness of the border, enter the new value in the Edge Pixels field. The results of specifying too thick a border are undefined.

A value of zero (0) will cause the Rectangle border to disappear, effectively making the Rectangle itself disappear unless it has a fill of a contrasting color.

### Geometry

Normally, the location of a widget is calculated at its upper-left corner. You can set the location of a Rectangle to be its upper-right corner instead, if you wish, by setting the appropriate radio button.

Additionally, by setting the width or height of a Rectangle to zero (0) pixels in the Advanced Properties dialog box, you can create the illusion of a line.

### Miscellaneous Properties (Other Settings)

Rectangles are such simple objects that they have very few properties. You can make changes to those few, if the default settings do not meet your needs:

- **Enable** — Normally set. Clearing this box causes the Rectangle to decline input focus when offered. For this option to be meaningful, you must also set either the Movable or Resizable property. For more information, see the "Advanced Properties" section.
- **Filled** — Normally cleared. Setting this box fills the Rectangle with the background color.
- **Graphic Edge** — Normally set. This box interacts with the Edge Pixels (border thickness) setting:
  - Specifying an Edge Pixels value of three or less causes a line-drawing character to be used for the border in character environments. Values over three clears this box and a block character is used for the border in character environments.
  - Clearing this box sets the Edge Pixels value to eight and a block character is used for the border in character environments.
- **Hidden** — Normally cleared. Setting this box prevents the Rectangle from displaying itself in response to implicit requests. See the online help for further information.

### Advanced Properties

You can make several changes to a Rectangle's behavior or appearance in the Advanced Properties dialog. To open the dialog box, shown in Figure 3–8, choose the Advanced button.

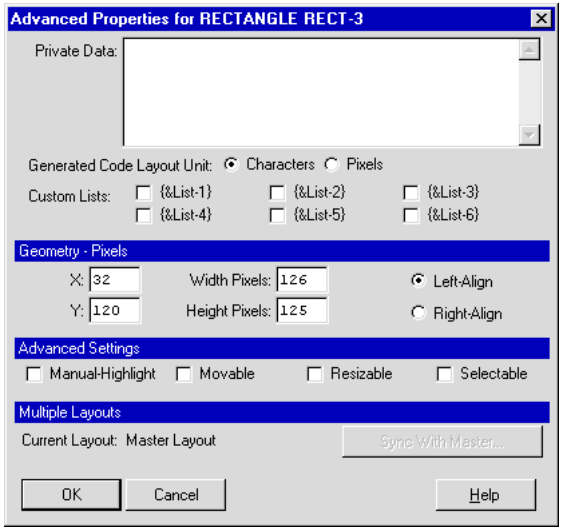



**Figure 3–8: Rectangle Advanced Properties Dialog Box**

Four properties you may wish to modify are:

- **Manual Highlight** — Normally cleared. Setting this box allows you to define and use a special highlighting scheme when the object is selected.
- **Movable** — Normally cleared. Setting this box plus the Selectable and Enabled boxes allows the user to move the Rectangle within the bounds of the enclosing Frame object.
- **Resizable** — Normally cleared. Setting this box plus the Selectable and Enabled boxes allows the user to resize the Rectangle within the bounds of the enclosing Frame object.
- **Selectable** — Normally cleared. Setting this box plus the Enabled box allows the user to select the Rectangle.

## 3.11 Simple SmartContainers

The Simple SmartContainer provides you with a way to create Smart modules that can run in batch mode without a user interface. You can see this usage illustrated in Figure 6–6, where loadable modules are created by packaging SmartDataObjects and SmartB2BObjects in Simple SmartContainers. Since neither the SmartDataObject nor the SmartB2BObject has a run-time representation, there is no need to use a visible container. The Simple SmartContainer provides

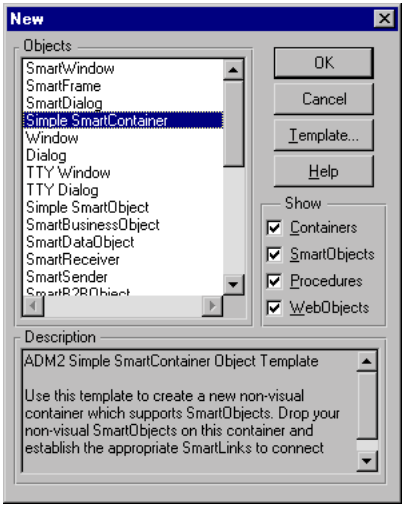
the critical services encapsulated in the SmartContainer super procedure without the overhead of a visible window. This can be very convenient in selected cases.

Of course, if you prefer to use a visible SmartContainer such as a SmartWindow, you are perfectly free to do so.

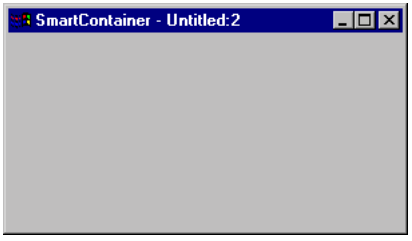
### 3.11.1 Creating and Populating a Simple SmartContainer

The Simple SmartContainer is supplied as a template. To create a complete object, follow these steps in AppBuilder:

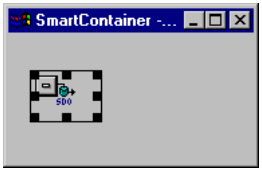
1 ♦ Choose File→ New. Choose Simple Smart Container from the dialog box that appears:




2 ♦ The container appears. Note that objects of this type do not have a Frame element, so you will not see a grid even if you have grid visibility turned on in your Preferences:

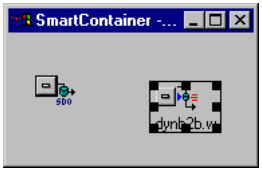


3 ♦ Reduce the size of the container workspace, if desired. Begin completing your new module by adding a non-visible SmartObject to the container. Shown here is a SmartDataObject:




**CAUTION:** If you place an object that has a visible representation at run time, the behavior of your Simple SmartContainer module will be undefined.

4 ♦ Add additional objects as desired to complete the module. Shown here is a SmartB2BObject:




As you add such additional objects, Advisors might appear and offer to create SmartLinks for you. Examine each offer and accept those that seem appropriate.

5 ♦ Configure each object you employ in the module.

6 ♦ Open a Section Editor window, if necessary, and add code to complete the module.

7 ♦ When you have completed the module, save your work using a distinctive filename. Note that Simple SmartContainer filenames conventionally begin with `c`.

You now have a module you can use in your application.

## 3.11.2 Placing and Configuring a SmartContainer Module

The Simple SmartContainer is an organizer object without a run-time representation. You can run it as an independent unit, or use it as a convenient wrapper for a module that you will make the child of some other Smart organizer object.

### As an Independent Module

To run a Simple SmartContainer as an independent object requires no special preparation. Build the module as described in "Creating and Populating a Simple SmartContainer", save it, and run it using the startup parameters appropriate to your circumstances.

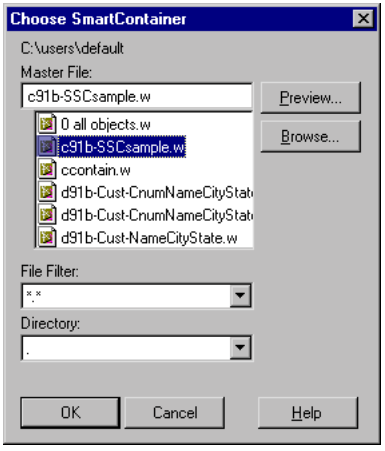
### As a Child Module

To place and configure a Simple SmartContainer module that you created, follow these steps:

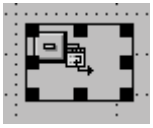
1 ♦ Click the SmartContainer tool icon in AppBuilder's Object Palette:



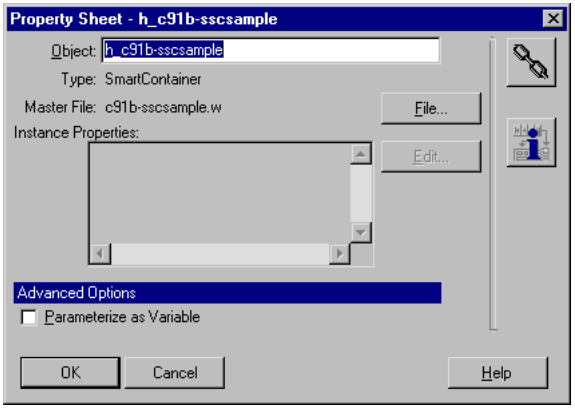
2 ♦ When the Choose dialog box opens, select a SmartContainer module and click OK:




3 ♦ Move your cursor to an empty place in your workspace and click to release the object. You will see a representation similar to this:



4 ♦ Click the instance's menu button and choose Properties. The SmartObject Properties dialog box opens:




5 ♦ Change the Object identifier to one that more clearly reflects the role of this container in your application.

6 ♦ If you wish to use this instance as a placeholder, check the Parameterize as Variable box. For information about placeholders, see the "Creating a SmartObject Placeholder" section in Appendix A, "Frequently Used Dialogs," and the *Progress ADM 2 Guide*.

7 ♦ Save your work.

Note that the Simple SmartContainer does not have a separate Instance Properties dialog box

## 3.12 Creating Paged Layouts

Every Smart organizer (SmartWindow, SmartFrame, or SmartDialog) allows you to partition sets of SmartObjects into logical pages. You can choose to represent these logical pages by separate windows, or by re-using (multiplexing) one or more areas of a single window. The number of different visual effects you can achieve by various combinations of arrangement and object choice is nearly unbounded.

In addition, AppBuilder offers a dedicated paging object: the SmartFolder. The SmartFolder implements the popular tabbed-file-folder metaphor, an industry-standard way to multiplex screen space and control the display of your logical pages.

By designing wisely with ADM paging, you can reduce the experience of visual and conceptual clutter, make better use of limited display space, and improve the usability of your applications.

### 3.12.1 Page Identifiers

Each ADM page is identified by a unique, non-negative integer. Page 0 (zero) exists by default; you must create and assign numbers to all other pages that you need.

You need not assign page numbers sequentially, and it is fine to leave gaps. You can define up to 32767 pages for a given context—a much larger number than you would probably want to try to manage. To create a new page, you need merely make it the current Design Page (see the "Special Pages" section).

A page remains in existence as long as it is the current Design Page, or has objects assigned to it. AppBuilder automatically destroys empty pages that are not current.

### 3.12.2 Special Pages

Three page designations have special meaning within the paging system:

- Design Page
- Page 0 (zero)
- Startup Page

These page designations are discussed individually in the following subsections.

#### Design Page

When you place SmartObjects into a Design Window workspace, you are placing them on the current Design Page, whatever that might be at the time. The current Design Page setting appears in the status bar of the AppBuilder main window, and is meaningful only for placing SmartObjects. The default Design Page is always Page 0. An ADM page may have any unique, non-negative integer as its identifier.

AppBuilder places basic 4GL and ActiveX objects on Page 0, **regardless of the Design Page setting**. Only SmartObjects may reside on pages other than Page 0. You may be able to work around this limitation by placing such non-Smart objects in a SmartFrame, and then placing the SmartFrame on the page of your choice.

As the name suggests, the Design Page setting is relevant only in design mode. It is not meaningful at run time, and is ignored.

To make a different page the Design Page, follow these steps:

1 ♦ Open the dialog box using one of these methods:

- Double-click on the Page Number field in the AppBuilder main window.
- Choose Edit→ Goto Page.
- Open the Procedure Settings dialog box (Tools→ Procedure Settings), choose the Pages button, and finally choose the Design button in the Pages dialog box.

2 ♦ Enter the number of the new Design Page. If that page does not exist, AppBuilder creates it immediately.

**NOTE:** Remember that although you can change the Design Page, the change only affects SmartObjects. When you place other objects, including ActiveX objects, you are placing them on Page 0 regardless of the Design Page setting.

To see the contents of all pages at once, change the Design Page value to a question mark (?). You can then align objects that reside on different pages so that page-changes appear neater and more orderly. All else being equal, users perceive neatness to indicate higher quality.

### Page 0 (Zero)

Page 0 is the default page in every SmartContainer. If you create a SmartContainer and add objects to it without changing any of the paging settings, then all of the objects will be on Page 0.

Page 0 objects are always displayed. If you have some objects assigned to Page 0, some assigned to Page 1, and some assigned to Page 2, the possibilities for display will be: Page 0 objects only; Page 0 objects + Page 1 objects; Page 0 objects + Page 2 objects. Unless you write special code to handle the situation, you cannot exclude Page 0 objects when displaying objects on other pages.

#### Startup Page

You can choose to designate one page, in addition to Page 0, to be displayed when the SmartContainer starts up. This is called the Startup Page.

Page 0 is the default Startup Page. Unless you explicitly specify a nonzero page number as the Startup Page, only the objects placed on Page 0 will display when you run the SmartContainer.

For example, if you use a SmartFolder to organize the viewing and hiding of pages, you typically designate Page 1 (which always corresponds to the leftmost folder tab) as the Startup Page. If you leave the Startup page set to Page 0 when you use a SmartFolder, then the SmartObjects in the folder will not appear when the application starts up.

You set the Startup Page in the Pages dialog box, which you invoke from the Procedure Settings dialog box.

### 3.12.3 Why Only SmartObjects?

The Progress ADM defines the paging system in terms of a special set of SmartLinks. When you assign a SmartObject to some Page *n* in a Smart organizer object, AppBuilder automatically creates a SmartLink of type Page*N* between the organizer and the new SmartObject. Since only SmartObjects support and can use SmartLinks, no other type of object can participate in ADM paging.

## 3.13 Paging with SmartFolders

The ADM SmartFolder object offers a convenient, popular way to develop a paged interface design. The SmartFolder implements the tabbed-file-folder metaphor with which most people who work in an office are familiar. The images in Figure 3–9 show the SmartFolder in use, managing the display of two pages in a toy contact-management application.

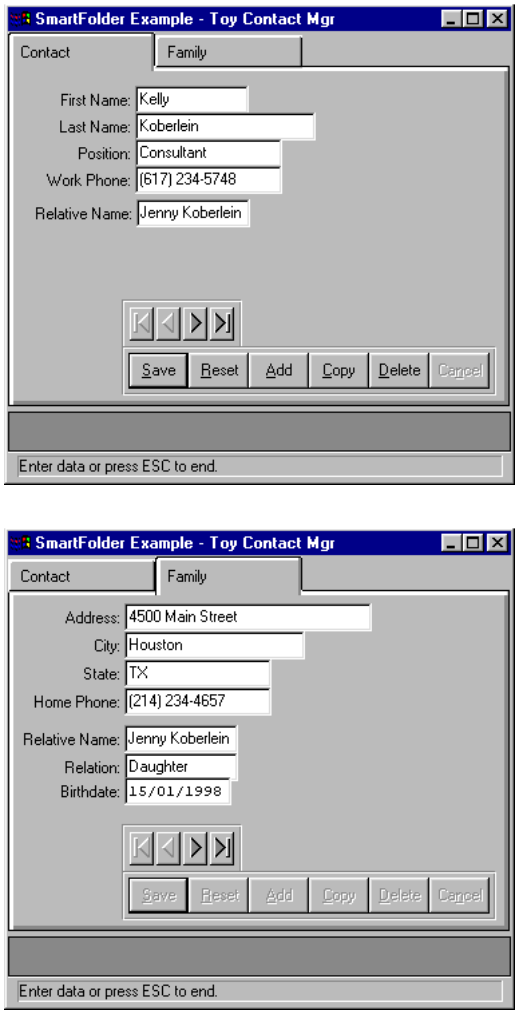



**Figure 3–9: Paging Using a SmartFolder**

While convenient to use and appropriate for many applications, SmartFolder may not be suitable for all cases. Although you can define up to 32 tabs, they will all be shown in a single line. This spares your users the disorientation of having tab rows re-order themselves, but might require that they use a horizontal scrollbar.

If you conclude that a SmartFolder is an appropriate way for you to manage paging in your application, follow these steps:

1 ♦ Sketch out the purpose and organization of each tabbed page. Keep in mind these points:

- There is no tab for Page 0. The leftmost tab always selects Page 1.
- Non-Smart objects will always be visible by default, regardless of the tab selected. They always reside on Page 0, and objects on Page 0 are always visible unless you add special code to manage the situation.
- If your design requires that you place non-Smart objects within the perimeter of the SmartFolder, you might need to treat them in a special way.

  For example, a Rectangle cannot be placed directly on the surface of a SmartFolder—it sinks below the surface and cannot be seen. Nor can it be brought to the surface using Layout→ Move-to-Top. To use a Rectangle for visual organization, you must first place a borderless (thus invisible) Frame object, with its Scroll Bars option turned off, on the SmartFolder. Then place the desired Rectangle on the Frame. The Frame will keep the Rectangle from sinking through the SmartFolder. (You should probably avoid placing non-Smart objects within the SmartFolder if you can.)

2 ♦ Be sure that your workspace is a SmartWindow, SmartDialog, or SmartFrame. No other organizer object will support ADM paging.

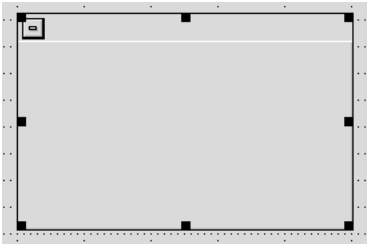
3 ♦ Set your Design Page to 0.

**CAUTION:** Since nested paging is not possible—ADM supports only a single set of pages for any single SmartContainer—**the SmartFolder must always reside on Page 0**. If you place the SmartFolder on, for example, Page 1, it will abruptly disappear from view the moment you select the tab for any other page.

4 ♦ Click the SmartFolder tool icon in the Objects Palette:



5 ♦ Move your cursor over an empty spot on your workspace, and click again to place the new SmartFolder:

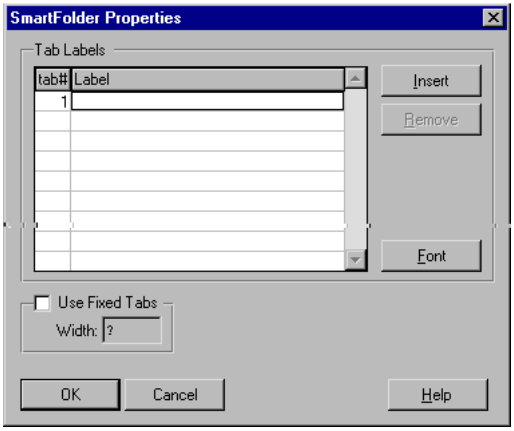


6 ♦ A Progress Advisor dialog box will appear, offering to create a Page SmartLink between the Smart workspace and the new SmartFolder object. Examine the offer, and accept it if it seems appropriate.

The Page link allows the SmartFolder to tell the Smart organizer which page to display when the user selects a tab. For more detailed information about SmartLinks and their meanings, see the *Progress ADM 2 Guide*.

7 ♦ If necessary, click on the SmartFolder to reveal its handles (you can see them illustrated in Step 5). Use the handles to adjust the size and position of the SmartFolder appropriately.

8 ♦ Click on the SmartFolder's menu button and choose Instance Properties. This dialog box will open:




- Enter the labels for the tabs you will use. To move to a new field, choose the Insert button. If you wish to insert a field before the current field, use the **UP-ARROW** key. To remove an entry, choose the Remove button.

- If you wish to use a special font for the tab labels, choose the Font button and select a font in the dialog box that opens in response (see the "Text Style Selector and Editor" section in Appendix A, "Frequently Used Dialogs,"for more information.)
- By default, the width of the each tab agrees with the label; tabs are not of fixed size. If you prefer fixed-size tabs (shown in Figure 3–9), check the appropriate box and enter a size for the tab. You may need to experiment to achieve a pleasing effect.

When you have finished configuring the tabs and their labels, choose OK to dismiss the dialog.

9 ♦ Set the Design Page to 1 (remember: the leftmost tab selects Page 1, not Page 0). Place and arrange the SmartObjects belonging to Page 1. To control how the objects layer on top of one another, use Layout→ Move-to-Top and Layout→ Move-to-Bottom. Check to be sure this is effective: not all objects respond to relayering requests.

**NOTE:** Remember that only SmartObjects can participate in ADM paging. If your layout includes non-Smart objects, they will remain visible regardless of the tab selected.

10 ♦ Repeat Step 9 for each succeeding page you defined, being sure to set the Design Page correctly each time before beginning to place objects. If you make a mistake, open the Pages dialog and move the misplaced objects to the correct page. See the "ADM Pages Manager" section in Appendix A, "Frequently Used Dialogs," for more information.

11 ♦ Set your Startup Page to the appropriate nonzero number, generally 1. Save your work.

## 3.14 Paging Without SmartFolders

Because paging is a property of the ADM itself rather than of SmartFolder objects, it is quite easy to develop an ADM-based paging scheme of your own, if you choose. Figure 3–10 and Figure 3–11 represent two possible designs for which AppBuilder supplies predefined controls.

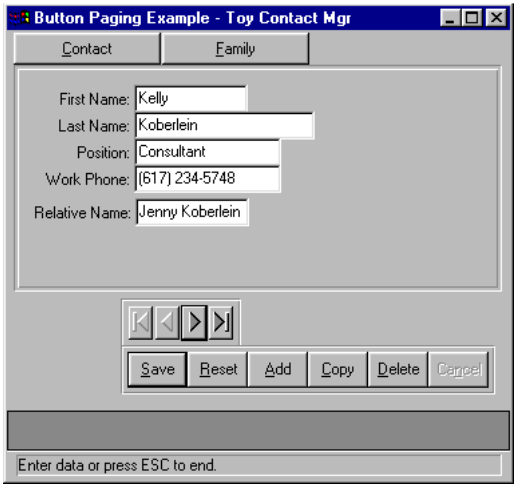



**Figure 3–10: Paging Using Pushbutton Controls**

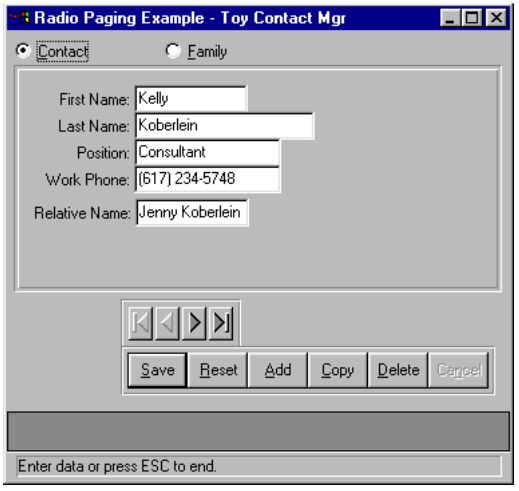



**Figure 3–11: Paging Using Radio-set Controls**

The two layouts illustrated here resemble a SmartFolder in appearance, but that was a deliberate design choice, not a requirement of the technology. You could create designs that look very different to these. You could use images rather than text on Pushbutton controls. You could arrange the control buttons along the side or bottom rather than the top, color-code them, use very long strings as labels, and so forth. The possibilities are numerous.

The key to paging is the selectPage( ) procedure:

```
RUN selectPage ( 63 ) .
```

Inserted as the trigger code for a Pushbutton CHOOSE event, the example call would reveal all SmartObjects assigned to page 63 and hide all others. The value of the argument can be any non-negative INTEGER.

An alternative to the selectPage( ) call is the viewPage( ) call. Calling viewPage( 63 ) would reveal all SmartObjects assigned to page 63, but it will **not** hide any others. Anything in view before the call will still be in view after. This creates quite a different effect.

AppBuilder supplies a Pushbutton that contains the following code for the Choose event:

```
DO
   RUN viewPage ( 1 ) .
END.
```

Right-click the Pushbutton object icon in the Palette and choose the View Page item from the list. You can convert this button very easily by substituting a call to selectPage( ) for the existing call to viewPage( ). Make this substitution using AppBuilder's built-in Section Editor.

Pushbuttons are not your only choice for paging controls. AppBuilder also supplies a set of Radio buttons predefined with selectPage( ) calls. Right-click the Radio-set object icon in the Palette and choose the Page Select item from the list.

**NOTE:** You should keep in mind that Progress Software Corporation supplies this Radio set as a convenience, but does not necessarily endorse its use. Most experienced GUI users expect Radio-button sets to represent data, so using them as action controls might cause confusion and reduce your application's usability.

## 3.15 Paging Whole Windows

You will find it easiest to construct an ADM-based paging system if you design with a single SmartWindow in mind as the paging context. The examples above follow such a single-window model.

If you are careful to design appropriately, you can construct a multi-window application that uses ADM paging to control window display and hiding. Figure 3–12 shows a sample application in which the larger window is opened or closed by the pushbuttons in the smaller window.

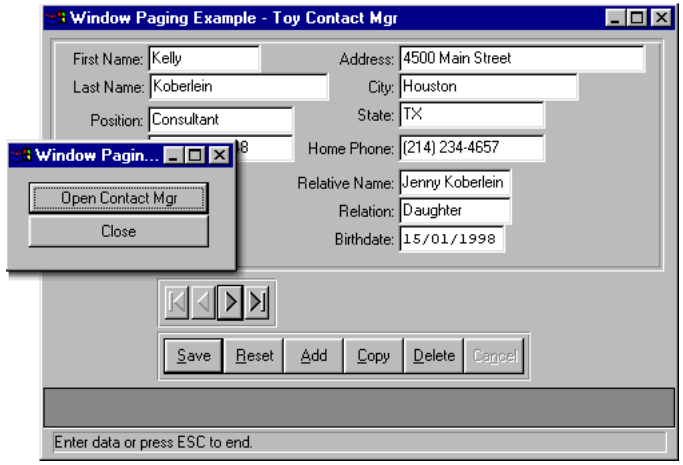



**Figure 3–12: Paging a Whole Window**

To reproduce the example shown in Figure 3–12, follow these steps:

1 ♦ Create a new SmartWindow. This will be the Work Window. Populate it with a SmartDataObject, SmartDataViewer, and the two SmartPanels (navigate and update). Save this assembly under a distinctive name.

2 ♦ Create a second new SmartWindow. This will be the Control Window. Populate it with two Pushbuttons, using the View Page versions of the buttons for convenience. Change their labels appropriately.

3 ♦ Select the button that will open the Work Window. Open the Section Editor and replace the call to viewPage(1) in the trigger code with a call to selectPage(1). Close the Section Editor window.

4 ♦ Set the Design Page to 1. Select the SmartContainer object tool from the Palette. When the selection dialog box opens, choose the file that represents the Work Window you created and saved in Step 1.

5 ♦ Place this object in the Control Window.

A SmartWindow placed in this way will not display its run-time representation, but instead will present as a non-visible SmartObject. This is as it should be—the Work Window is an independent SmartWindow and thus has no code that would allow it to open within the Control Window as a component object. Within the Control Window, the Work Window is indeed a non-visible object.

6 ♦ Select the button that will close the Work Window. Open the Section Editor and replace the call to viewPage(1) in the trigger code with a call to selectPage(32768). Note that the number need not be 32768—simply choose some distinctive, non-negative integer that you are not using for an actual page number.

A call to selectPage( ) always turns off the old page before turning on the new one. However, since you have not defined a page 32768, there is no new page to turn on. So pressing this button will only have the effect of closing (turning off) the Work Window.

7 ♦ Save the Control Window object. Run it, and choose the buttons to open and close the work window.

# 4

# Data-access Objects

Locating, reading, and writing database records is the one fundamental activity in any database application. This chapter describes the data-access objects AppBuilder supplies for your use, and discusses how to create, modify, and use them.

Topics discussed in this chapter include

- Static and Dynamic SmartDataObject
- SmartFilter Object
- SmartBusinessObject
- Free-standing Query Object
- Defining and Editing Queries using Query Builder
- Selecting Database Fields for Browsing

## 4.1 Introduction to Data-access Objects

Almost all database operations are performed against only a subset of the records in a database, and often against only a subset of the fields within those records. The reasons for this typically include security and privacy issues, the limits of human cognition and, in modern multi-terabyte databases, the computational and real-time costs of repeatedly traversing enormous numbers of irrelevant records.

Every Progress application you build, regardless of what it does with the records afterwards, will start by connecting to at least one database and fetching records that meet certain criteria.

AppBuilder offers several powerful ADM SmartObject modular building blocks for data access:

- Static and Dynamic SmartDataObjects
- SmartBusinessObject
- SmartFilter

These are in addition to the standard Progress objects including standalone Queries and objects that combine data access with visualization and update capability.

The rest of this chapter discusses how to use these building blocks in your applications.

## 4.2 SmartDataObjects

SmartDataObjects are modular data pumps that incorporates ADM Smart technology. They communicate in a loosely coupled way with other ADM SmartObjects using SmartLinks, and with objects on the Web using WebSpeed. They can also function in an Open4GL environment with non-Progress objects. SmartDataObjects are supplied for your use in both dynamic and static forms.

### 4.2.1 Static SmartDataObjects

Static SmartDataObjects act as custom data pumps, each producing and managing a specific data stream based on the terms of a specific query. You define a base version of that query within a master version of the object at design time.

Depending on your business needs, you can also define and add custom validation routines to the master object. These validation routines can operate at various levels of granularity, from a single field to a whole transaction. Once you have the master defined, you can then use instances made from it as often as you like.

When you compile such a static SmartDataObject, the Compiler creates two different versions of the executable r-code. One represents the complete object, usable in any context. The other is a stripped-down, client-only version. Figure 4–1 illustrates this process.

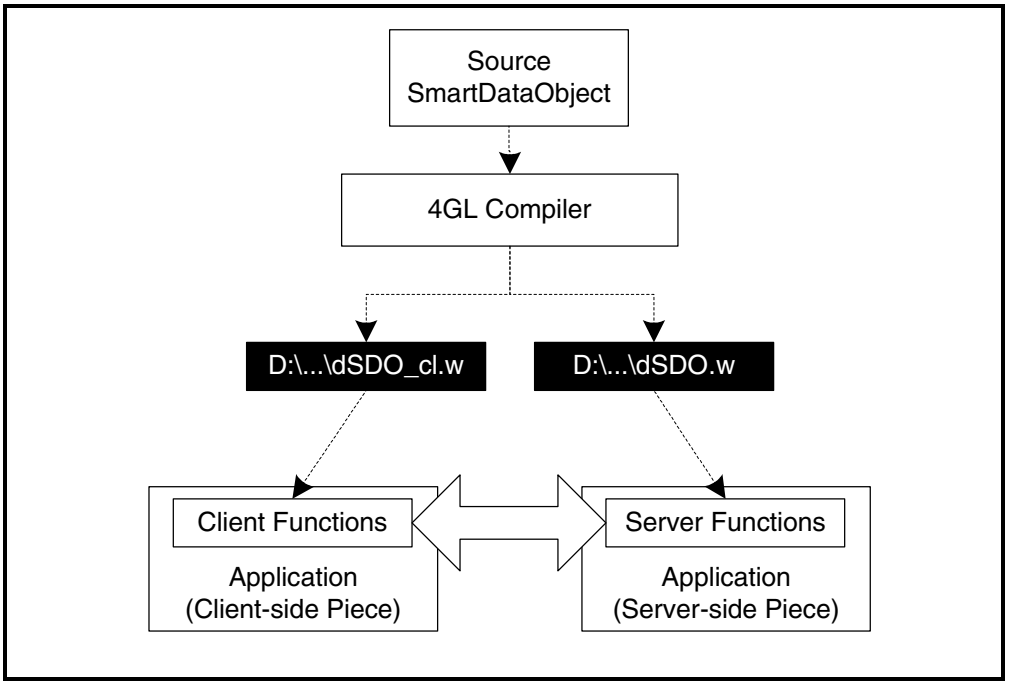



**Figure 4–1: Compilation of a SmartDataObject**

The stripped-down version (identified by `_cl` in the filename, for example: `dTest_cl.w`) will load whenever there is no local database connection. A `_cl` object expects to cooperate with the complete version of the same object loaded as part of an AppServer partition.

You can associate business rules or any other required logic with the object by writing the necessary code.

### 4.2.2 Dynamic SmartDataObjects

The dynamic version of the SmartDataObject is designed particularly for distributed use.

A static client-side SmartDataObject can only deal with one particular data stream—the specific data stream being supplied by its counterpart on the server. This is a limitation that can be significant where many such streams are involved, if the client-side objects are not stored locally. The Progress WebClient is one such case—it is designed to be downloaded for use over a wide-area network such as the Internet. (See the book *Progress on the Web*, for detailed information about downloading and using the new WebClient.)

The dynamic SmartDataObject solves a large part of this problem.

The dynamic object acts as a generic, general-purpose data pump. It is not limited to handling a specific data stream, as a static SmartDataObject is. The dynamic object automatically cooperates with any static, server-side SmartDataObject and can be attached to any client-side display object such as a SmartDataBrowser or SmartDataViewer.

These broad capabilities come at the cost of client-side customization**.**

Because it is designed to handle a data stream of any composition, you cannot enhance the dynamic SmartDataObject with custom routines. This is true even of custom validation procedures. You can add custom client-side validation routines to the static object, if you choose, but you should remain aware that **all** such custom routines will actually run in the static object on the AppServer, not on the client.

Since any SmartDataObject is only a data pump, you must connect its data stream to other modules for display or manipulation. Those other objects can take a number of forms, both Progress and non-Progress. You define and create them separately from the SmartDataObject.

### 4.2.3 Creating a SmartDataObject Master

The individual static SmartDataObjects you employ as building blocks in your application are instances of one or more master objects. Using the AppBuilder's SmartDataObject design Wizard, you can create any number of masters that produce different data streams based on the queries you define in them.

#### Design Considerations

Before you start the SmartDataObject design Wizard, you should consider these questions:

- Will you need to define a temp-table first to reduce the amount of processing? Large quantities of data can affect performance. If your application must repeatedly traverse a very large database, you might get a significant performance increase by first subsetting to a temp-table, and then operating on that subset instead.

  Note that although the Wizard will allow you to define such a temp-table, and will allow you to assign it as the SmartDataObject's data source, you must write the code by hand that actually locks and copies records from the permanent table. This population step must take place before you give the SmartDataObject access to the temp-table at run time, if the rest of your application expects to find records in the table.

- What databases and tables do you want this SmartDataObject to use? This base query can be refined or sorted at run time.

- What fields do you want to expose to client objects, and what properties (such as read-only) do you want to set for each data element?
- What are your plans for validating data on the client side (the user interface side) and the SmartDataObject side? See the "Data Validation Limitation" section in this chapter for a discussion of an unusual and potentially important issue.

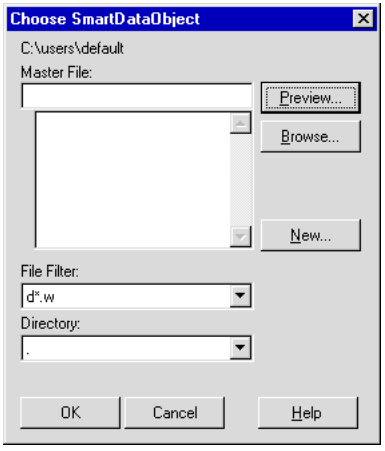
### Creating the Master

AppBuilder's SmartDataObject Wizard has four pages. Only pages 2 and 3 involve significant work:

1. Introduction
2. Creating the temp-tables, if any, and the query
3. Selecting the fields the SmartDataObject will offer to visualization objects
4. Congratulations

To create a SmartDataObject master, follow these steps:

1 ♦ Open the Choose dialog box by clicking on the SmartDataObject tool icon in the Object Palette. When the dialog box opens, start the SmartDataObject Wizard by choosing New:

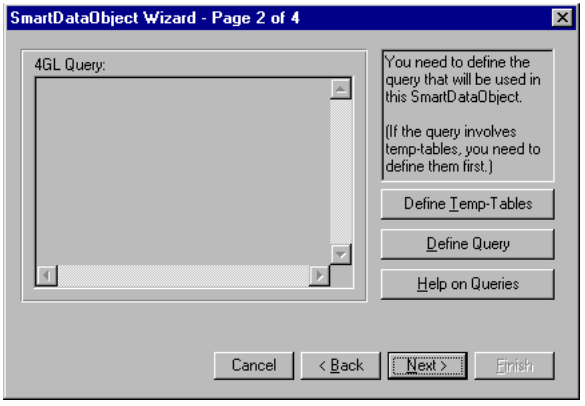



**NOTE:** You could also start the Wizard by choosing File→ New and selecting the SmartDataObject item from the list, or by right-clicking the icon in the Palette and selecting New SmartDataObject from the pop-up menu.

2 ♦ Page 1 of the SmartDataObject Wizard appears:




3 ♦ When you have read the introductory text, choose Next to move to page 2:

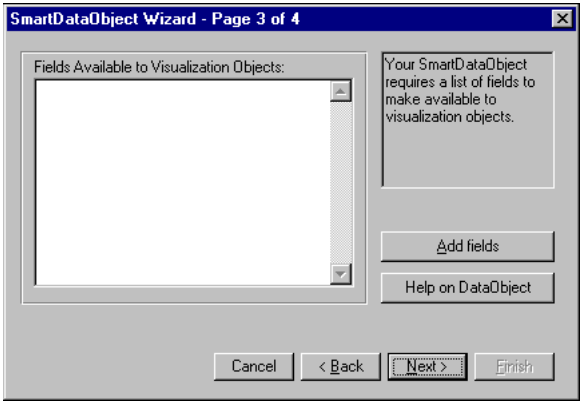



4 ♦ If this SmartDataObject will operate on one or more temp-tables, choose the Define temp-tables button to open the Temp-Tables Maintenance editor. Using the editor, define the temp-tables this SmartDataObject will use. See the "Temp-Tables Maintenance" section in Appendix A, "Frequently Used Dialogs," for information about how to use the editor.

5 ♦ Choose the Define Query button to start Query Builder. Define the base query this SmartDataObject will use. See the section "Defining and Editing Queries Using Query Builder" in this chapter, for information about how to use Query Builder.

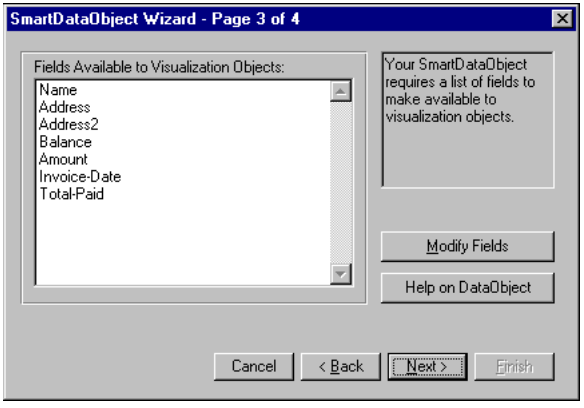
**NOTE:** The option to create a free-form query is not available to you at this stage. If you need or want to create a free-form query, you must create a placeholder using Query Builder and then edit the query after the Wizard completes. See the "Editing a SmartDataObject Master"section in this chapter, for further information.

6 ♦ When you finish with Query Builder, you will see displayed on page 2 the text of the query you just created. When you are satisfied with your choices for temp-tables and query, choose Next to go to page 3 of the Wizard:




7 ♦ Note that at this point, the list of fields is empty. Choose Add Fields to start the Column Editor, and use the editor to select the fields that you want to make available through this SmartDataObject. For information about how to use the Column Editor, see the "Selecting Database Fields for Browsing" section in this chapter.

8 ♦ When you return from the Column Editor, the list will be populated with the fields you have chosen:



9 ♦ Choose Next on page 3 of the Wizard to advance to the Wizard's final page. If you successfully defined a database query and the related SmartDataObject RowObject temp-table fields, the Congratulatory screen displays. (If not, press the Back button and make any necessary changes.)

10 ♦ Choose Finish to dismiss the Wizard and reveal the new SmartDataObject master, if it was not already visible:




11 ♦ Choose File→ Save and give the new master a descriptive filename. Note that, by convention, SmartDataObject filenames begin with d.

The new SmartDataObject master is now available for your use.

Note, too, that AppBuilder actually creates two different forms of the master. If you were to give it the name dExampleSDO and then check the filesystem, you would see

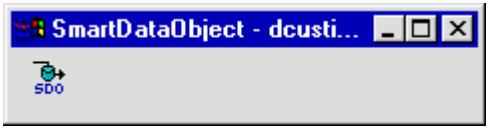
- A standalone version, named `dExampleSDO.w` This version is loaded whenever there is a local database connection. AppBuilder creates it with the DB–REQUIRED preprocessor value set to YES, since it should not load if there is no database connection.
- A client-side version, named `dExampleSDO_cl.w`. AppBuilder creates it with DB–REQUIRED set to NO. This version is loaded when there is no local database connection, on the presumption that the application is running in client/server mode and the database is resident on the server side. This version is compiled with all DB–REQUIRED internal procedures stripped out, which can create a problem. See "Data Validation Limitation" in this chapter for further information.

### 4.2.4 Editing a SmartDataObject Master

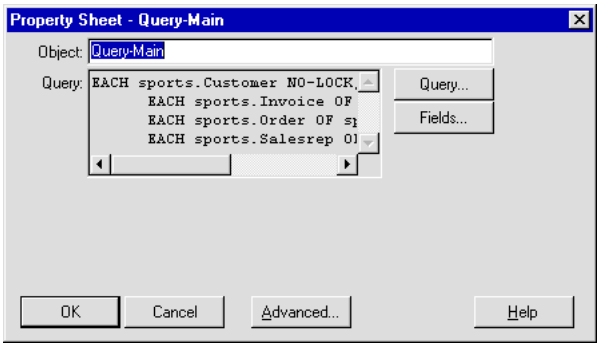
Any modular system must have and enforce strict data-exchange requirements if it is to avoid possible data corruption. Although you can always update your SmartDataObject masters to reflect your changing needs, you must also update any other master objects that share data with them. Furthermore, you must sometimes update all the masters in a specific order if you are to avoid breaking the signatures on which data communication depends. For more information about making data-stream changes successfully, see the "Making Changes to the Data Stream" section in this chapter.

When you are ready to edit a SmartDataObject master, follow these steps:

1 ♦ Choose File→ Open and select the main source file (for example `dExampleSDO.w`, not the `dExampleSDO_cl.w` client-only file). The SmartDataObject appears:




2 ♦ Double-click on the object. The SmartDataObject property sheet dialog box appears:




3 ♦ Choose the Query button to edit the existing query, if necessary. See the "Defining and Editing Queries Using Query Builder" section in this chapter for information about how to use Query Builder.

4 ♦ Click on the Fields button to modify the original fields and their attributes. See the "Selecting Database Fields for Browsing" section in this chapter for information about how to use the Column Editor.

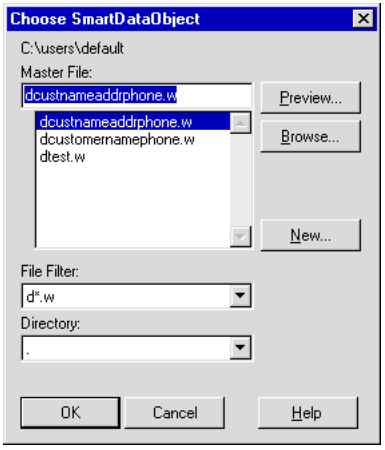
5 ♦ Save your changes.

### 4.2.5 Creating a SmartDataObject Instance

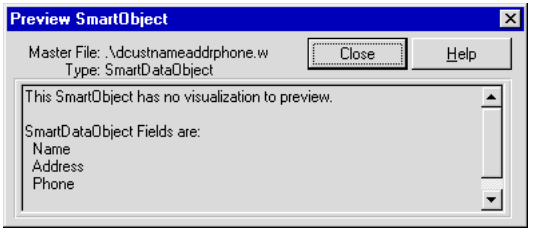
Once you have a SmartDataObject master designed that will supply the data you need, you can re-use it indefinitely. Because of the way AppBuilder checks for opportunities to create SmartLinks, you will probably find it good practice to select and place your SmartDataObject instance before you place the instances of the SmartObjects that will communicate with it.

Follow these steps to use a SmartDataObject in your application:

1 ♦ Open the Choose dialog box by clicking on the SmartDataObject tool icon in the Object Palette. When the dialog box opens, select the appropriate SmartDataObject master from the list:



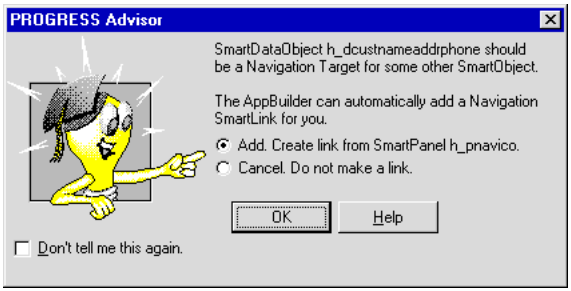

If you have difficulty remembering which SmartDataObject master is which, choose Preview to open a dialog box that displays the fields listing for the selected master:




2 ♦ Position the new instance (created automatically when you chose the master) anywhere you like in your Smart organizer workspace, and click to release. This is the design-time representation you will see:



3 ♦ AppBuilder will now automatically look for opportunities to create SmartLinks between it and other SmartObjects in the same organizer context. If it thinks it has found such an opportunity, an Advisor window opens. The example shown here presumes that you had previously placed a Navigation SmartPanel:




If the suggested link seems appropriate for the needs of your application, choose OK. If you decide later to make changes, you can easily revise the linkage using the SmartLinks editor. For more information the SmartLinks Editor, see the "SmartLinks Editor" section in Appendix A, "Frequently Used Dialogs."

4 ♦ Set the properties of this SmartDataObject instance. See the "Configuring a SmartDataObject Instance" section in this chapter for further information.

5 ♦ Save your work.

### 4.2.6 Configuring a SmartDataObject Instance

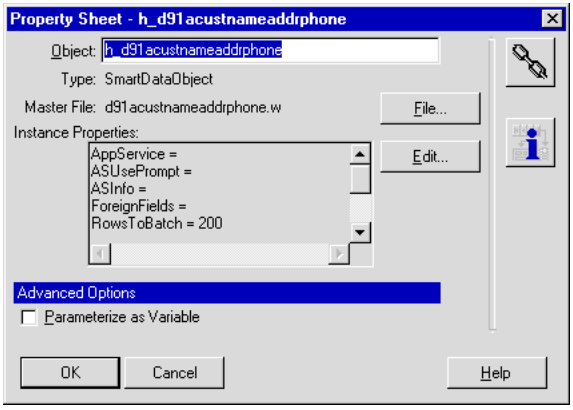
Every SmartDataObject instance has a number of default properties you might wish to change. Two of the most important are:

- **Determinism** — By default, every instance you place represents a particular master object, with a defined behavior. Under some circumstances, it might be more appropriate to treat the instance as a placeholder at design time, and conditionally load a particular SmartDataObject under program control at run time. The possible applications of this ability are numerous.

- **Statelessness** — By default, SmartDataObjects do not bind their AppServer session unless the AppServer Broker is configured to run statefully, with session-long connections. If the AppServer Broker is running in stateless (connectionless) mode, so will the SmartDataObject. Statelessness allows many more clients to share the available AppServer sessions. The cost is an increase—typically small—in the time it takes to re-establish connection for each request. To avoid this time penalty in selected cases, you can choose to force the SmartDataObject to bind its session and operate statefully.

To configure the SmartDataObject instance, follow these steps:

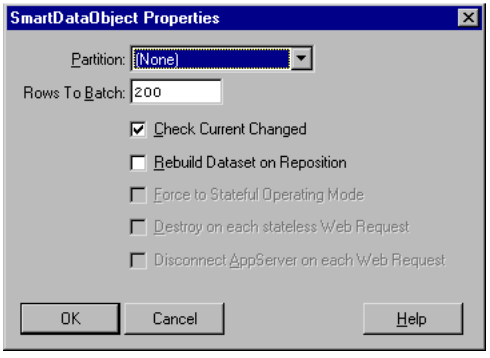
1 ♦ Click the menu button to open the SmartDataObject menu. Choose the Properties item. The properties dialog box opens:




2 ♦ Change the Object identifier to one that more accurately reflects the role of this SmartDataObject in your application.

3 ♦ If you wish to use the object as a placeholder, see the "Creating a SmartObject Placeholder" section in Appendix A, "Frequently Used Dialogs."

4 ♦ Choose OK to dismiss the dialog box. Click the menu button again, and choose the Instance Properties item. That dialog box opens:




5 ♦ Set the properties and choose OK.

The Instance Properties you can change in Step 5 are:

- **Partition** — Normally cleared (set to None). If this SmartDataObject is to run on an AppServer, you must assign it to a partition from this list. If there are no partitions listed, you can define as many as you need using the AppServer Partition Deployment Tool, available from the PRO*Tools toolbar. Partition tags that you define in this way are immediately made available to AppBuilder, and appear in this list.

- **Rows to Batch** — Normally 200. This number represents the number of records fetched from the data source into the RowObject temp-table per read operation. Any choice you make here has performance implications that cannot be easily characterized. Experiment to see what number gives the best performance for your particular application.

- **Force to Stateful Operating Mode** — Cleared and cannot be set unless you assign this object to a partition. Setting this box will cause this object to bind its AppServer connection for the duration of the session. When the box is cleared, connection binding is determined by the AppServer itself.

The final two options are specific to use of the SmartDataObject in a WebSpeed context. They have no meaning in a standard client-server context. For further information, see the online help and the WebSpeed documentation.

- **Destroy on each stateless Web Request** — Normally cleared.

- **Disconnect AppServer on each Web Request** — Normally cleared.

### 4.2.7 Data Validation Limitation

You can choose to add data-validation procedures to your SmartDataObject. These optional procedures can operate at the column, row, or transaction level. If you do choose to create such routines, you must create them as internal procedures. Because the SmartDataObject calls these procedures automatically, you must follow the expected naming and error-handling conventions. See the *Progress ADM 2 Guide* and the *Progress ADM 2 Reference* for details.

SmartDataObjects have the DB–AWARE flag set. For DB–AWARE objects, AppBuilder sets the DB–REQUIRED flag in all new internal procedures, including validation procedures. Generally this is appropriate and useful. In one specific set of circumstances, however, it is not, and you will have to explicitly turn off the DB–REQUIRED flag. Those conditions are:

- The application will run on an AppServer. Applications that do not run on an AppServer are not affected. Note that good practice suggests that you always design for the greatest flexibility of use.

- The client program does not require a local database connection via a SmartDataObject. Client programs that must have such a local connection in order to run are not affected. Note that good practice suggests that you always account for the case of no local connection.

- Your routines validate rows or columns (*fieldname*Validate(), RowObjectValidate()). Routines that validate whole transactions (TransactionValidate()) are not affected.

The problem comes about because all DB–REQUIRED routines are removed from the client-only version of the SmartDataObject during compilation. This allows the program to load and run in the absence of a database connection. If your validation routines are marked DB–REQUIRED, then they, too, will be removed.

Normally, calling a non-existent routine would cause a run-time error. But calls to optional routines never complain if they fail, and validation routines are always optional. The net result is that, when the client-only object is loaded, database updates are carried out without client-side validation checks even though you wrote the routines to perform them. Neither the compiler nor the interpreter can detect this logical error.

### Solving the Problem

- Divide your validation tasks according to whether they require a database lookup operation. References to RowObject or other temp-tables do not count as lookup operations.

- Write your *fieldname*Validate() and RowObjectValidate() procedures to include only the validation operations that do not require lookup. Do not include direct references to a database in these procedures, and turn off the DB–REQUIRED flag in them. AppBuilder offers a convenient option in the Section Editor for this purpose.

- Put any validation operations requiring lookup into one of the TransactionValidate() procedures. Keep the DB–REQUIRED flag turned on in these procedures.

By writing your *fieldname*Validate( ) and RowObjectValidate( ) procedures such that you can legitimately turn off the DB–REQUIRED flag in them, you ensure that these routines will survive compilation and that all validation will occur as you intend.

### Forcing Server-side Validation

It is generally more efficient to do row and column validation on the client side, as part of SubmitRow(). There are some cases where this is not possible—such as calling a SmartDataObject from a Java program where no client-side version of the object exists. You would then want to force all validation, including row/column validation, to take place on the server. You can do this with a single line of code.

This code also works in the case where there is a client. In that case, the client-side validation operations take place first on the client side, and then a second time on the server side. This introduces some inefficiency, but is not otherwise harmful.

To force all validation to take place on the server, regardless of whether it also takes place in the client, follow these steps:

1 ♦ Open your SmartDataObject master for editing. It will appear on your display:




2 ♦ Choose Window→ Code Section Editor, and then select Procedures from the sections list.

3 ♦ Create a new procedure called initializeObject( ) —or edit the existing procedure by that name, if you already created it—and include in it this line:

```
dynamic-function ( 'setServerSubmitValidate', yes ) .
```

4 ♦ Save your work.

All instances made from the SmartDataObject master you just changed will now perform all row/column validation on the client side—if there is a client—and on the server side too.

### 4.2.8 Using the Dynamic SmartDataObject

In a situation involving an AppServer, part of the application runs on the client and part on the AppServer. Dividing the application in this way is done almost automatically—the only special thing you do is identify the AppServer partition at design time.

Changing your application to use the dynamic SmartDataObject is a one-step process: once your application is complete, move the `_cl` file (for example: `dSample_cl.w`) out of the PROPATH search space. If you are certain you will never want to use the static version on the client side, you can delete it. Otherwise, move it to a safe backup location.

That is all you need to do. If the `_cl` file cannot be found at run time, the dynamic object will be loaded and used instead.

**CAUTION:** The static SmartBusinessObject (see the "SmartBusinessObject" section for information about that object) does not support the dynamic SmartDataObject. Generally speaking, you will get the best results if you create separate SmartDataObjects for standalone and SmartBusinessObject use. If you must use the same SmartDataObject in both a SmartBusinessObject and in a standalone context, **do not move or delete the _cl file**. If it cannot be found during initialization of the SmartBusinessObject, initialization will fail and the SmartBusinessObject will not run.

## 4.3 SmartBusinessObject

The SmartBusinessObject allows you to neatly package and fully synchronize—including the ability to update in a single, server-side transaction—up to twenty SmartDataObjects. Figure 4–2 illustrates the place of a SmartBusinessObject in a Web-enabled application.

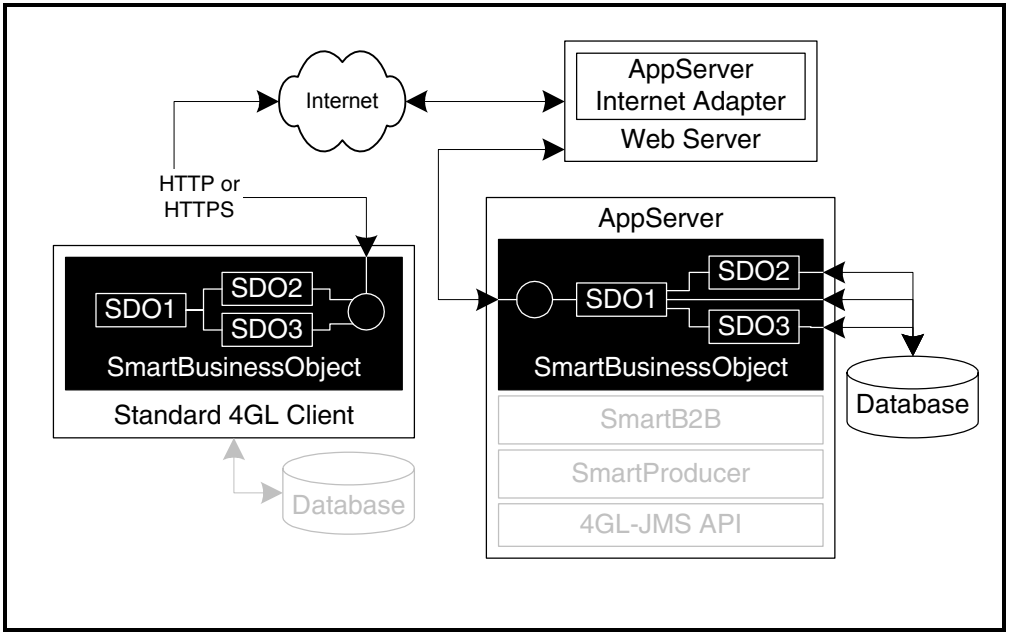



**Figure 4–2: SmartBusinessObject in a Web Environment**

Each static SmartDataObject produces and manages a single data stream. The nature and composition of that data stream is dependent on the query that you define within the object. Although you can cause a single SmartDataObject to produce a data stream of great complexity by the skillful use of JOINs, the result is often messy and difficult to use for business purposes.

One way to reduce complexity is to create a number of SmartDataObjects, each with a more simple query, and then synchronize them at the application level. For example, you might create five SmartDataObjects that respectively supply:

1. Customer records
2. Order records for a particular Customer
3. OrderLine records from a particular Order
4. Item records related to a particular OrderLine
5. Inventory records related to a particular OrderLine

Figure 4–3 illustrates these relationships.

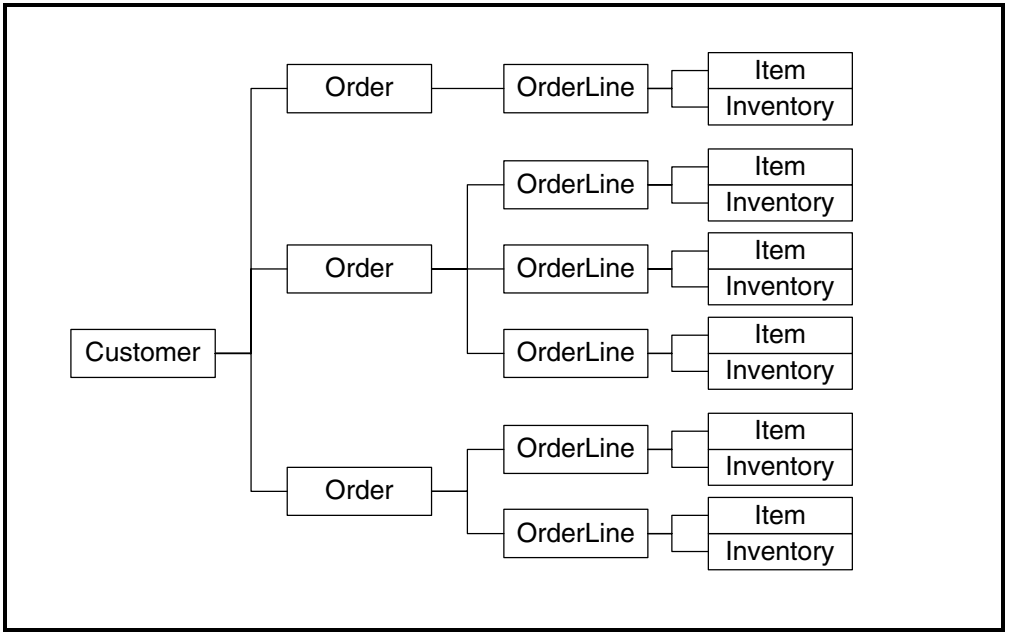



**Figure 4–3: Data Relationship Among Five Tables**

Synchronizing their read operations is not difficult. You can use DATA SmartLinks for that purpose regardless of whether you use a SmartBusinessObject.

The difficult task is synchronizing updates. In general, there is no simple way to update multiple, standalone SmartDataObjects in a single transaction, if the objects run in a distributed environment. Such objects operate independently of one another on an AppServer.

The new SmartBusinessObject solves this problem by providing a single context for all the SmartDataObjects you embed in it. Because the code you write in the SmartBusinessObject can operate on all the RowObject temp-tables belonging to its subordinate SmartDataObjects, the SmartBusinessObject allows you to perform updates in a single server-side transaction. With the SmartBusinessObject, you can create and manage integrated data streams that are as complex and powerful as your customers' business needs demand.

The SmartBusinessObject presents a single point of contact for external modules such as Smart data-display objects. Internally, it uses its CONTAINER links to connect those external objects with the SmartDataObjects it contains—communicating navigation instructions from a SmartPanel, for example, or sending data out to a SmartDataViewer for visualization. This all happens automatically; you need do nothing to make it work. The SmartDataObjects synchronize themselves using DATA links that you create.

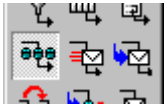
### 4.3.1 Creating and Placing a SmartBusinessObject

The SmartBusinessObject is a special-purpose organizer object and a member of the class SmartContainer. As is true of other organizer objects, the SmartBusinessObject is supplied as a nearly finished template. There is no Wizard involved; you need only create a copy, populate it, and add SmartLinks.

**CAUTION:** The SmartBusinessObject only supports **static** SmartDataObjects. If it cannot find the client-side (_cl) files at run time, it will not be able to use the dynamic SmartDataObject and will report an error when it fails to start up successfully.

To create a SmartBusinessObject master, follow these steps:

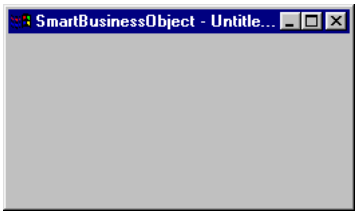
1 ♦ Click the SmartBusinessObject tool icon in the Object Palette:



When the Choose dialog box opens, click the New button.

**NOTE:** You can also choose File→ New or the New button on AppBuilder's main-window toolbar. When the New dialog box appears, select SmartBusinessObject from the Object list and click OK.

2 ♦ AppBuilder creates the appropriate workspace, which looks like a frameless SmartWindow. Because a SmartBusinessObject has no Frame element, you will see no layout grid even if you have grid visibility turned on:




3 ♦ Populate the object and configure it. See the "Populating and Configuring a SmartBusinessObject" section for more detailed information. Save the object using a distinctive filename. Note that SmartBusinessObject names conventionally begin with `sbo`.

4 ♦ Click the SmartContainer tool icon in the Palette. When the selection dialog box opens, select the SmartBusinessObject master you just defined.

5 ♦ Move your mouse cursor over your application workspace and click to place the new object instance.

6 ♦ If you have already placed other SmartObjects such as a SmartDataBrowser, an Advisor might now appear and offer to create SmartLinks between those objects and the new SmartBusinessObject. Examine the offers and accept those that meet your needs.

7 ♦ If the Advisor does not offer to create all the links you desire, choose Tools→ Procedure Settings and open the SmartLinks Editor to add links by hand. For information about the SmartLinks Editor, see the "SmartLinks Editor" section in Appendix A, "Frequently Used Dialogs."

8 ♦ Save your work.

### 4.3.2 Populating and Configuring a SmartBusinessObject

You have four major tasks when configuring your SmartBusinessObject master:

1. Determine what data this object will supply and decide how you will divide the total effort.

   In general, consider using one SmartDataObject per table. You can populate your SmartBusinessObject with a maximum of twenty SmartDataObjects.

2. Create and place the SmartDataObjects, add DATA links to represent the logical dependencies among them, and identify the foreign-field (relational) mappings.

   For detailed information about creating and placing SmartDataObjects, see the "Synchronizing SmartDataObjects Using DATA Links" section.

3. Determine whether the users of your application are more likely to prefer efficiency or completeness when browsing the data stream and set the property CascadeOnBrowse accordingly. This is a somewhat subtle issue. See the "How the CascadeOnBrowse Property Works" section for further information.

4. Determine your validation needs and create the appropriate routines to perform those tests.

   The SmartBusinessObject makes it easier to perform comprehensive validation tests. Besides the tests local to each SmartDataObject, you can write validation routines that run in the SmartBusinessObject itself. The SmartBusinessObject can see the Update tables belonging to each of its contained SmartDataObjects and you can take advantage of that fact to create whole-stream validation.

   Within the SmartDataObjects themselves, their update tables all have the same `RowObjUpd` identifier. Since they have separate namespaces internally, this is not a problem. But at the level of the SmartBusinessObject, each such update table needs a unique name. The ObjectName property serves that purpose.

   So if your SmartBusinessObject contains two SmartDataObjects to which you have given the ObjectNames `CustSDO` and `OrderSDO`, you can write code referring to fields in their respective `RowObjUpd` tables as `CustSDO.`*`SomeFieldName`* and `OrderSDO.`*`SomeFieldName`* and everything will work as you intend.

   For additional information about creating validation routines, see the *Progress ADM 2 Guide*.

### 4.3.3 Synchronizing SmartDataObjects Using DATA Links

To get the benefits the SmartBusinessObject can provide, you must identify the business hierarchy between the contained SmartDataObjects. You do this using SmartLinks of type DATA. The DATA-SOURCE is always the object that initiates a change and the DATA-TARGET is the object that must respond by staying in sync.

In the situation shown in Figure 4–3, you would create DATA links from the Customer object to the Order object, from Order to OrderLine, and from OrderLine to both Item and Inventory.

Figure 4–4 shows the hierarchy imposed by the DATA links.

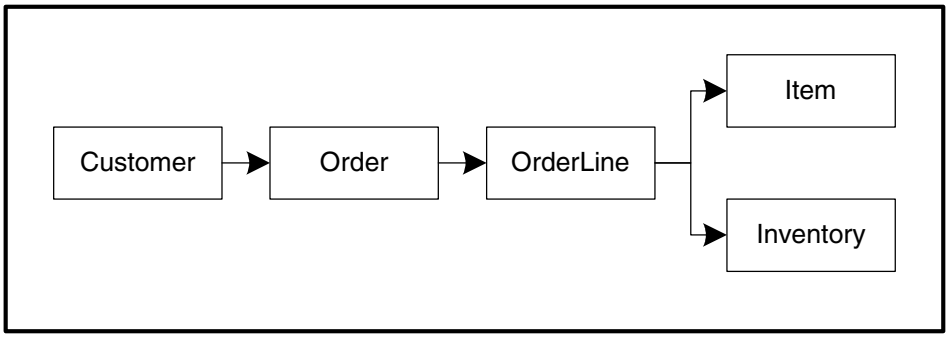



**Figure 4–4: Resynchronization Hierarchy Defined by DATA SmartLinks**

Before deciding how to organize your DATA links, determine your business focus. If you are interested in both Customer and Order records, which are more central? Are you interested in orders only in connection with a particular customer? Or is your focus the order itself, with customer information being secondary? In the first case, you would create your DATA link from the Customer object to the Order object. In the second case, the link would go in the opposite direction. How you organize the dependencies is a function of your business goals.

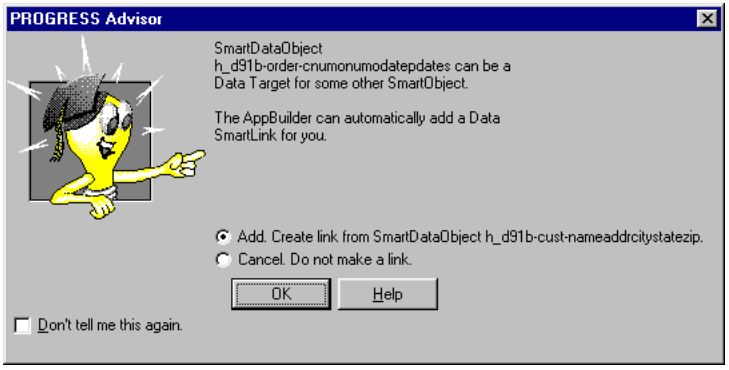
**NOTE:** The SmartBusinessObject itself can be the target of a DATA link from some SmartDataObject outside itself. The effect when resynchronization takes place is exactly as though both objects were combined—the resynchronization process propagates through both objects.

Follow these steps to create a synchronizing relationship between two SmartDataObjects within the SmartBusinessObject:

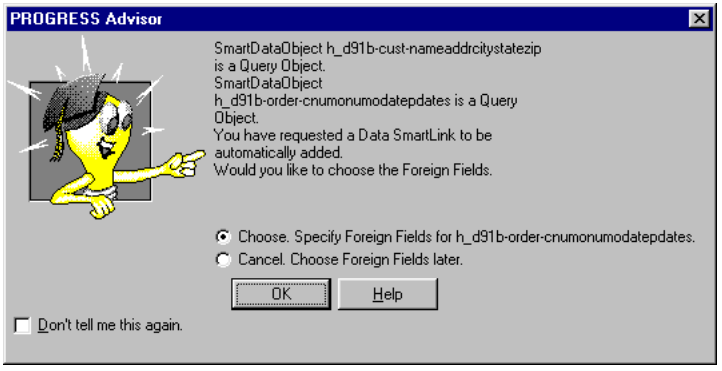
1 ♦ Create and place the first (controlling) object.

2 ♦ Create the second (controlled) object, being sure to include at least one field shared with the first object. For example, the CustNum field is common to both the Customer table and the Order table. That field (the foreign field) relates those two tables. Note that the field need not have the same name in both tables, although it often does.

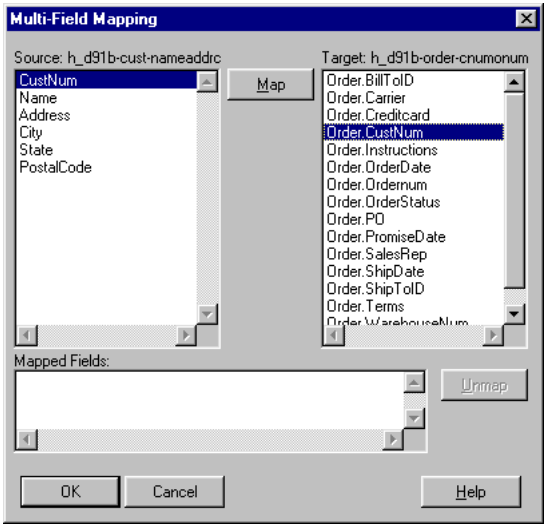
3 ♦ Place the second object. An Advisor dialog box appears and offers to create a DATA link from the first object. Accept the offer:



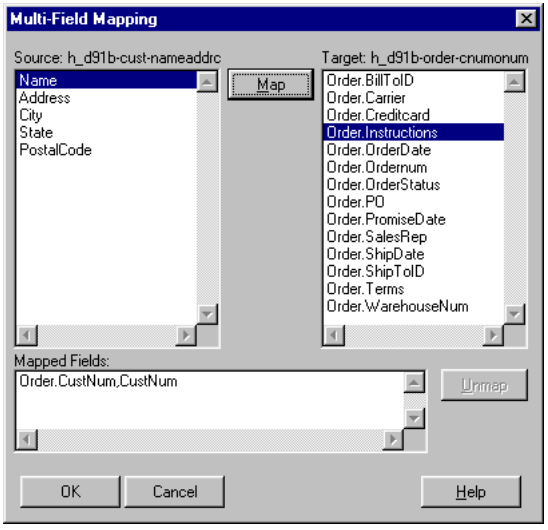

4 ♦ Another Advisor dialog box appears asking you if you wish to identify the foreign-field (relational) mapping. Unless you have a reason to defer this operation, accept the offer:



5 ♦ The Multi-Field Mapping dialog box now opens, presenting you with lists of available fields. Select the field common to both tables, for example CustNum:




6 ♦ Choose the Map button. Your selections disappear from the Source and Target lists and reappear as an associated pair in the Mapped Fields field:



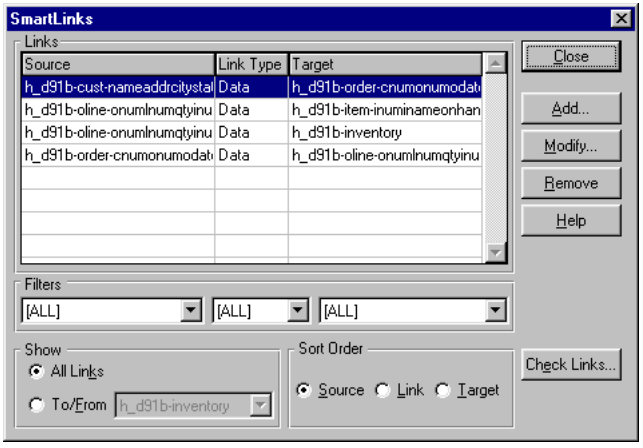

7 ♦ Create additional associations, if you so desire and there are suitable field pairs. When you have finished, choose OK to close the dialog box. Synchronization between these two SmartDataObjects is now established.

8 ♦ Repeat from Step 2 for each additional SmartDataObject you place, creating a synchronizing DATA link only from the immediately preceding object.

**NOTE:** The Advisor is not aware of your goal, so it will offer to create a link from any existing object. It is easy to become confused and select the wrong object. If you make such a mistake, use the SmartLinks Editor to correct it.

9 ♦ When you have linked all the SmartDataObjects appropriately, save your work.

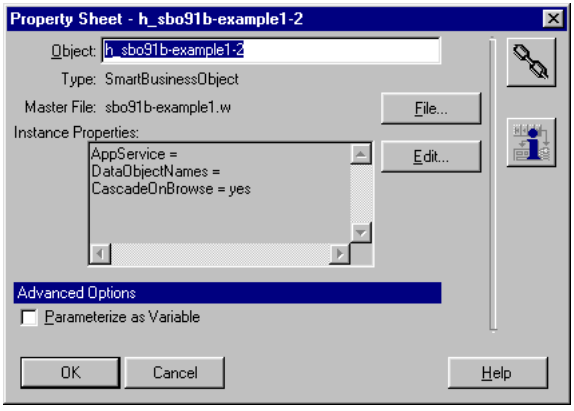
If you were to create the relationships shown in Figure 4–4, they would appear similar to this in the SmartLinks Editor:



### 4.3.4 Configuring a SmartBusinessObject Instance

When you have populated and created the synchronizing relationships in your SmartBusinessObject, follow these steps to configure its properties:

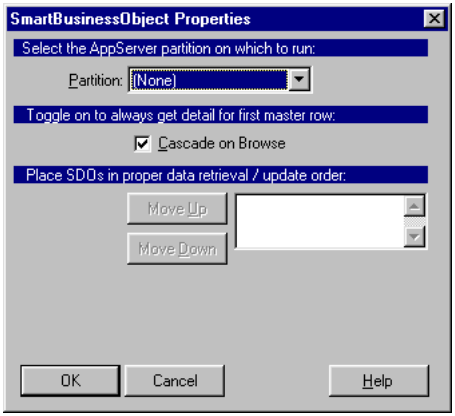
1 ♦ Click the menu button to open the SmartBusinessObject menu. Choose the Properties item. The properties dialog box opens:




2 ♦ Change the Object identifier to more accurately reflect the role of this SmartBusinessObject in your application.

3 ♦ If you wish to use the object as a placeholder, check the Parameterize as Variable box. See the "Creating a SmartObject Placeholder," section in Appendix A, "Frequently Used Dialogs," for further information about placeholders and resolving them at run time.

4 ♦ Choose OK to close the dialog box. Click the menu button again and choose the Instance Properties item. That dialog box opens:



5 ♦ Select the Partition to which this object will be assigned when running on an AppServer.

If there are no partitions listed, you can define as many as you need. Close the dialog box and use the AppServer Service Parameter Maintenance Tool, available from the PRO*Tools toolbar. The Partition tags that you define in this way are immediately made available to AppBuilder and will appear in this list when you reopen this dialog box.

**NOTE:** Unlike the SmartDataObject, a SmartBusinessObject always binds a dedicated AppServer connection for the entire application session. SmartBusinessObjects never run in connectionless mode. Because of how complex they are internally, having to repeatedly make and break momentary connections would be impractical.

6 ♦ If desired, clear the CascadeOnBrowse check box to improve performance when browsing. See the "How the CascadeOnBrowse Property Works" section for more detailed information about how this property affects the object's behavior.

7 ♦ Use the MoveUp/MoveDown buttons to arrange the list of SmartDataObject names to agree with the synchronization hierarchy you have defined. The first object (Customer, in the example shown in Figure 4–4) must be at the top of the list. The order of objects at the same level (in Figure 4–4, Item and Inventory are at the same level) is not significant.

8 ♦ Choose OK to close the Instance Properties dialog box.

9 ♦ Save your work.

### 4.3.5 How the CascadeOnBrowse Property Works

When this property is set (the default), navigation changes propagate downstream to every object in the DATA links chain. Clearing this check box causes navigation changes to propagate downstream only until more than one record is returned.

Using the situation in Figure 4–4 as an example, selecting a different customer causes that customer's orders to be found and displayed. That in turn causes the order lines for the first order to be found and displayed and finally the items and inventory records for the first order line.

Keeping the entire data stream constantly synchronized in this way can represent a significant overhead expense. It can slow your application unacceptably, especially if the user is looking for a particular record rather than examining each one.

To reduce the problem, you can choose to clear the CascadeOnBrowse property.

In Figure 4–4, if the user selects a different Customer record, the Order object will be directed to fetch the orders associated with this new customer. If there is only one order record, the OrderLine object will in turn be directed to fetch the records associated with that order. If there

is only one order-line record, the Item and Inventory objects will fetch the records associated with that order line.

But if, for example, the OrderLine object returns more than one record, then synchronization stops there. The Item and Inventory objects are not instructed to resynchronize themselves. The Item and Inventory record data for the previous Customer are simply discarded from the temp-table because they are not relevant to the current Customer.

To achieve additional levels of detail when CascadeOnBrowse is not set, the user must select one of the records that stopped the resync operation. This will cause the SmartBusinessObject to start resyncing again, from that level downward. Of course, if the next level below that point also returns multiple records, then resync would again stop and have to be restarted again from that new level.

So, if you believe the users of your application would prefer completeness at some cost in performance, leave CascadeOnBrowse set. If you believe users would prefer higher performance at the cost of having to explicitly restart resynchronization when needed, clear the property.

**CAUTION:** If you clear this property, it might not be immediately obvious to the user why resynchronization propagates fully in some cases but not others. To limit the number of support calls you receive, you might wish to explain this behavior in your documentation and training.

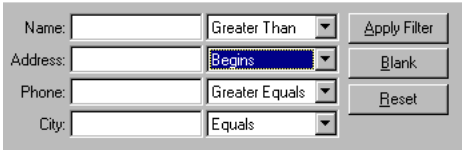
## 4.4 SmartFilters

The query you build into your SmartDataObject might return thousands—or even millions—of records. A return of such size can totally overwhelm users, causing them to fervently wish for a way to further subset the data stream. AppBuilder offers an ADM SmartObject to deal with that problem: SmartFilter.

SmartFilter provides a query-by-form (QBF) interface to the data stream supplied by your SmartDataObject. Using SmartFilter, the user of your application can narrow the query as much as desired, possibly causing the query to return only a few records or even no records at all. Being able to focus the query more tightly can be very convenient for the user.

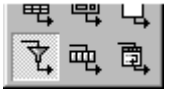
As seen by the user, SmartFilter is rather simple. Each filterable field is represented by one or two editable fields representing a value and a comparison operator. The user controls the object's behavior using three buttons:

- **Apply Filter** — Immediately activates the filter, reducing the size of the data stream. To clear the effect of a filter, the user must choose Blank and then Apply.
- **Blank** — Clears all editable fields and resets the operators (if visible) to their default states, ready for the next setup.
- **Reset** — Restores the most recent filter expressions. Does not clear the fields.



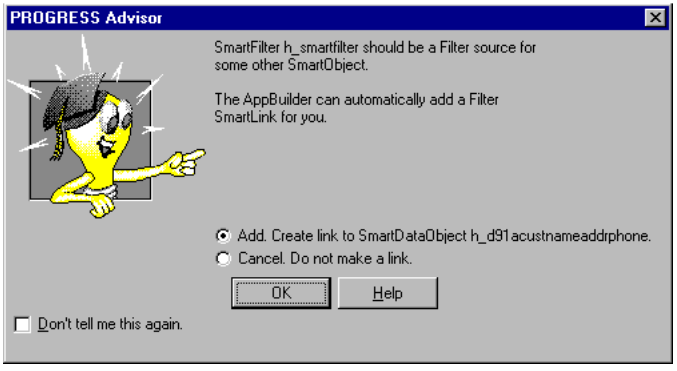

SmartFilter is supplied as a pre-compiled master rather than as a template. To include an instance of SmartFilter in your application, follow these steps:

1 ♦ Click on the SmartFilter tool icon in the Object Palette:



2 ♦ There might be a short pause while AppBuilder creates the instance. Position the cursor over the workspace and click to place the instance.

3 ♦ If you previously placed a SmartDataObject, AppBuilder now opens an Advisor window offering to create a Filter link to it. Unless you have reason not to do so, accept the offer:



4 ♦ Inspect the properties for this SmartFilter and change any that do not meet your needs. See the "Configuring SmartFilter Properties" section in this chapter for information about how to do that.

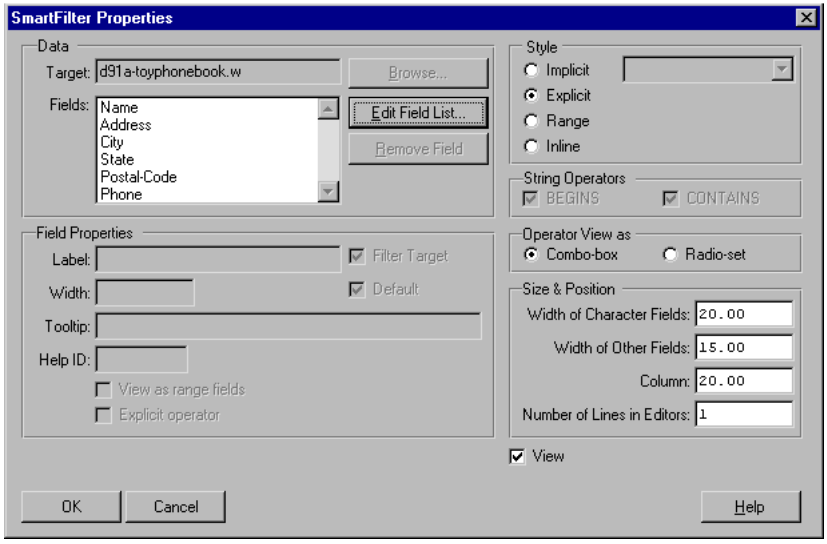
5 ♦ When you have finished making changes, choose OK to dismiss the properties dialog box.

6 ♦ Click the object to select it. Drag the handles to position and size the object to suit your layout. Your SmartFilter is now ready for use.

### 4.4.1 Configuring SmartFilter Properties

There are a number of changes you can make to the appearance and behavior of a SmartFilter instance through the Properties dialog box. This section discusses the options available to you.

To open the dialog box, choose Instance Properties from the menu-button menu. The dialog box appears as shown here:




#### Data

This area identifies the data source, and the fields to be filtered:

- For your convenience, if you place the SmartFilter object into your workspace after you place the SmartDataObject, AppBuilder automatically fills in the Target field for you. This is generally the best way to do it.

But if you place the SmartFilter instance first, you can still fully configure it. Click the Browse button and, when the Choose dialog box opens, select the appropriate SmartDataObject and choose OK. The object filename immediately appears in the Target field, and a list of database fields supplied by that master populates the Fields field. Note that you are not in any way attaching that data source at this point; you have merely identified it so that you can configure your SmartFilter.

- By default, none of the fields supplied by the SmartDataObject are filterable.

  Choose Edit Field List and select the fields to make available for filtering. They will appear in the Fields list. If you change your mind about one of the fields, simply select it and choose Remove. Removing the field from the filterable list has no effect on the data stream or the SmartObject assigned to display the data. To restore a field to the filterable list, choose Edit Field List and add it back.

  Note that performance might be unacceptably low if the number of records is large and you allow the user to filter on fields that are not indexed.

### Style

You can define the amount of control the user has over the filter operation by setting the Style value. The Style you choose applies to all fields unless you modify the setting for a particular field.

The possible values for Style are:

- **Implicit** — Choosing this option removes the relationship operator from the user's control. The user still selects the value against which the filter tests, but cannot change the nature of the test (equality, inequality, etc.). You set the type of test at design time using the related combo box.

  By default, the operation you choose in the combo box is ignored for character string fields. Character strings undergo a BEGINS test regardless of the setting in the combo box. If you want character strings to be tested using the operator you set in the combo box, uncheck the BEGINS box in the String Operator frame (below the Style radio buttons).

  When Style is set to Implicit, you can set individual fields to be Ranges, instead. See the View as Range Fields option in the "Field Properties," section.

- **Explicit** — The default Style. Choosing this style allows the user to select the type of test on a per-field basis. The user selects the operator from a combo box (default) or by setting a radio button; see the "Operator View As" section.

Because BEGINS can be applied on a per-field basis for character fields (it appears in their list of operator choices), the global BEGINS option is disabled when Style is set to Explicit.

You can set individual fields to be Ranges. See the View as Range Fields option in the "Field Properties" section.

- **Range** — Choosing this style causes the SmartFilter to display two fill-ins for each field, representing the upper and lower bounds of a range. The user types in the limiting values, which are considered **inclusive**. For example, entering c as the lower bound and **e** as the upper, for a CHARACTER field, will return all records where the value for that field is in the range c* through e, inclusive. Note that while e itself would be included, e* is not. Similarly, entering bounds of 10 and 100 for an INTEGER field would return all records where the value of that field is in the range 10 – 100, inclusive.

  When Style is set to Range, you can set individual fields to be Explicit, instead. See the Explicit Operator option in the "Field Properties" section.

- **Inline** — Choosing this style requires that the user enter the comparison operator along with the value. For example, to test for INTEGER values over 10, the user must type in **> 10**.

  If the user enters a value without an operator, for example **10** rather than **>10**, SmartFilter will presume the intended operator is EQUALS. The only exception is if the field is of type CHARACTER and the BEGINS check box is set. In that case SmartFilter will perform the BEGINS test instead.

### String Operators

When these check boxes are enabled, setting them overrides any conflicting setting in Style.

- **BEGINS** — When checked, SmartFilter will perform a BEGINS test against CHARACTER fields

- **CONTAINS** — If the field being filtered is word-indexed, SmartFilter will represent the field as an Editor widget sized according to the Number of Lines in Editors setting in the Size & Position section.

### Operator View As

By default, the SmartFilter lists the operators for Explicit fields in a combo box. You can force the options to be listed as radio buttons instead. To choose a representation, set the appropriate radio button.

### Size & Position

Allows you to set three visual characteristics of the SmartFilter instance:

- **Default Field Widths** — Set the displayed width of the editable portion of the filter fields. You can set the width separately for non-CHARACTER fields, and can override these settings on a per-field basis in the Field Properties section. The width you set here only affects the visible portion of the field, and it has no effect outside the SmartFilter itself.

- **Field Position** — You can control where the editable portion of the field starts. To minimize the amount of space the filter takes up in the X dimension, set this value to the character count of the longest field label, plus two. For example, if your longest label is "Postal-Code:", set the value to 14. If you use a smaller number, the field will hide the right end of one or more labels. If you use a larger number, both the fields and their labels will move to the right within the SmartFilter's frame.

- **Number of Lines in Editors** — If SmartFilter represents any field as an Editor widget, this value is the number of lines that will be displayed. If you set this value to 1, the widget will display with the same height as a Fill–in, and it will have no word-wrapping. If you set the value to 2 or more, the field will have word-wrapping and a vertical scroll bar.

### Field Properties

Each filterable field has several default properties that you can change. Note that such changes have no effect on the database, the data stream, or how the data-display object treats the field. All changes are local to the SmartFilter.

To begin, select the field whose properties you wish to change:

- To change the label that identifies the field, uncheck the Filter Target box and type in the new text for the label. By default, the label is inherited from the fieldname in the schema.

- To change the field width, uncheck the Use Default box and type in the new value in character units. The default value is taken from the field definition in the schema.

- You can add text for a Tooltip identification by typing it into that fill-in.

- If you plan to provide context-sensitive help, enter a unique INTEGER value for this field in the Help ID fill-in.

- The filter operators (equals, greater than, etc.) all refer to a single value. To select instead for values that fall within a certain range, check the box View as Range Fields. This option is available only when Style is Implicit or Explicit.

  When View as Range Fields is checked, a second editable field appears in place of the operators list.

- You can override the Range style on a per-field basis by setting the Explicit Operator box. The user will then see a single fill-in field instead of two, and be able to choose the comparison operator from a combo box.

## 4.5 Making Changes to the Data Stream

The SmartDataObject provides data to the SmartDataBrowser and SmartDataViewer for display. In turn, the SmartDataViewer or SmartDataBrowser might provide data changes that must be written back to the database by the SmartDataObject. To preserve their ability to communicate with one another, you must make any changes to the data stream in an orderly way. Table 4–1 summarizes possible changes and the order in which you must make them.

**Table 4–1: How to Make Changes to the Data Stream** *(1 of 2)*

| Your Goal | How to Reach It |
|---|---|
| Get exactly the same fields, but from a different database. | Change the database name in the SmartDataObject. |
| Get exactly the same fields, but arranged in a different order than before. | Simply recompile. |
| Stop getting one of the fields. | If the field is being displayed, remove the field reference from the SmartDataBrowser and SmartDataViewer first, then remove it from the SmartDataObject.<br>If it is not being displayed, simply recompile. |
| Start getting a new field. | Add the field reference to the SmartDataObject first. Then add it to the SmartDataBrowser and SmartDataViewer, if you want them to display it. If you do not want them to display it, simply recompile. |

**Table 4–1: How to Make Changes to the Data Stream** *(2 of 2)*

| Your Goal | How to Reach It |
|---|---|
| Make a field updatable. | Make the field updatable in the SmartDataObject first. Then make it updatable in the SmartDataBrowser and SmartDataViewer. |
| Make a field not updatable (read-only). | Make it read-only in the SmartDataBrowser and SmartDataViewer first. Then make it read-only in the SmartDataObject. |

## 4.6 Creating a Freestanding Query Object

Besides the queries embedded in and bound to other objects such as the SmartDataObject, you can create a Query that exists by itself. Other objects can use such a freestanding Query object by referring to its handle.

Such a Query object, like the SmartDataObject, has no run-time representation. It is visible in the workspace only at design time.

To create a free-standing Query object, follow these steps:

1 ♦ Click on the Query tool button in the Object Palette:



2 ♦ Place the object anywhere in the workspace—it is not visible at run time. In general, it is a good idea to place it in an unoccupied corner, if you can, in case you need to gain access to it later. The Query object is represented by this icon in your workspace, at design time:



3 ♦ When you click to place the new Query object, the Query Builder dialog will open. Use that dialog box to compose the terms of the query. See the "Defining and Editing Queries Using Query Builder" section in this chapter for further information.

## 4.7 Defining and Editing Queries Using Query Builder

A central step in developing any database application is creating the query or queries to be used for retrieving records from the database. AppBuilder offers a structured query editor for your use. You can invoke Query Builder in a number of ways:

- **Define a new SmartDataObject** — AppBuilder then starts the SmartDataObject Wizard, and the Wizard calls Query Builder on your behalf. SmartObjects are the most powerful way to use queries.

- **Place a basic Data Browser (Browse) object in the workspace** — AppBuilder then starts Query Builder. The Browse object has all the internal logic necessary to make use of the query, though only in a self-contained way. You must manually add any additional logic you require.

- **Place a Query object in the workspace** — AppBuilder then starts Query Builder. Such a Query object is not connected to anything after you define it. You must manually add the code to make use of it.

- **Choose the Query button from the property sheet of a Frame object** — AppBuilder then starts Query Builder. As is true of the Query object, you must manually write the code that makes use of this embedded query, after you have finished defining it.

**NOTE:** You can also construct or modify queries by hand, using AppBuilder's Section Editor, but such queries cannot later be edited using Query Builder. If you wish to have all queries be maintainable in Query Builder, you would have to replace any such free-form queries with their Query Builder-generated logical equivalents.

Query Builder always starts in Table-selection mode.

### 4.7.1 Table Mode

In this mode, the Query Builder allows you to specify the tables you want in the query. When you modify a query from the property sheet of a frame or browse, the Query Builder also initializes to this mode. Figure 4–5 shows how the Query Builder appears in Table mode.

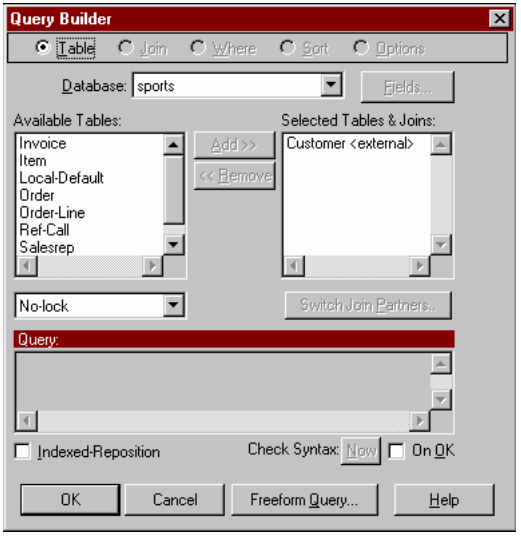



**Figure 4–5: Query Builder in Table Mode**

To select a database table, select the table name in the Available Tables list and choose the Add button; you can also double-click on the name.

To remove a table from the Selected Tables & Joins list, select the table name and choose the Remove button; you can also double-click on the name.

After you select the tables you want, the AppBuilder displays the query in the Query panel. For example, if you choose the Customer, Order, and Order-Line tables, the Query field appears, as shown in Figure 4–6.

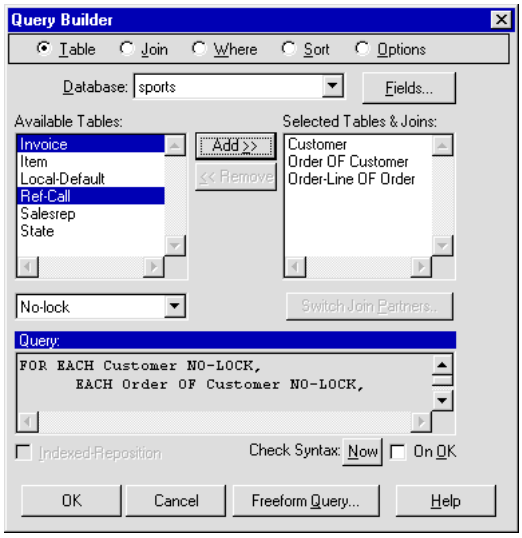



**Figure 4–6: Sample Query for OF-joinable Tables**

As you select tables, the AppBuilder establishes the relationship among them and the other tables that you have selected. If two tables share a common key with the same name and data type, they are OF joinable. If they are not OF joinable, you must establish how they will be joined. For example, if you choose the Customer and Local–Default tables, the AppBuilder displays the query in the Query panel, as shown in Figure 4–7.

```
FOR EACH sports.Customer NO-LOCK,
  EACH sports.Local-Default WHERE TRUE /* Join to Customer incomplete */
NO-LOCK:
```

**Figure 4–7: Query Sample with Incomplete Join Criteria**

The comment indicates that you must complete the relationship of the Local-Default table to the Customer table. Complete the join criteria by choosing the Join button to enter Join mode.

If you select three or more tables and AppBuilder can establish the relationship among them, the AppBuilder still allows you to choose a table and change its join criteria. You do this by selecting the table in the Selected Tables & Joins list and choosing the Switch Join Partners button.

### 4.7.2 Join Mode

When you activate the Join radio-set button in the Query Builder dialog box, you enter Join mode, and the Query Builder appears as in Figure 4–8.

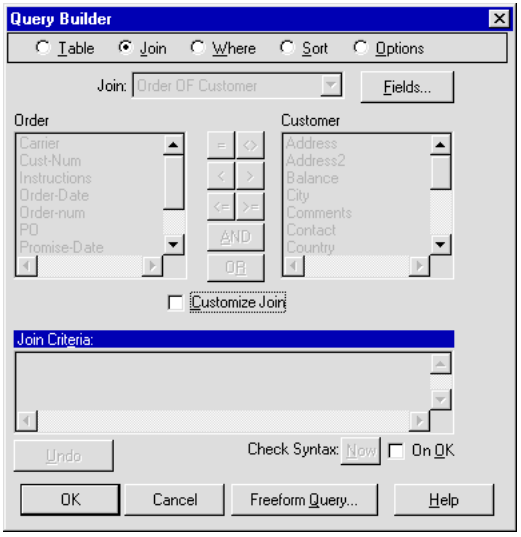



**Figure 4–8: Query Builder in Join Mode**

To enter this mode, you must already have selected at least two database tables. If you select more than two tables, you can use the down arrow button in the Join combo box to choose the combination of tables you want to join.

If two tables are OF joinable, you can change their join criteria by activating the Customize Join toggle box to establish a WHERE clause between the two tables. Your comparison options are equal, not equal, less than, greater than, less than or equal, and greater than or equal. To use one of these options, double-click in a field from one of the tables, double-click on one of the comparison buttons, then click in another field from the other table.

As you establish comparisons between fields, the AppBuilder displays the 4GL code in the Join Criteria panel. When you choose OK, the AppBuilder displays the code in the Query Builder's Query field (and in the widget's property sheet, if the query is an embedded one). You can link the comparisons with AND or OR.

If you are familiar with the Progress 4GL, you can edit the Join Criteria field by clicking in it and typing in the code directly.

### 4.7.3 Where Mode

To specify search criteria for a table, activate the Where radio-set button in the Query Builder dialog box. The Query Builder enters Where mode and appears as in Figure 4–9.

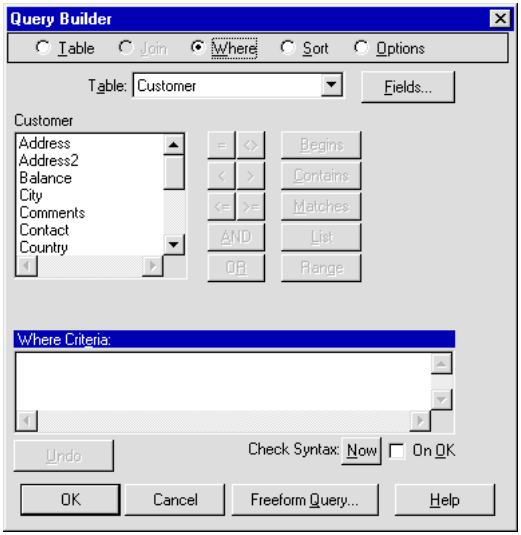



**Figure 4–9: Query Builder in Where Mode**

First, select a field from the list of fields. The AppBuilder displays the possible comparison operators for that field. These operators are sensitized based on the type of field.

For example, if you select the Cust-num field, the Contains button is not sensitized because Cust-num is not a word-indexed field. Also, Begins and Matches are not sensitized either, because Cust-num is an integer field.

After you select a field, choose one of the enabled comparison buttons. A dialog box appears that lets you supply a constant for comparison (you cannot specify variables). If you choose either the List or Range button, the dialog boxes that appear allow you to specify whether the list or range is inclusive or exclusive. The AND and OR buttons extend the relationships between logical fields and also let you combine logical phrases with a complex expression.

The AppBuilder displays the 4GL text of the clause in the Where Criteria editor. When you choose OK, the AppBuilder displays the code in the Query Builder when it is in Table mode, and in the widget's property sheet if the query is an embedded one.

If you are familiar with the Progress 4GL, you can type the search criteria directly into the Where Criteria panel.

### 4.7.4 Sort Mode

To specify the sort order you want for your query, activate the Sort radio-set button. The Query Builder enters Sort mode and appears as in Figure 4–10.

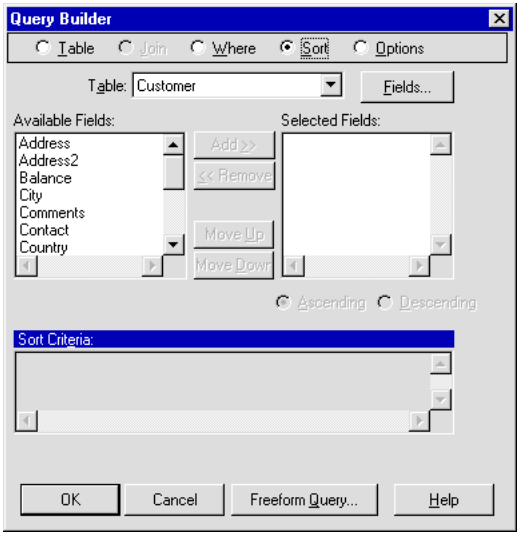



**Figure 4–10: Query Builder in Sort Mode**

**NOTE:** If your query uses the {&SORTBY–PHRASE} preprocessor value to specify a sort order for the final result, the AppBuilder prevents you from entering this Sort mode by making the Sort button insensitive. You can set the query to use the {&SORTBY–PHRASE} from the Options mode of the Query Builder. For more information, see the "Options Mode" section in this chapter.

To specify the sort criteria, select fields from the Available Fields list. The Table combo box allows you to move between tables if you chose more than one table while in Table mode.

In the Sort Criteria editor, AppBuilder displays the fields you choose and the sort criteria you specify. When you choose OK, the sort criteria is also displayed in Table mode. The Ascending and Descending radio buttons allow you to specify the display order of the sorted records.

### 4.7.5 Options Mode

To specify query-tuning options and other query parameters, choose the Options button. Query Builder enters the Options mode and appears as in Figure 4–11.

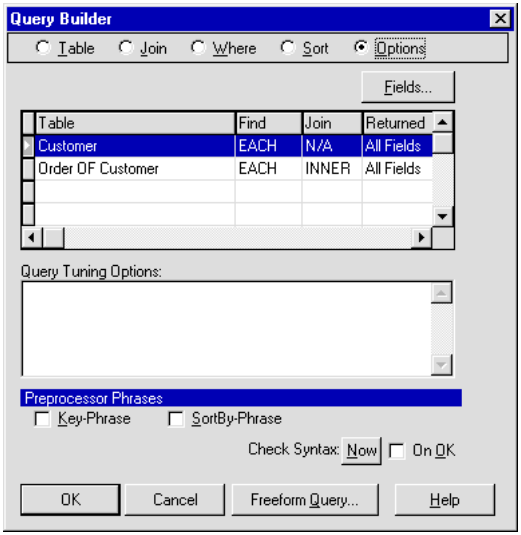



**Figure 4–11: Query Builder in Options Mode**

#### Editable Find, Join, and Field List Criteria

To support the QUERY FIELD–LIST feature, Query Builder displays an editable browse with the following fields:

**Table** — Specifies the tables in the query. This field is not editable.

**Find** — Specifies the find criteria (EACH, FIRST, or LAST). This field is editable.

**Join** — Specifies the type of join (INNER or OUTER). This field is editable.

**Returned** — Specifies which fields are returned (included in the field list) when records are fetched to satisfy the query (All Fields or Fields Used). This field is editable.

To change the Find field, double-click in it until the desired value appears or type **E**, **F**, or **L**.

To change the Join field, double-click in it until the desired value appears or type **I** for INNER or **O** for OUTER.

The Returned field is a logical field that can be set in one of two modes: All Fields (the default) and Fields Used. When you select All Fields, the query fetches all fields of the table. When Fields Used is selected, the query returns only those fields in the FIELDS–IN–QUERY Preprocessor list. For browses, this is controlled in the Column Editor; for frame queries, this is controlled by what fields you place in the frame.

To select All Fields, either double-click in the field until All Fields appears or type **A**. To select Fields Used, double-click in the field or type **F**.

### Editable Query Tuning Options

You specify query-tuning parameters by entering your choice of parameters in the Query Tuning Options panel. Depending on your choices, the AppBuilder inserts code like the following into the OPEN–QUERY statement:

```
LOOKAHEAD
NO-DEBUG
INDEX-HINT
```

Query-tuning parameters correspond to the QUERY–TUNING keyword. This keyword is available with the FIND, FOR EACH/LAST/FIRST, and OPEN–QUERY statements. For more information, see the *Progress Language Reference*.

## 4.8 Selecting Database Fields for Browsing

When you create a query associated with a Browse, Query Builder makes an additional dialog box available to you: the Column Editor. Using the Column Editor, you can choose fields to be displayed by the browser object, impose an ordering, and define other display characteristics such as colors, formats, and so forth.

Query Builder provides access to the Column Editor through the Fields button. This button appears at the right of the Database combo-box field, as shown in Figure 4–12.




**Figure 4–12: Column Editor Button in Query Builder**

Because columns are defined only in relationship to a table, this option is disabled until you have selected at least one table in Query Builder. When you choose this button, the Column Editor dialog box opens, as shown in Figure 4–13.

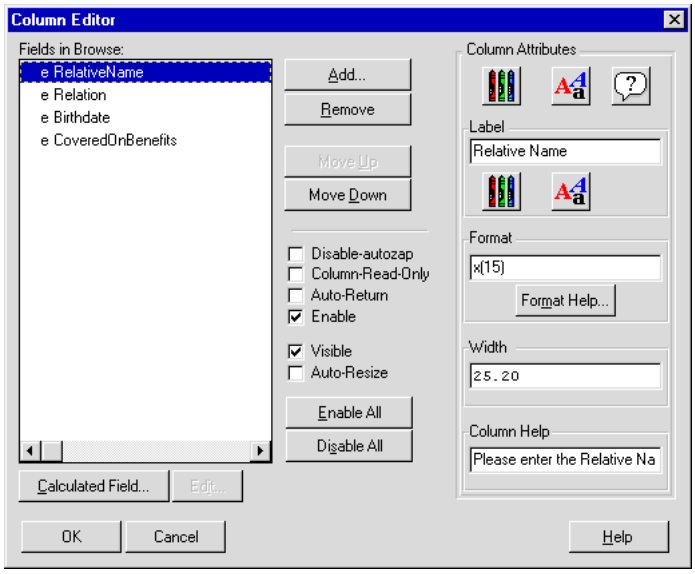
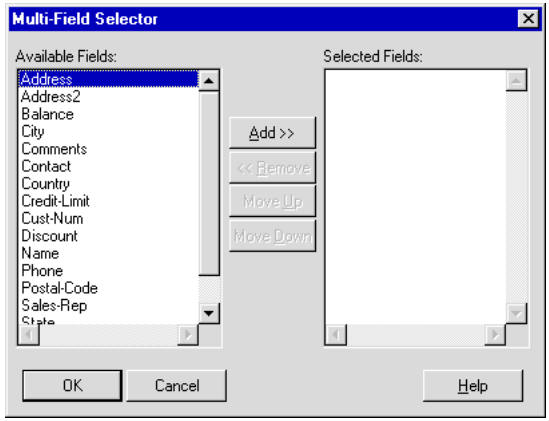



**Figure 4–13: Column Editor Dialog Box**

Initially, the Fields list is empty apart from the notice reassuring you that the empty state is normal.

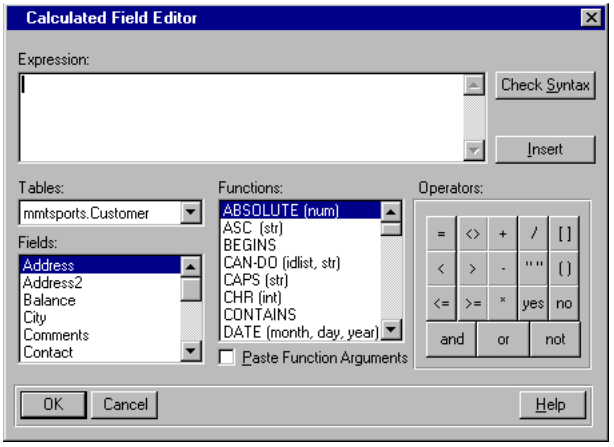
To select fields (columns) and define the conditions for their display, follow these steps:

1 ♦ Choose the Add button. The Multi-Field Selector dialog box opens:




Choose the fields you wish to display and move them to the Selected list. You can move them by double-clicking, or by selecting them and choosing Add. Hold down the **CTRL** key to select more than one field at a time. Use the Move Up/Move Down buttons to reorder the list. When finished, choose OK.

2 ♦ If you wish to create and display calculated fields, choose the Calculated Field button. The Calculated Field Editor dialog box opens:




Calculated fields are fields whose content does not come directly from the database. The content of a calculated field represents the result of some transforming expression applied to the content of one or more other fields, which can themselves be calculated fields. For information about how to use this dialog box to create calculated fields, see "Creating Calculated Fields" in this chapter.

3 ♦ Set the Enable check box for each updatable field. If you wish to enable or disable all fields at once, choose the appropriate button. When you select a field that is enabled for update, you can set additional characteristics:

- **Disable Autozap** — By default, a field is cleared when the user begins typing into it. This characteristic is called *autozapping*. Set this check box to disable such automatic clearing of the current field.
- **Column Read-only** — Set this check box to temporarily disable update on the field (column). Fields must be enabled at design time if they are to be enabled at all; you cannot first enable them under program control.
- **Auto-Return** — Automatically moves the input focus to the next field when this field is full. If this field is the last field, having Auto-Return set causes a GO event to be generated.

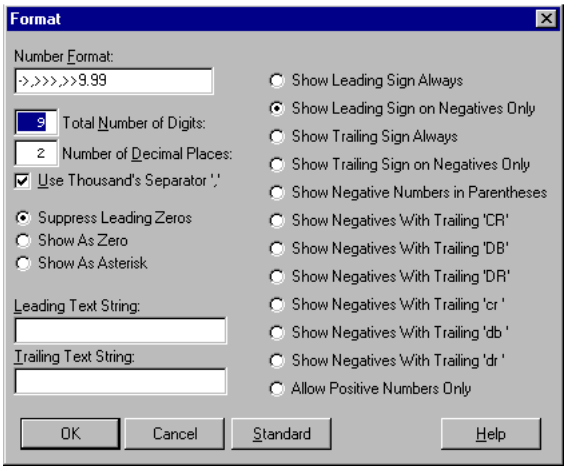
4 ♦ Clear the Visible check box for any field that is not to be seen at initialization. Fields are Visible by default.

5 ♦ Set the Auto-Resize check box if you want the field to be able to change its size to agree with changes in certain run-time conditions, such as font size or text format.

6 ♦ Set the font, colors, and string attributes for the data to be displayed in the column.

7 ♦ Set the string value, font, and colors for the label that will appear at the top of the column.

8 ♦ Set the format for the data to be displayed in the column. Choose the Format Help button to see the Format dialog box:



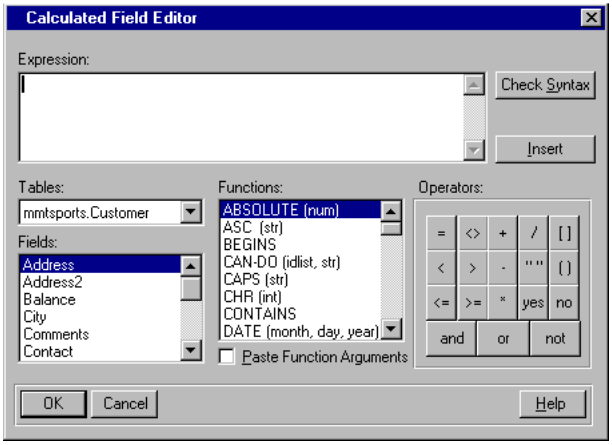

9 ♦ Set the Column Help string. If the parent window of the Browse has a status area defined, this string will appear there whenever a field in this column has the input focus. If the parent window has no status area defined, this string does not appear. If this field is left blank, the Browse displays a generic prompt.

### 4.8.1 Creating Calculated Fields

Very often, when creating applications, you will want to present a field to the user that is the result of some transformation applied to one or more other fields. Current value of inventory is one possible case, and invoice line-item total is another. The number of such potential uses is large.

Column Editor allows you to create such calculated fields in a straightforward way. To create a calculated field, follow these steps from the Column Editor's dialog box:

1 ♦ Click the Calculated Fields button. The dialog box opens:




2 ♦ Compose the expression whose result will appear in the field. You can compose by typing directly into the field, by double-clicking on tokens (Fields, Functions, Operators), or by some combination of these two methods. The expression can be of any complexity. Choose OK when the expression is complete. The dialog box closes.

3 ♦ The newly defined field appears in the list of fields with the default identifier CALC, a default type of CHARACTER x(8), and no label. Change the identifier to one more meaningful in the context of your application. Add a label, if appropriate.

4 ♦ Change the data type and format, if necessary, to reflect the nature of the data. For example, if the expression yields a floating-point result, change the data type to DECIMAL and the format string to one that portrays DECIMAL values.

5 ♦ If appropriate, choose Advanced and enter a Help string for this field.

6 ♦ Use the Move Up/Down buttons to put the new field into the correct relationship to the other fields that will display.

**NOTE:** Calculated fields are created in a not-updatable state. Although you can set them to be updatable, in general you should not: a calculated field is a scratch field, and is undefined in the context of the database. It can be easy to forget this. If you do choose to make a calculated field updatable, you must write the code yourself that will interpret any changes that a user makes.

# 5

# Data-display/Capture Objects

Presenting data to the user in a way appropriate for inspection or update is a major activity in a database application. Typically, the same objects that present data also capture new and changed data. This chapter describes the data-display/data-capture objects AppBuilder supplies for your use:

- SmartDataBrowser and SmartDataViewer
- SmartDataField and SmartSelect Objects
- Basic Data Browser and Data Viewer Objects
- Basic and OCX Combo Boxes
- Single-line and Multi-line Editors
- Radio-button Set and Toggle (Check) Box Objects
- Selection List Object
- Slider Control and ActiveX Spin Control Objects

## 5.1 Introduction to Data-display Objects

Presenting data to the user is an important intermediate goal in nearly any database application. Presentation techniques range from the most generic row/column display to sophisticated, application-specific graphing and charting.

AppBuilder provides four SmartObject building blocks:

- Dynamic and Static SmartDataBrowsers, for presenting data to your users in a simple, tabular row/column format
- SmartDataViewer, for presenting one record at a time, in a layout you determine
- SmartDataField, including the dedicated SmartSelect, for adding ADM capability down to the individual field level in the SmartDataViewer

Additionally, the Progress 4GL provides a number of basic objects that display and capture data. These include:

- Data Browser with built-in Query, for representing data in tabular row/column format
- Combo Box, for presenting a list of choices
- Data Viewer (DB–Fields) with built-in Query, for representing data on a per-record basis
- Editor (Multi-line editor), with most of the capabilities of a standard text editor
- Fill–in (Single–line editor), for capturing a small amount of data
- Radio–button Set, for representing an exclusive set of (usually nonstring) choices
- Selection List, for presenting a list of choices
- Slider Control, for representing a point within a subrange of INTEGER
- Toggle Box (Check Box), for representing a boolean (LOGICAL) choice

This chapter discusses these building blocks and their role in your applications.

## 5.2 SmartDataBrowsers

SmartDataBrowsers are Browse widgets enhanced with ADM Smart technology. A SmartDataBrowser presents the results of a query in simple row/column tabular format. Each row displayed in a SmartDataBrowser always represents one record from the data stream (the record itself might be a dynamic composite created by a JOIN operation). Each column represents one of the fields in the records. The SmartDataBrowser can display any of the standard data types that can be represented by ASCII text.

A SmartDataBrowser displays and possibly updates data supplied by a SmartDataObject to which the SmartDataBrowser is connected by SmartLinks.

AppBuilder supplies two types of SmartDataBrowser for your use:

- **Dynamic** — You place an instance of the predefined dynamic SmartDataBrowser in your application. This object is able to connect to and interact with any SmartDataObject.
- **Static** — Using the Wizard provided by AppBuilder, you define a static SmartDataBrowser master object to connect to and cooperate with a particular SmartDataObject. You later select and place an instance of the master object in your application.

## 5.3 Dynamic SmartDataBrowser

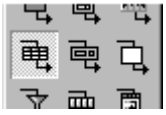
The dynamic SmartDataBrowser is capable of displaying and updating records supplied by any SmartDataObject. You make the connection and set up the conditions at design time.

### 5.3.1 Creating a Dynamic SmartDataBrowser Instance

Creating, placing, and configuring a dynamic SmartDataBrowser is a very straightforward process. Follow these steps:

1 ♦ First create and place an appropriate SmartDataObject to supply the data stream the Browser will display.

2 ♦ Right-click the SmartDataBrowser tool icon in the Object Palette:



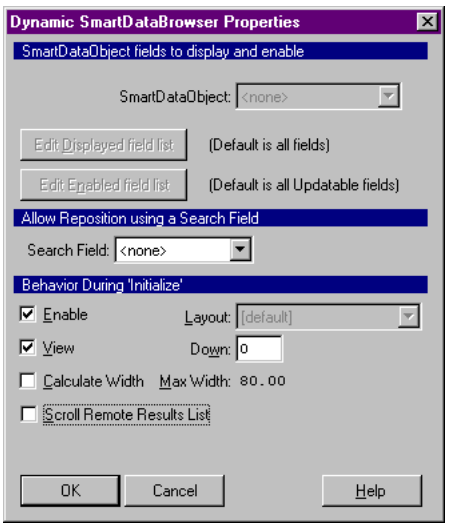
Choose Dynamic SmartDataBrowser from the menu that opens.

3 ♦ Position your mouse cursor over the workspace and click to place the object. It appears as a Browse widget, initially with only a single column:



4 ♦ AppBuilder immediately opens an Advisor window if it finds opportunities to create SmartLinks to other objects. Examine the recommendations and accept all that seem appropriate.

5 ♦ Click on the menu button and select Instance Properties from the menu. That dialog box opens:




6 ♦ If you wish to display or allow updates to fewer fields than the SmartDataObject would permit, make those changes now. Choose the appropriate button to open a selector dialog. See Table 5–1 for a description of how Progress makes use of your choices.

**Table 5–1: Dynamic SmartDataBrowser Display/Update Rules**

| Your Choice | Progress Response | |
|---|---|---|
| | **Display** | **Update** |
| Make no changes | Display all fields provided by the SmartDataObject (the value of the DataColumns property). | Allow updates on all fields enabled in the SmartDataObject (the value of the UpdatableColumns property). |
| Specify only the display list. | Display the fields you chose. | If there is a TableIO source, then allow updates on all fields that are enabled in the SmartDataObject, if they are also being displayed.<br>If no TableIO source, then allow no updates at all. |
| Specify only the enabled list. | Display all fields provided by the SmartDataObject (the value of the Data Columns property). | Allow updates on the fields you chose. |
| Specify both the display list and the enabled list. | Display the fields you chose. | Allow updates on the fields you chose. |

7 ♦ If you wish to allow the user to search on one of the displayed fields, select it from the Search Field combo box.

8 ♦ Clear the Enable and View check boxes to initialize the object in a hidden, insensitive state.

9 ♦ Set the Down value to the number of rows you wish the Browser to display. If you wish the object to determine its own best width, check the Calculate Width box and, if desired, enter a Max Width limiting value.

**NOTE:** The design-time representation does not account for the Down and Width choices you make, so you cannot predict the run-time appearance from it. You will need to experiment to determine how to get the run-time appearance you desire. A good first approximation for number of lines might be to show 1 or 2 more lines than your Down setting (1 to account for the column-labels line, and a second if there will be a horizontal scroll bar)

10 ♦ Unless the total number of records in the data stream is same or less than the Rows To Batch value, the user will not be able to use the scroll bar to reach the last record. The last record reachable by using the scroll bar is the last record in the buffer, not the last record in the data stream. This can be confusing to the user.

To solve this problem, check the Scroll Remote Results List box. You might also have to write some code. For more information see the "Scrolling to the Last Record in the Data Stream" section.

Unless you need to add special logic, the Dynamic Browser will typically be a lower-cost choice than a Static Browser.

## 5.4 Static SmartDataBrowser

The static SmartDataBrowser you use in your application is an instance of a master object that you create. Each master SmartDataBrowser is designed to cooperate with a particular SmartDataObject master, exchanging a well-defined set of data fields in a particular order.

### 5.4.1 Creating a SmartDataBrowser Master

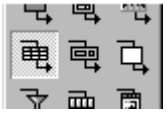
The first step in defining a new SmartDataBrowser master is always to identify the source of the data stream. Because of this dependency, you must first create your SmartDataObject, then create the SmartDataBrowser. See the "Creating a SmartDataObject Master" section in Chapter 4, "Data-access Objects", for information about that process.

AppBuilder supplies a Wizard to help with creating SmartDataBrowser masters. The Wizard has four pages, only two of which involve significant work:

1. Introduction
2. Identifying the data source
3. Selecting the fields to display from the data stream
4. Congratulations

To create a SmartDataBrowser master, follow these steps:

1 ♦ Start the SmartDataBrowser Wizard by clicking on the SmartDataBrowser tool icon in the Object Palette:



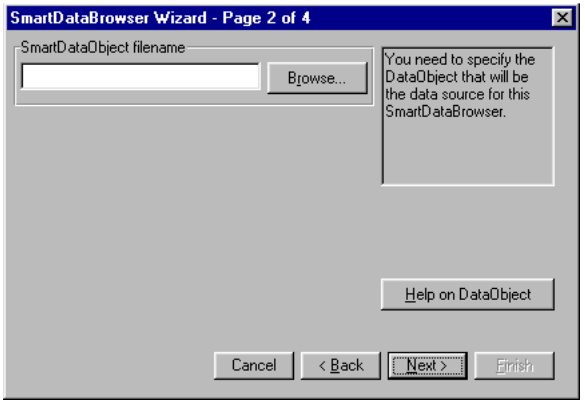
When the Choose dialog box opens, choose New.

**NOTE:** You could also start the Wizard by choosing File→ New and selecting the SmartDataBrowser item from the list, or by right-clicking the icon in the Palette and selecting New SmartDataBrowser from the menu that appears.

2 ♦ The SmartDataBrowser Wizard starts up, and displays its first page:



3 ♦ When you have finished reading the introductory text, choose Next to move to Page 2:

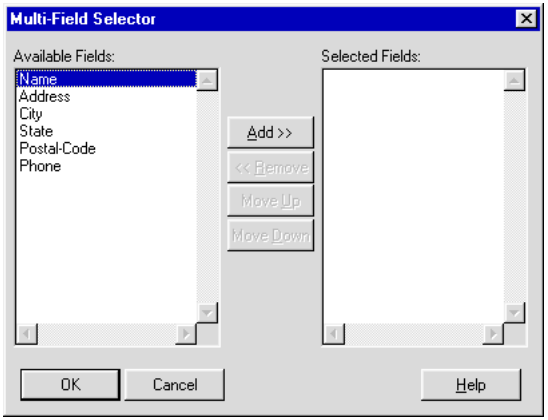



4 ♦ Choose the Browse button and select the appropriate SmartDataObject master. Its filename will appear in the fill–in.

5 ♦ Choose Next to move to Page 3:



6 ♦ Choose the Add Fields button to open a Multi-Field Selector dialog box:




7 ♦ The fields Available for display appear in the left-hand list. Select those you wish the SmartDataBrowser to display and click the Add button to move them to the Selected list in the right-hand window. If you make a mistake, click the Remove button.

When you have finished selecting fields, you can reorder them using the Move Up and Move Down buttons. When they are in the order you want them displayed, choose OK. The Selector dialog box closes and the list of selected fields appears in the Wizard:

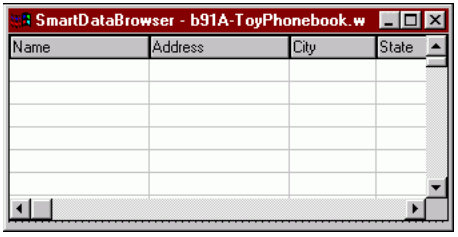


8 ♦ Choose Next to advance to the Wizard's final page. If you have successfully defined a data source and selected the fields to display, you will see "Congratulations!". If you do not, choose the Back button and make any necessary changes:




9 ♦ Choose Finish to dismiss the Wizard and reveal the new SmartDataBrowser master, if it was not already visible.

10 ♦ Choose File→ Save and give the new master a descriptive filename. Note that SmartDataBrowser filenames conventionally begin with b:



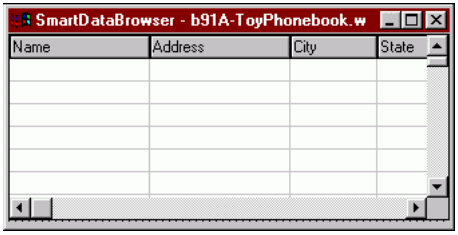

The new master object is now available for your use.

## 5.4.2 Configuring SmartDataBrowser Properties

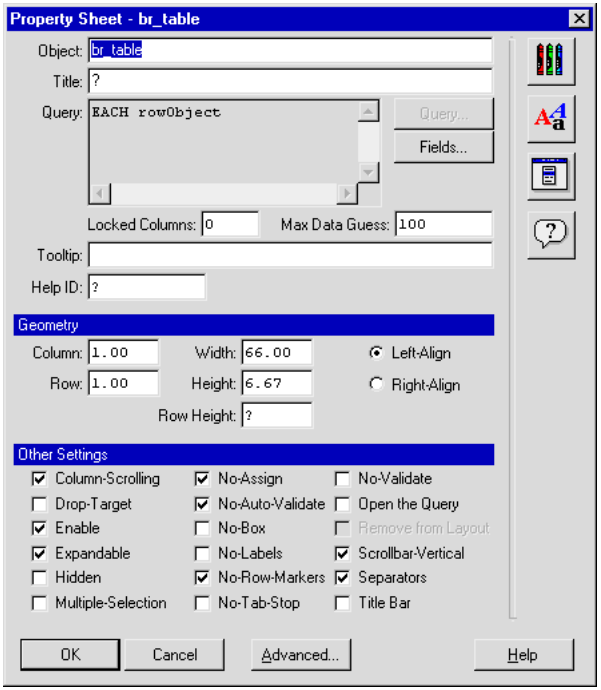
SmartDataBrowsers have properties associated both with their Browse component and with their nature as a procedure-based object. You can make a number of changes to those settings, if the default values do not meet your needs.

To open the property sheet associated with the Browse component, follow these steps:

1 ♦ Choose File→ Open, select the master you wish to configure, and choose OK. The SmartDataBrowser object appears:




2 ♦ Click on the object to select it, if necessary, and choose Tools→ Property Sheet. The Property Sheet dialog box opens:



### Minimal Configuration

Although you can accept many of the default settings, you might wish to make two changes to individualize the object. Note that these changes will apply to **all** created instances:

- Change the identifier to more closely reflect the object's nature.
- Add a title string, if you plan to allow the Browse component to display a title bar.

### Query and Fields

Since the SmartDataBrowser uses the data stream from a SmartDataObject, you can change only the fields and their ordering. The Query button is disabled because the query belongs to the SmartDataObject, not the Browse widget:

- Choosing the Fields button starts Column Editor and allows you to modify the columns this SmartDataBrowser will display. For information about how to use Column Editor, see the "Selecting Database Fields for Browsing" section in Chapter 4, "Data-access Objects."
- By default, all columns can be scrolled horizontally, if there are too many columns to display at once. You can choose to lock the leftmost columns in place, so that scrolling does not affect them. Enter in the Locked Columns box the number of columns you want to freeze in place.

**CAUTION:** If you lock more columns than will fit in the Browse's viewable width, you effectively disable the horizontal scroll bar, because all scrolling will take place out of sight to the right.

- When displaying a vertical scroll bar, the system tries to size the scroll bar's thumb slider to reflect the total number of lines in the data stream. The larger the number of lines, the smaller the thumb. However, that thumb size is based on the number of records being batched by the SmartDataObject, not on the total number in the stream. In general, the total number of records in the stream will be larger. When that is the case, the system cannot correctly size the thumb.

  If you can predict, even approximately, the total number of records in the data stream, you can improve the accuracy of the scroll bar's appearance. Enter your estimate in the Max Data Guess field.

- Type into the Tooltip field the identifying text that you want the system to display whenever the mouse cursor hovers over this SmartDataBrowser.

### Geometry

You can set the exact origin and size of any SmartDataBrowser instance created from this master. Set the values, in character units, in the appropriate fields. You can choose whether the origin is measured at the upper-left corner (the default) or the upper-right corner.

If you prefer to use pixels rather than character units, choose the Advanced button and set the values in that dialog box instead.

### Miscellaneous Properties (Other Settings)

You can make a number of changes to the appearance and behavior of the Browse component of the SmartDataObject master, if the default settings do not meet your needs:

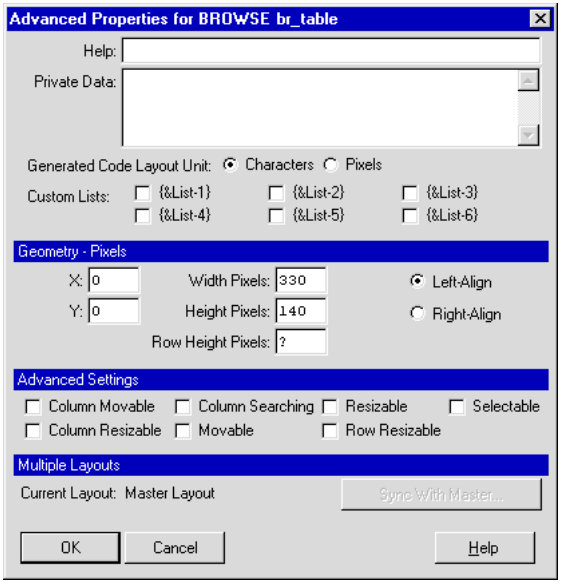
- **Column Scrolling** — Normally set. Clearing this check box causes the horizontal scrolling granularity to be one character. The default is one field (column). This option applies only when running under MS-Windows.
- **Drop Target** — Normally cleared. Setting this check box causes the system to generate an event when the user drags and drops another object onto this one. You must write code to deal with that event.
- **Enable** — Normally set. Clearing this check box causes the entire widget to be insensitive.
- **Expandable** — Normally set. Clearing this check box prevents the last column from stretching to cover any background exposed by scrolling. When Column Scrolling is set, and the Browse window is scrolled all the way to the right, some background will generally be exposed because it is unlikely that the last *n* columns will exactly fit the Browse width. When this option is set, the last column is expanded cosmetically to the right to mask any exposed background from view. This option applies only when running under MS-Windows.
- **Hidden** — Normally cleared. Setting this check box prevents implicit display of the object.
- **Multiple Selection** — Normally cleared. Setting this check box allows the user to select multiple rows. If any of the fields are updatable, you must also clear the No Row Markers check box.

  **NOTE:** Multiple Selection is not supported by the ADM. If you wish to use this option, you will have to write code to manage and communicate the multiple values as a special case.

- **No Assign** — Normally set. When cleared, any changes to updatable fields are automatically written back to the database when the user moves to a different row (record).
- **No Auto-Validate** — Normally set. Clearing this box causes the interpreter to automatically run validation checks.
- **No Box** — Normally cleared. Setting this check box causes the Browse window to display without a border. Under MS-Windows, this border is a single-pixel line.
- **No Labels** — Normally cleared. Setting this check box prevents display of the column labels.
- **No Tab Stop** — Normally cleared. Setting this check box removes the SmartDataBrowser from the applicable **TAB**-traversal list. Under MS-Windows, removing the object from the traversal list has some effects that might confuse the user.
- **No Validate** — Normally cleared. Setting this check box disables all validation conditions specified in the Data Dictionary for fields in this SmartDataBrowser.
- **Open the Query** — Normally cleared. Setting this check box activates the query and populates the fields during initialization. In a SmartDataBrowser, this option has no effect.
- **Scroll bar Vertical** — Normally set. Clearing this option hides the vertical scroll bar. The user can still scroll using the arrow keys. If you use a SmartPanel for navigation with this SmartDataBrowser, you should probably clear this box to avoid confusion.
- **Separators** — Normally set. Clearing this check box causes the widget to paint itself without lines separating the rows and columns. When this box is set, the Colors selector (see the "Colors Selector and Editor" section in Appendix A, "Frequently Used Dialogs") offers you the option of selecting a special color for the separator lines.
- **Title Bar** — Normally cleared. Setting this check box causes the Browse widget to display a full-width Frame-like title bar. Specify the title string if you set this option.

### Advanced Properties

Choose the Advanced button. The Advanced Properties dialog box will open:




You can make a number of changes to the SmartDataBrowser's appearance and behavior by changing the default settings in this dialog box:

- **Help** — Text in this field displays in the status bar of the parent window when the SmartDataBrowser has the focus. If the parent window has no status area, this text will not display.
- **Private Data** — AppBuilder writes out the contents of this field as the value of the object's PRIVATE-DATA attribute. You can write code to read and operate on that value in any way that meets your needs.
- **Generated Code Layout Unit** — Controls whether AppBuilder describes the position and size of this SmartDataBrowser in character or pixel units when it generates source code.
- **Geometry** — Expresses the origin and size of this widget in pixels. You can set a particular origin and size for the object here rather than interactively, if you wish. You can also choose to have the origin measured at the upper-right rather than the upper-left corner.
- **Column Movable** — Normally cleared. Setting this check box allows the user to reorder the columns by dragging.

- **Column Resizable** — Normally cleared. Setting this check box allows the user to resize the columns interactively. Resizing the column has no effect on the actual field widths.
- **Column Searching** — Normally cleared. Setting this box in read-only Browsers allows the user to search in an individual field across all records. Updatable Browsers allow such searching by default.

  The search is started by clicking the desired column title and typing the letter for which to search. No prompt appears for the letter. The Browser scrolls its window to bring into view and make current the next record, if any, in which the search field data begins with the typed-in letter.
- **Movable** — Normally cleared. Setting this check box and the Selectable check box allows the user to move the entire SmartDataBrowser object around within the bounds of the enclosing Frame.
- **Resizable** — Normally cleared. Setting this check box and the Selectable check box allows the user to change the size of the SmartDataBrowser object, within the limits of the enclosing Frame.
- **Row Resizable** — Normally cleared. Setting this check box allows the user to change the vertical size (height) of the rows by dragging a horizontal separator line.
- **Selectable** — Normally cleared. Setting this check box allows the user to select the SmartDataBrowser object.

#### Procedure Settings

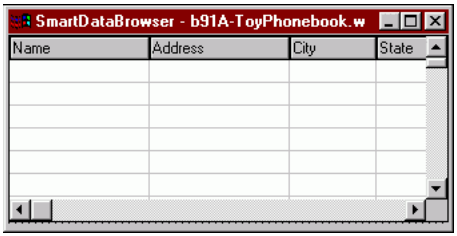
See the "Procedure Settings" section in Appendix A, "Frequently Used Dialogs."

### 5.4.3 Editing a SmartDataBrowser Master

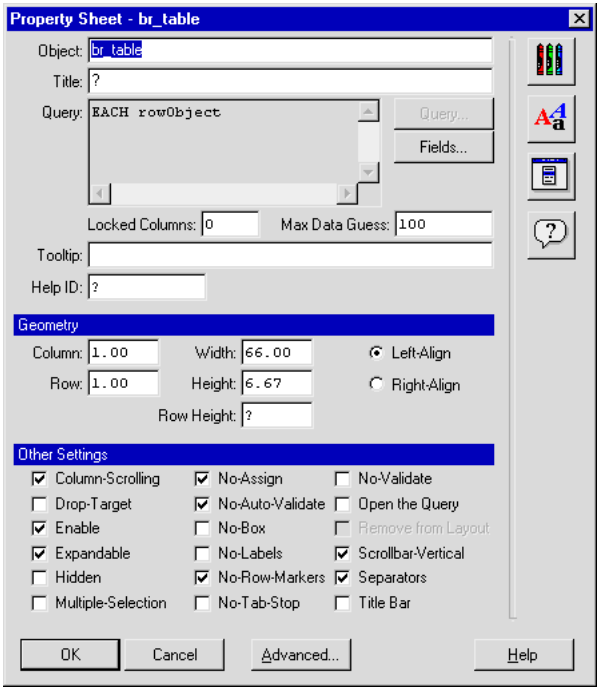
Any modular system must have and enforce strict data-exchange requirements if it is to avoid possible data corruption and loss. You can always update the field selections in your SmartDataBrowser to reflect your changing needs, though you might also need to make corresponding changes to the associated SmartDataObject (see Table 5–1).

When you are ready to edit your SmartDataBrowser master, follow these steps:

1 ♦ Choose File→ Open and select the source file (for example: `bSampleSDB.w`). The SmartDataBrowser object appears, displaying its Browse widget in a layout workspace:




2 ♦ Click on the Browse widget to select it, and choose Tools→ Property Sheet. The Property Sheet dialog box opens:




3 ♦ Choose the Fields button to open the Column Editor dialog box (note that the Query button is not available). See the "Selecting Database Fields for Browsing" section in Chapter 4, "Data-access Objects," for instructions on how to use the Column Editor.

4 ♦ Make your changes to the fields list. If you wish to change other master properties, see the "Configuring SmartDataBrowser Properties" section in this chapter.

5 ♦ Save your changes.

### 5.4.4 Creating and Placing a SmartDataBrowser Instance

Once you have a SmartDataBrowser master designed that will display the data you need, you can reuse it indefinitely in your applications.

To add a SmartDataBrowser to your organizer workspace, follow these steps:

1 ♦ Click on the SmartDataBrowser tool icon in the Object Palette. When the Choose dialog box opens, select the appropriate master and click OK.

2 ♦ Position the cursor over the workspace at the point where you want the upper-left corner of the widget to appear. Click, and the new SmartDataBrowser object appears.

3 ♦ AppBuilder immediately opens an Advisor window if it finds opportunities to create SmartLinks to other objects. Accept the Advisor's recommendations, if they seem appropriate.

4 ♦ Drag the edges of the new SmartDataBrowser object to position and resize it.

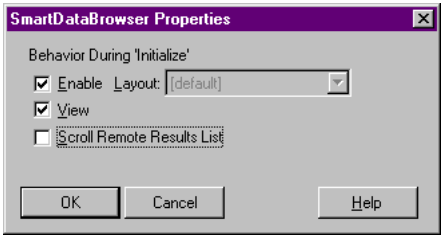
5 ♦ Configure the object's instance properties. See the section.

6 ♦ Save your work.

### 5.4.5 Configuring a SmartDataBrowser Instance

There are several properties you can configure in a SmartDataBrowser instance, if the default values do not meet your needs. To do that, follow these steps:

1 ♦ Click the menu button and choose Instance Properties from the menu that appears. That dialog box opens:



2 ♦ To disable the object at startup, clear the Enable box.

3 ♦ If multiple layouts are defined, choose one from the list.

4 ♦ To hide the object at startup, clear the View box.

5 ♦ To enable scrolling to the last record in the data stream, set the Scroll Remote Results List checkbox. You might also need to write some code. For details, see the "Scrolling to the Last Record in the Data Stream" section.

6 ♦ Choose OK and save your work.

### 5.4.6 Scrolling to the Last Record in the Data Stream

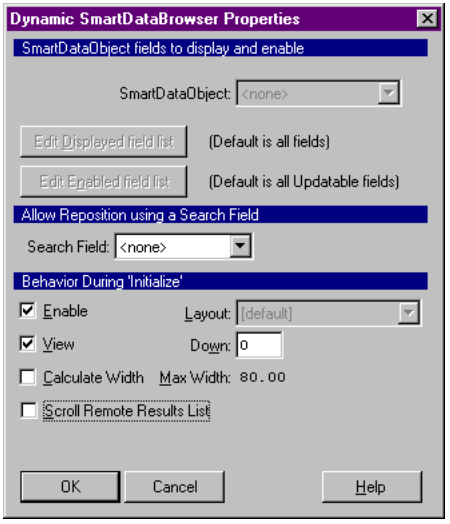
Conventionally, the size and position of the thumb slider within a scrollbar provides the user a hint about the overall size of the document being examined, and the position of the viewport within that document. The ratio of thumb size to scrollbar size indicates how much of the whole document is currently visible. The position of the thumb within the scrollbar's length represents how far the visible portion is from the beginning and end of the document.

In the case of the SmartDataBrowser and its basic counterpart, the document being examined is actually a data stream. Typically, that data stream will contain many records. Many more records, in general, than are practical to buffer. But if there are more records than are buffered, the size and position of the thumb will reflect the buffer, not the whole data stream. Moving the thumb to the bottom of the scrollbar track brings into view the last record in the buffer, not the last record in the data stream.

This can be confusing to the user of your application.

Follow these steps to fix the problem in your SmartDataBrowser:

1 ♦ Click the SmartDataBrowser's menu button and select Instance Properties. That dialog box opens:

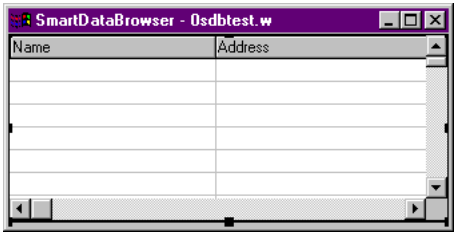



2 ♦ Set the Scroll Remote Results List check box.

3 ♦ Save your work.

If you have a trigger defined for the ROW-DISPLAY event, you must modify it. Follow these steps.

1 ♦ Click the menu button and choose Edit Master. The master workspace opens (its appearance may be different):




2 ♦ Select the workspace if necessary and open a Section Editor window. Create a new procedure, for example OldRowDisplayHandler, and move the code from your ROW-DISPLAY handler into the new procedure.

3 ♦ Create a new function as an override for rowDisplay. Put in it a call to the procedure you just created, followed by a RUN SUPER statement. If you name your procedure OldRowDisplayHandler, your function body would look like this:

```
RUN OldRowDisplayHandler . /* run original local code */
RUN SUPER .                /* run the handler in the super procedure*/
```

4 ♦ Open your local ROW-DISPLAY trigger in the Section Editor. It should now have no code in it, because you moved that code to the new procedure you created.

5 ♦ Add this line as the new body of your local trigger:

```
RUN rowDisplay. /* run local override of super-procedure handler*/
```

6 ♦ Save your work.

Now, when the user drags the thumb to the bottom of the scroll bar, bringing the last record in the buffer into view, the Browser requests a new batch of records from the Data-Source. When the new batch is received, the Browser repositions its thumb to account for the new total record count. Thus, the first time a new batch is requested, the thumb will in general reposition to the middle of the scrollbar because the former last record is now in the middle of the buffer. The second read will create a total of three batches, with the thumb positioned at the two-thirds point, and so forth.

## 5.5 SmartDataViewers

While the SmartDataBrowser is useful for rapidly scanning many records, it is less easy to focus on the fields of a single record. This is particularly true when there are a large number of fields involved. The SmartDataBrowser can only display data in tabular format; no more subtle arrangement is possible.

The SmartDataViewer displays fields from a single record at a time. By default, each field is represented by a Fill–in widget, and the fields are stacked in a single left-aligned column. In contrast to the SmartDataBrowser, you can completely rearrange the fields of a SmartDataViewer master to suit your layout, and can even replace individual Fill–ins with special-purpose SmartDataFields that you create.

The SmartDataViewer instances you use in your applications are each based on a master object that you design. SmartDataViewer masters are static objects; you design each one to display records supplied by a particular SmartDataObject. Once you design a SmartDataViewer master, you can reuse it repeatedly in your applications.

### 5.5.1 Creating a SmartDataViewer Master

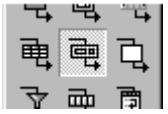
The first step in defining a new SmartDataViewer master is always to identify the source of the data stream. Because of this dependency, you must first create your SmartDataObject, then create the SmartDataViewer. See the "Creating a SmartDataObject Master" section in Chapter 4, "Data-access Objects," for information about that process.

AppBuilder supplies a Wizard to help with creating SmartDataViewer masters. The Wizard has four pages, only two of which involve significant work:

1. Introduction
2. Identifying the data source
3. Selecting the fields to display from the data stream
4. Congratulations

To create a SmartDataViewer master, follow these steps:

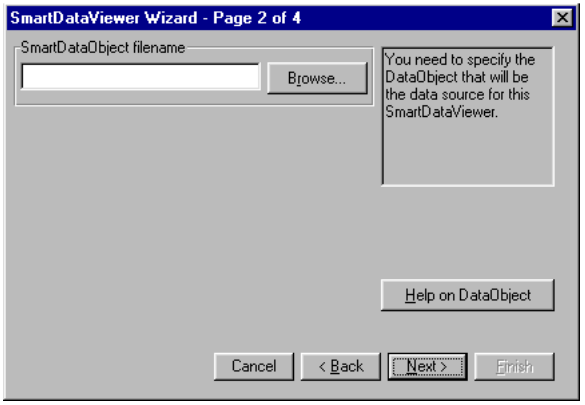
1 ♦ Start the SmartDataViewer Wizard by clicking on the SmartDataViewer tool icon in the Object Palette:



When the Choose dialog box opens, choose New.

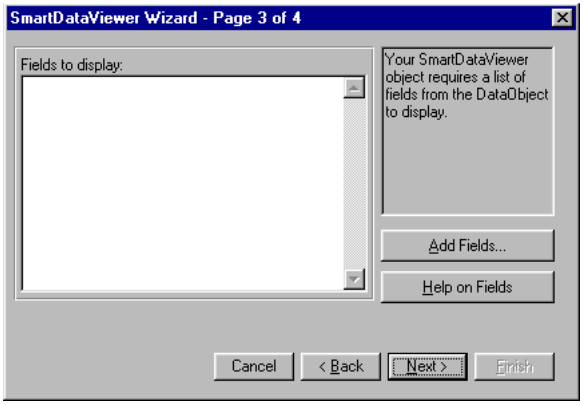
**NOTE:** You could also start the Wizard by choosing File→ New and selecting the SmartDataViewer item from the list, or by right-clicking the icon in the Palette and selecting New SmartDataViewer from the menu that appears.

2 ♦ The SmartDataViewer Wizard starts up and displays its first page. When you have finished reading the introductory text, choose Next to move to Page 2:

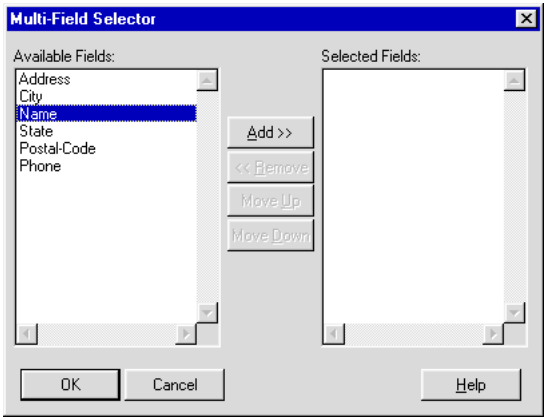



3 ♦ Choose the Browse button and select the appropriate SmartDataObject master. Its filename appears in the fill–in.

4 ♦ Choose Next to move to Page 3:

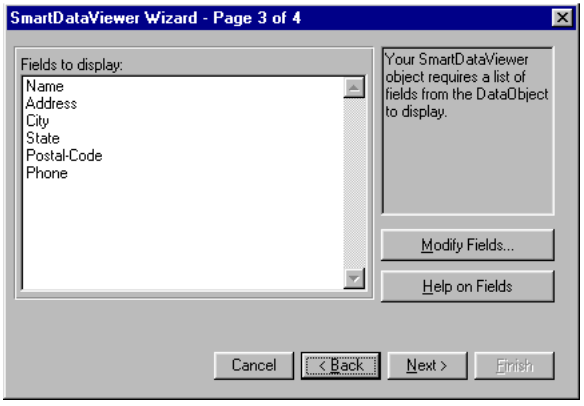


5 ♦ Choose the Add Fields button to open a Multi-Field Selector dialog box:



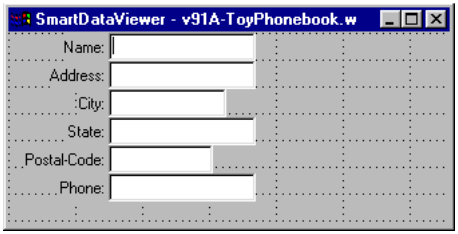

6 ♦ The fields Available for display appear in the left-hand list. Select those you wish the SmartDataViewer to display and click the Add button to move them to the Selected list in the right-hand window. If you make a mistake, click the Remove button.

When you have finished selecting fields, you can reorder them using the Move Up and Move Down buttons. When they are in the order you want them displayed, choose OK. The Selector dialog box closes and the list of selected fields appears in the Wizard:




7 ♦ Choose Next to advance to the Wizard's final page. If you have successfully defined a data source and selected the fields to display, you will see "Congratulations!". If you do not see that page, choose the Back button and make necessary changes.

8 ♦ Choose Finish to dismiss the Wizard and reveal the new SmartDataViewer master, if it was not already visible:




9 ♦ Fields in a SmartDataViewer are read-only unless you connect them to the appropriate TableIO source, such as an Update SmartPanel. If you do connect them to Update controls, then all fields become updatable unless you explicitly make them read-only.

To make any of the individual fields read-only, double-click on it to open its property sheet. When the Property Sheet dialog box opens, set the Read-only check box and choose OK.

Note the difference in how fields display at run time, depending on their state:

In the first case, there is no Update SmartPanel, and so all fields are read-only. The Fill–ins display their content without displaying themselves—the fields are not outlined or modelled in any way; they are invisible. All text looks static. This is the same effect you will get if you set the fields to be not-updatable in the SmartDataObject:



In the second case, there is an Update SmartPanel, and so all fields are updatable if they are updatable in the SmartDataObject. The Fill–ins display their content in the ordinary way:




In the third case, there is an Update SmartPanel, but the Name field has its Read-only property set. Here, the Name field is clearly in a different state to the others, and the experienced user will recognize that it is not editable. However, this style of presentation only applies where the field is updatable in the SmartDataObject, but set read-only in the Fill–in. If it is not updatable in the SmartDataObject, it appears as in the first example:




10 ♦ Choose File→ Save and give the new master a descriptive filename. Note that SmartDataViewer filenames conventionally begin with v. The new master object is now available for your use.

If you plan to replace one or more of the simple fields with SmartDataFields, do it now. See the "Placing and Configuring a SmartDataField Instance" section in this chapter for information about that process.

**NOTE:** Although SmartDataViewers can be members of the class SmartContainer, they are not organizer objects, so placing SmartObjects in them other than SmartDataFields is not supported. The only exception is that, if you replace one or more of the fields with SmartSelect objects, you can also include an equal number of SmartDataObjects to feed them.

## 5.5.2 Adding and Removing SmartDataViewer Fields

If you modify the data stream being supplied to a SmartDataViewer master, you might have to add and/or remove fields.

### Removing SmartDataViewer Fields

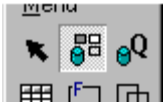
Removing fields is very easy: just select and delete them. AppBuilder handles the situation gracefully, without leaving residue behind.

### Adding SmartDataViewer Fields

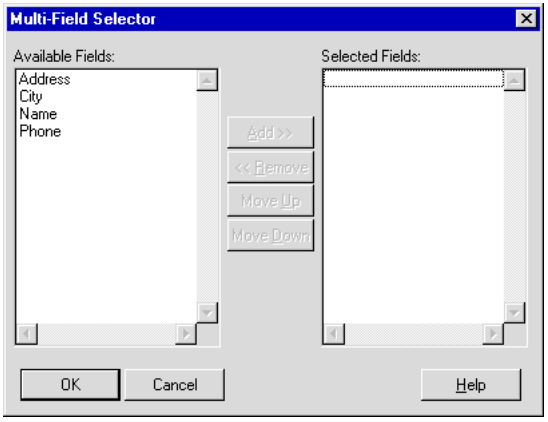
Adding additional fields for display is also easy, once you know how. Follow these steps:

1 ♦ Choose File→ Open and open the master object. Drag the edges of its workspace, if necessary, to add space for the new fields.

2 ♦ Click on the DB–Fields tool icon in the Objects Palette:



3 ♦ Click on an empty spot in the master object's workspace, not the main workspace. The Multi-Field Selector dialog box opens:




4 ♦ Move the desired fields from the Available list to the Selected list. Choose OK.

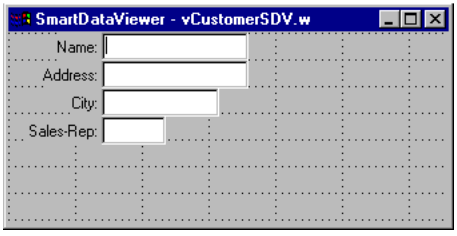
5 ♦ Position, size, and configure the newly added fields in the master workspace. Save your work.

### 5.5.3 Configuring SmartDataViewer Properties

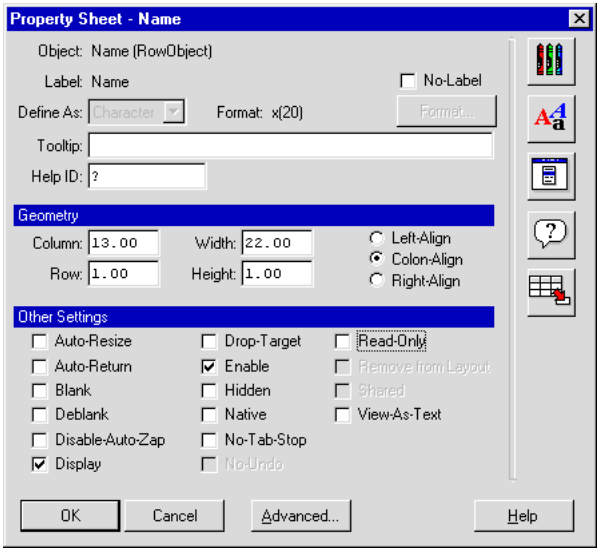
SmartDataViewers have properties associated both with their Fill–in components and with their nature as a procedure-based object. You can make a number of changes to those settings, if the default values do not meet your needs.

To open the property sheet associated with a Fill–in component, follow these steps:

1 ♦ Choose File→ Open, select the master you wish to configure, and choose OK. The SmartDataViewer master object appears in its own workspace:




2 ♦ Click on the Fill–in to select it, and choose Tools→ Property Sheet. The Property Sheet dialog box opens:



### Minimal Configuration

Although you can accept many of the default settings, you might wish to make three changes:

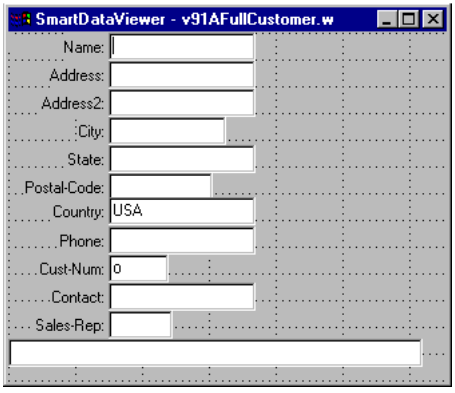
- Add a Tooltip string. Under MS-Windows, Tooltips display whenever the user allows the mouse cursor to hover for a few seconds over an object for which a Tooltip is defined.
- Add a unique integer as the identifier for context-sensitive help, if you plan to provide such help as part of your application.
- Set the Read-Only check box (in the Other Settings area) to prevent the user from modifying the data in the field.

### Rearranging the Layout

The way a Viewer's fields are laid out is also a property that can be changed, though only at the level of the master. All instances created from a given master share the same physical layout. To have different layouts for different applications, you must either create extra copies of the master and lay out each one differently—effectively creating separate masters—or use the alternate-layouts feature. See Appendix B, "Multiple Layouts" for information about that feature.

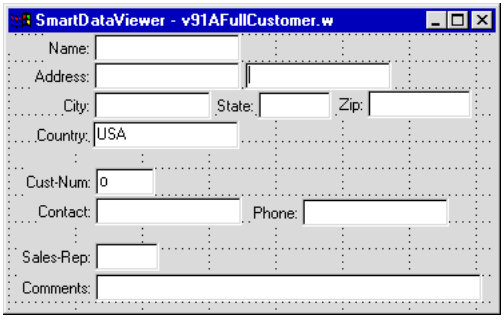
To create a custom layout, follow these steps:

1 ♦ Open the master object by choosing File→ Open. The object opens in a design window workspace:




2 ♦ Save the master under the name you will use for the new arrangement. Do this before you actually make any changes, so that you will not risk getting confused later and overwriting the original arrangement.

3 ♦ Resize the workspace appropriately, drag the individual Fill–ins into the new arrangement, and save again:




Note that you should make the workspace no larger than needed, since the amount of space taken up by an instance is always the same as the master's workspace.

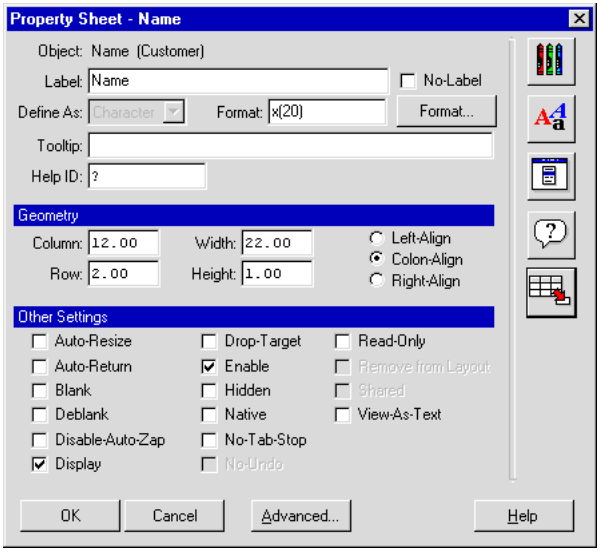
### Other Property Changes

By default, AppBuilder predefines certain properties of each Fill–in that is used by a SmartDataViewer, and makes some of the properties read-only:

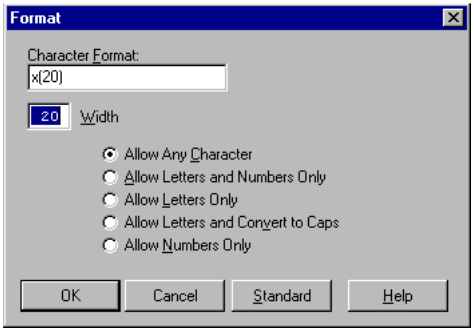
- Object identifier
- Label
- Allowed field format and size
- Help string

AppBuilder creates and assigns a unique identifier, and sets the other properties to conform to the field definition from the data stream. You cannot assign a different identifier or change the basic data type, but should you wish to make changes to the label, field specifications, or help string, you can do so by following these steps:

1 ♦ Click the Database Field button, and when the dialog box opens, clear the check boxes for the properties you wish to set by hand. Choose OK to dismiss the dialog box. If you clear all three, the properties dialog box appears as shown here:




2 ♦ You can now enter new values for the label and the format specification. If you wish to pick the format specifier from a list, choose the Format button to open that dialog box:



You can more directly change other properties:

- **No Label** — Normally cleared. Setting this check box turns off display of any label. Setting this check box does not clear the actual label text; the label string remains visible in the property sheet. The only other effect of setting or clearing this box is to alter the values in the Geometry section so that the Fill–in stays at the same XY location within the enclosing Frame.

- **Geometry** — Reflects the XY origin and size, in character units, of the Fill–in. By default, the Fill–in is colon-aligned. You can change that to left- or right-alignment, though the only noticeable effect that will have is to change the X origin value.

- **Auto-Resize** — Normally cleared. Setting this box causes the object to automatically change its displayed size to agree with the current type size.

- **Auto-Return** — Normally cleared. Setting this box causes focus to move to the next object in the traversal list once this Fill–in has accepted as many characters as it can.

- **Blank** — Normally cleared. Setting this box prevents the Fill–in from echoing input back to the display. Useful for password fields and similar applications.

- **Deblank** — Normally cleared. Setting this box causes the Fill–in to automatically discard any leading blanks from input.

- **Disable Auto-Zap** — Normally cleared. Setting this box prevents the Fill–in from automatically clearing its input field whenever it gets the focus.

- **Display** — Normally set. Clearing this box prevents the SmartDataViewer from automatically populating this Fill–in during initialization.

- **Drop Target** — Normally cleared. Setting this box causes this object to experience an event whenever the user drags another object onto this one. You must write the appropriate event-handling code.

- **Enable** — Normally set. Clearing this box makes the Fill–in decline input focus when offered.

- **Hidden** — Normally cleared. Setting this box prevents the Fill–in from responding to implicit requests to display itself. It will only honor explicit requests.

- **Native** — Normally cleared. Setting this box causes the Fill–in to vary its behavior according to the underlying platform (MS-Windows, for example) rather than behaving in the same way regardless of platform.

- **No Tab Stop** — Normally cleared. Setting this box removes this Fill–in from the enclosing Frame's traversal list. Normally, pressing the key causes focus to move to the next object in the list. When this box is set, the Fill–in can neither lose nor gain focus when the user presses the **TAB** key. If the widget has the focus, it will ignore the key, and if it does not, focus will cycle through the other members of the traversal list while ignoring this widget.

**CAUTION:** Making this Fill–in Movable or Resizable (Advanced Properties) prevents this Fill–in from gaining input focus using the mouse. So if you also set this property, you effectively prevent the widget from ever gaining input focus.

- **No Undo** — Cleared. Cannot be set. Progress will always journal changes to the data in this Fill–in.

- **Read-Only** — Normally cleared. Setting this box prevents the user from changing the content of the Fill–in.

- **Remove from Layout** — Cleared. Cannot be set unless defining an alternate layout.

- **Shared** — Cleared. Cannot be set. This Fill–in will always be local to the SmartDataViewer within which it is defined.

- **View as Text** — Normally cleared. Setting this box causes this Fill–in to display the contents of its field as though it were static text: read-only, and without any border or 3D effect.

### Advanced Properties

A Fill–in has only the minimum Advanced Properties, but you can change those that do not meet your needs.

- **Initial Value** — You cannot set this for a Fill–in that is part of a SmartDataViewer.

- **Help** — Unless this Fill–in inherits the help string defined in the data dictionary, the text you enter here will display in the parent Window's status bar whenever this Fill–in has input focus. No text will display if the Window has no status bar.

- **Private Data** — AppBuilder assigns the contents of this field as the value of the PRIVATE–DATA property. You can write code to manipulate this data in any way you desire.

- **Layout Units** — You can choose character or pixels. This setting affects source code generation only.

- **Custom Lists** — You can add this Fill–in to any or all of the six lists (macros) maintained by the enclosing Frame.
- **Geometry** — Shows the same information as in the base property sheet, but in pixel units.
- **Manual Highlight** — Normally cleared. Setting this box allows you to write code to define a custom highlight effect. Your custom effect will be applied whenever this Fill–in has input focus.
- **Movable** — Normally cleared. Setting this box makes it impossible for the user to give this Fill–in input focus by clicking with the mouse. Instead, the user can use the mouse to drag the Fill–in field—but not the label, which remains where it was—to a different position within the bounds of the enclosing Frame.
- **Resizable** — Normally cleared. Setting this box makes it impossible for the user to give this Fill–in input focus by clicking with the mouse. Instead, if the Selectable box is also set, clicking with the mouse causes handles to appear so that the user can resize the object.
- **Selectable** — Normally cleared. Setting this box allows the user to select this object.

**CAUTION:** If you make this Fill–in Movable or Resizable, and you also set the No Tab Stop option in the base property sheet, you effectively prevent this object from ever gaining input focus.

## 5.6 SmartDataFields

A simple Fill–in is not necessarily the best representation for all data fields. Sometimes it would be very convenient to bring the power of ADM Smart technology down to the level of the individual data field in a SmartDataViewer. AppBuilder offers the SmartDataField object as a way for you to do that.

The SmartDataField instances you use in your applications are each based on a master object that you design for a particular purpose. Once you design a SmartDataField master, you can reuse it repeatedly whenever you need a SmartDataViewer field with that particular capability.

AppBuilder supports SmartDataFields only as replacement for simple SmartDataViewer fields, not for standalone placement. You can extend their applicability, if you wish, by writing the necessary supporting code.

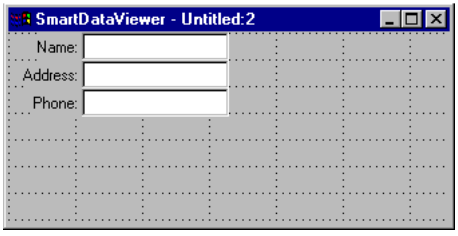
For information about creating SmartDataField masters, see the *Progress ADM 2 Guide*.

### 5.6.1 Placing and Configuring a SmartDataField Instance

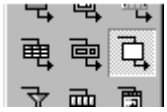
Each field in a SmartDataViewer represents a certain data type—an INTEGER, for example, or a DATE. The SmartDataField object you use as a replacement for such a field must represent the same type.

To use a SmartDataField in a SmartDataViewer, follow these steps:

1 ♦ Choose File→ Open and select the SmartDataViewer master you intend to modify. Click OK. The master opens in a workspace window, as illustrated here:



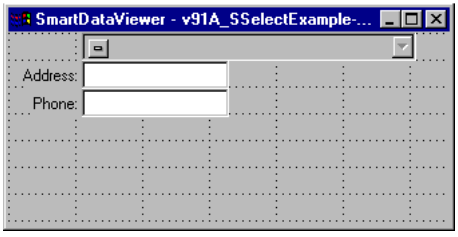

2 ♦ Click on the SmartDataField tool icon in the Object Palette:



When the Choose dialog box opens, select the appropriate master and click OK.

Alternatively, right-click on the SmartDataField tool icon in the Object Palette, then select an object from the menu that opens.

3 ♦ Position your mouse cursor over the field you are replacing, and click to make the replacement. The SmartDataField representation appears in place of the original Fill–in:



4 ♦ Add a static text label, if appropriate. A SmartDataField does not inherit the label of the field it replaces.

5 ♦ AppBuilder supplies stubs for the enableField() and disableField() procedures. Open the Section Editor and add additional code to make those procedures work for the particular SmartDataField you added.

## 5.7 SmartSelects

The SmartSelect is a predefined SmartDataField object that represents a self-populating set of choices. You would typically use a SmartSelect to choose a new value for a field from among a relatively small group of possibilities drawn from a related table.

The SmartSelect object is *data-driven*. Most widgets used in an application have their values embedded as data within the program itself, so that changing the values in any way typically requires recompilation. Since the SmartSelect obtains its values from the database being modified, instead, recompilation is rarely required.

Some examples of the kind of situation for which a SmartSelect would be appropriate:

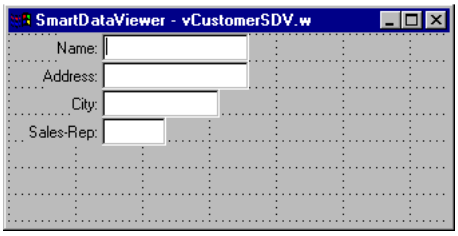
- Assigning a different sales rep to an account. Typically a company has a small number of sales reps, each of whom calls on many accounts.
- Changing a customer's credit limit, if the possible choices are few. For example: discredited; credit hold; credit not requested; $10,000; $50,000; $100,000; unlimited.
- Changing some demographic value in a personnel record. For example: single, partnered, married, separated, divorced, widowed, deceased, dependent.
- Setting the circulation status of a book in a library-management system. For example: on-order, reference, reserved, available, checked-out, bindery, lost.
- Picking a customer number for use in order-entry. Since customer numbers are important to the seller rather than the customer, customers rarely remember them.

You can configure each SmartSelect instance to represent its set of values using your choice of four different widgets: Combo Box (the default), Radio Set, Selection List, or a Browse that opens in its own popup window. Of those four, only the Browse is capable of handling a large or unpredictable number of values. You might find it helpful to experiment before deciding on a representation.

### 5.7.1 Placing a SmartSelect Instance

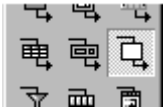
To use a SmartSelect as a field in a SmartDataViewer, follow these steps:

1 ♦ Choose File→ Open and select the SmartDataViewer master you intend to modify. Click OK. The master opens in a workspace window, as illustrated here:



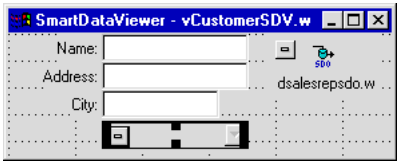

2 ♦ Create, if necessary, and place the SmartDataObject that will supply values to the SmartSelect.

3 ♦ Right-click on the SmartDataField tool icon in the Object Palette:



Choose SmartSelect from the menu that opens.

4 ♦ Position your mouse cursor over the field you are replacing, and click to make the replacement. The SmartSelect representation appears in place of the original Fill–in:



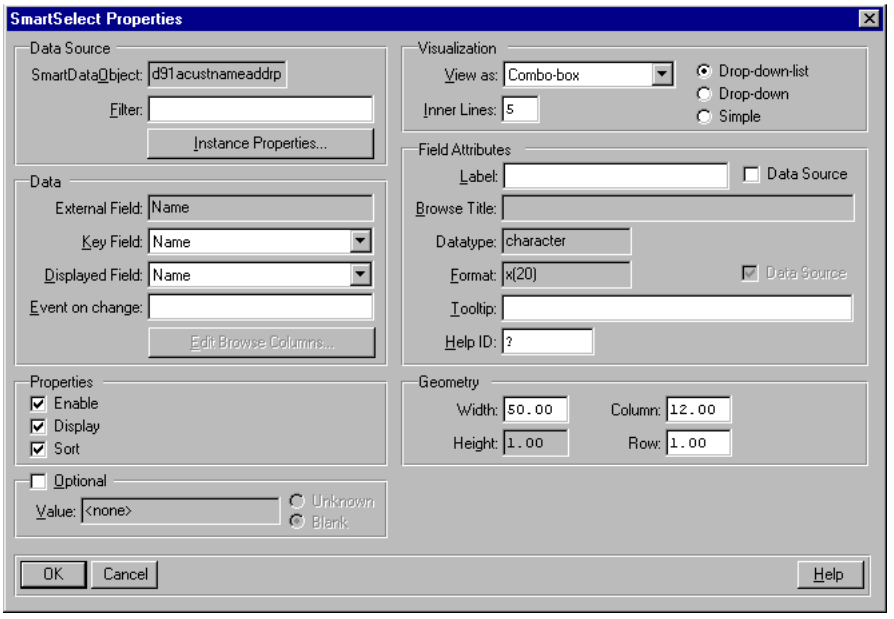

5 ♦ An Advisor should now appear and offer to add the Data link from the SmartDataObject to the SmartSelect. Agree to the offer.

6 ♦ If no Advisor appears, open the SmartLink editor and add it yourself.

7 ♦ Click on the SmartSelect's menu button and choose Instance Properties from the menu. Configure the instance. See the "Configuring a SmartSelect Instance" section in this chapter for information.

## 5.7.2 Configuring a SmartSelect Instance

The SmartSelect has a relatively complex Instance Properties dialog box:




### Data Source

This section identifies the SmartDataObject that will feed the SmartSelect object. You cannot enter the filename of the SmartDataObject directly; AppBuilder fills in this field as part of creating the SmartLinks, once you place the SmartDataObject in the Viewer workspace.

You can subset the data that the SmartSelect displays. Type a subsetting expression—for example:

**Credit–Limit > 10000**

into the Filter field. The SmartSelect will then display only the subset of items that conform to that restriction.

### Data

This section describes the field being replaced, and the identifying characteristics of its replacement:

- **External Field** — Identifies the field in the SmartDataViewer that you are replacing. This identification is read-only, supplied for your information only.

- **Key Field** — If the replacement data stream has more than one field, use this combo box to select the field that will appear in place of the original. Be certain to select the correct one.

  For example, the State table in the sample database has three fields: State, State-name, and Region. If you use a SmartSelect fed from the State table to update the Customer.State field, the only correct choice for the key field would be State.State, since that is the related field. If you choose State-name or Region instead, confusion is almost certain to result, later.

- **Displayed Field** — If the replacement data stream has more than one field, use this combo box to select the field that the user will see. To continue the State example, you might prefer to display the more recognizable State-name field. That would be a more user-friendly choice than displaying the cryptic two-character post-office abbreviation from the State field itself, even though it is that two-letter code that you will store.

- **Event on change** — You can declare a name for the event to be experienced when the value of this field changes. This allows you to write an appropriate event-handling procedure. The name you define is local to the object. See the *Progress ADM 2 Guide* and the for further information.

- **Edit Browser Columns** — Active only when the field is to be represented as a Browse widget. Choosing this button opens the standard field-selection dialog box so that you can select the fields to be displayed.

### Properties

This section presents three important behavioral properties:

- **Enable** — Normally set. Clearing this box prevents the SmartSelect from enabling itself even when the other fields of the SmartDataViewer enable themselves.

- **Display** — Normally set. Clearing this box prevents the SmartSelect from displaying data even when the other fields of the SmartDataViewer display data.

- **Sort** — Normally set. Clearing this box causes the items to appear in unsorted order. Normally the visible fields appear sorted on the Displayed Field.

### Optional

This check box is normally cleared. Setting this box means that the associated field need not have a value.

You can specify the text that represents that nonvalue; the default is “<none>”. Note that this text does not necessarily appear at the top of your list of choices. If you wish it to appear as the

first item in the list, you must either explicitly make it the first item in the source (for unsorted lists) or begin with some character, such as the angle bracket (<), that the sort operation will move to the top of the list.

### Visualization

This section allows you to specify how the SmartSelect list will represent itself at run time. The choices are:

- **Browser** — The Browser representation is the best choice for handling lists that are known to be very large or are of unknown size. This representation initially appears in a closed state, under an iconic pushbutton next to the current value of the field:

  Clicking the button opens the list of items as a Browse in a separate window. Click on the Edit Browser Columns button to choose what fields to display. Set the number of lines to display by entering an integer in the Inner Lines field.

- **Combo Box** — Default. This representation is configured as a Drop–down List by default. For a description of the three types of representation, see the "Combo Boxes" section in this chapter.

  Set the number of lines to make visible in the list by entering an integer in the Inner Lines field.

- **Selection List** — Selection lists generally take up more space than combo boxes because you must specify their size at design time. Note that setting the Inner Lines field property has no effect on the visible size of the widget. You must resize it manually by dragging the handles.

- **Radio Set** — Best suited to very small, fixed-size data sets that can be represented as short labels. You can specify a vertical (default) or horizontal orientation.

### Displayed Field Attributes

Specify the characteristics of the field you display as a prompt to the user:

- **Label** — Specify the label, if any, to display with the SmartSelect. The SmartSelect object does not inherit the label of the SmartDataViewer field it replaces. You can set the Data Source check box to use the schema name of the field, or type in a label of your choice.

- **Browse title** — Active only when Browse is the representation type. You can set the title string that the Browse widget will display.

- **Datatype** — Read-only. Displays the underlying data type of the displayed field from the schema definition.
- **Format** — Displays the field descriptor from the schema definition. Clear the Data Source check box if you have reason to modify this descriptor.
- **Tooltip** — Enter the string the SmartSelect will display as the MS-Windows Tooltip.
- **Help ID** — If you will offer context-sensitive help for this object, enter a unique integer identifier here.

#### Geometry

Displays the origin and size of the SmartSelect representation in character units. You can set these values here, or in pixel units in the Advanced Properties dialog box.

### 5.7.3 Example: Assigning a New Sales Rep

This example illustrates how to use a SmartSelect instance to update the Sales–Rep field in a Customer record. The tables it uses are found in the sports2000 database.

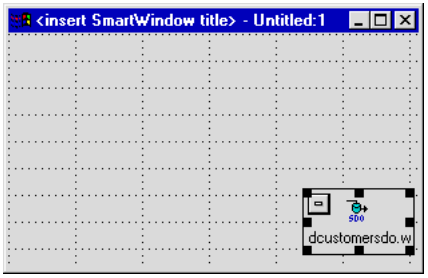
Follow these steps, agreeing to all Advisor suggestions for links:

1 ♦ Create a SmartWindow as the application workspace. Save it.

2 ♦ Create the SmartDataObject master that will pump the Customer records. Choose the Customer table. Use only the Name, Address, City, and Sales–Rep fields from the Customer table. Save the master:



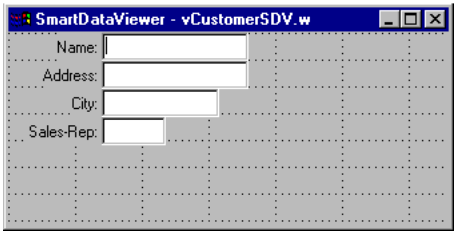
3 ♦ Place into the application workspace an instance of the Customer SmartDataObject you just created:



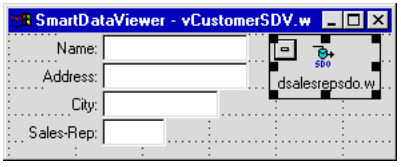

4 ♦ Create the SmartDataObject master that will feed the SmartSelect field. Choose the Salesrep table. Use only the Sales–Rep field from that table. Save the master:



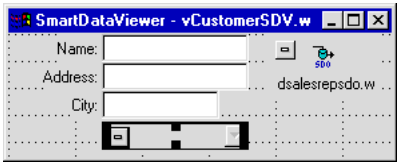

5 ♦ Create the SmartDataViewer master that will display the Customer records. Include all four of the fields. Save the master, but leave its workspace open:



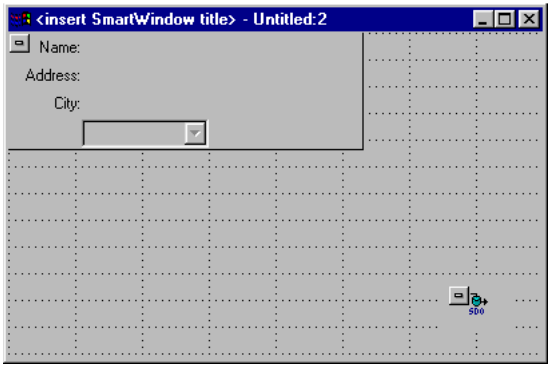

6 ♦ Place an instance of the Sales–Rep SmartDataObject into the master SmartDataViewer workspace:



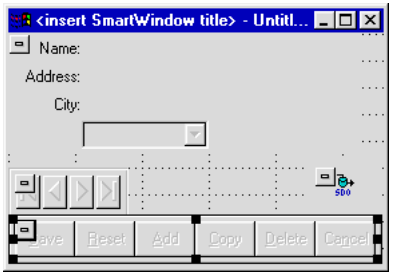
7 ♦ Replace the Sales–Rep field in the SmartDataViewer with a SmartSelect instance. Resize the instance for cosmetic reasons:



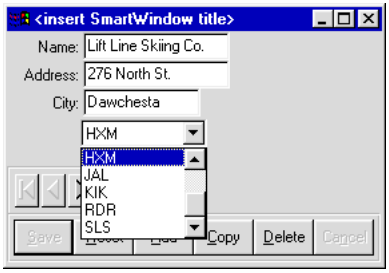

8 ♦ Save the changes in the SmartDataViewer. Place an instance of the SmartDataViewer into application workspace:




9 ♦ Place Navigation and Update SmartPanels into the application workspace:



10 ♦ Run the workspace. You should be able to assign a new sales rep to any account by choosing from the combo box:




## 5.8 Basic Data Browsers

In addition to the SmartDataBrowser, AppBuilder supplies the basic 4GL Data Browser (Browse widget), with its built-in query feature. Like the SmartDataBrowser, a basic Data Browser presents the results of a query in simple row/column tabular format. Each row displayed in a Browse always represents one record from the data stream. Each column represents one of the fields in the records. The Browse object can display any of the standard data types that can be represented by ASCII text.

Unlike a SmartDataBrowser, the basic Browse object manages its own data stream, communicating with the data source on its own behalf rather than tapping into the data stream supplied by a modular data pump such as the SmartDataObject.

Although you can freely place a basic Browse object in a Smart workspace, for example a SmartWindow, doing so will not enable it to take advantage of SmartLinks or participate in ADM paging.

### 5.8.1 Creating a Basic Data Browser Instance

Unlike a SmartDataBrowser, a basic Browse object has no master, and obtains its data stream directly using its built-in query feature. Although there is no master to define, you must define that query.

To create and place a basic Data Browser, follow these steps:

1 ♦ Click on the basic Browse tool icon in the Object Palette. Click on a bare spot in your workspace to place it.

2 ♦ Query Builder immediately opens. Define the query. See the "Defining and Editing Queries Using Query Builder" section in Chapter 4, "Data-access Objects," for more information.

**NOTE:** While you can elect to jump out of Query Builder and create a free-form query, you cannot later use Query Builder to maintain it. Confirming that you wish to create a free-form query locks out Query Builder immediately; you cannot change your mind. Your only recourse will be to delete the Browse object altogether and create a new one.

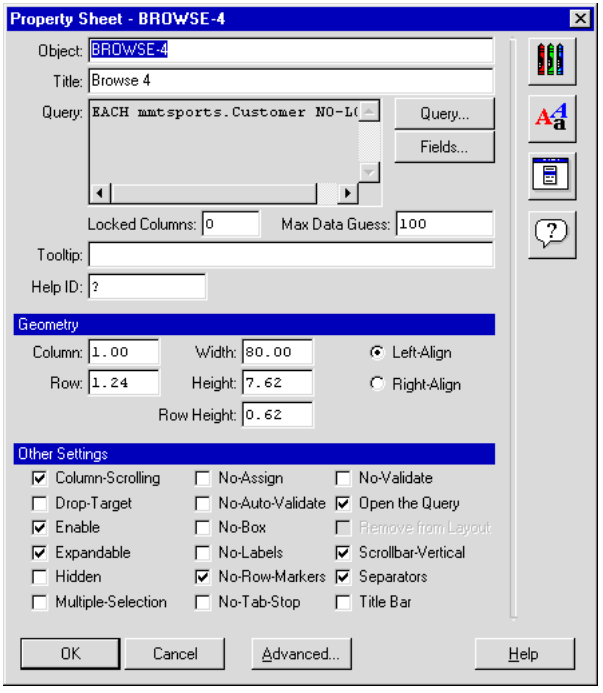
3 ♦ When you have defined the query terms and chosen the fields the Browse will display, choose OK to dismiss Query Builder. Your Browse widget displays the fields you have chosen.

4 ♦ Select the Browse, if necessary, and drag the handles to resize and position it to meet your needs. Note that the Browse does not display a horizontal scroll bar as long as it has enough space to display at least a fragment of the rightmost field. Check to be sure it displays the whole field.

## 5.8.2 Configuring a Basic Data Browser Instance

You can make a number of changes to the appearance and behavior of your basic Browse object, if the default settings in the property sheet do not meet your needs.

Click on the object to select it, if necessary, and choose Tools→ Property Sheet. The Property Sheet dialog box opens:



### Minimal Configuration

Although you can accept many of the default settings, you might wish to make two changes to individualize this object:

- Change the identifier to more closely reflect this Data Browser's role in your application.
- Add a title string, if you plan to allow the Browse to display a title bar.

### Query and Fields

If the Query and Fields buttons are visible, it means you can edit those elements using Query Builder and Column Editor rather than the Section Editor. If the buttons are not visible, you must use the Section editor, and the balance of this section does not apply to this Browse.

Since the query is local to this Browse, rather than in a separate module, you can make changes both to the terms of the query and to the fields the Browse will display.

- Choosing the Query button will start Query Builder and allow you to modify the terms of the query. See the "Defining and Editing Queries Using Query Builder" section in Chapter 4, "Data-access Objects," for more information.
- Choosing the Fields button will start Column Editor and allow you to modify the columns this SmartDataBrowser will display. For information about how to use Column Editor, see the "Selecting Database Fields for Browsing" section in Chapter 4, "Data-access Objects."
- By default, all columns can be scrolled horizontally, if there are too many columns to display at once. You can choose to lock the leftmost columns in place, so that scrolling does not affect them. Enter in the Locked Columns box the number of columns you want to freeze in place.

**CAUTION:** If you lock more columns than will fit in the Browse's window, you effectively disable the horizontal scroll bar.

- When displaying a vertical scroll bar, the system tries to size the scroll bar's thumb slider to reflect the total number of lines in the data stream. Ideally, the size of the thumb varies inversely with the number of lines (records). In general, the number of records being returned by a query will be larger than the number of lines that can be buffered and, when that is the case, the system cannot begin to correctly size the thumb. If you have a good idea about the total number of records in this data stream, you can improve the accuracy of the scroll bar's appearance. Enter your estimate in the Max Data Guess field.
- Type into the Tooltip field the identifying text that you want the system to display whenever the mouse cursor hovers over this Browse.

### Geometry

You can set the exact XY origin (relative to the containing Frame) and size of this Browse instance. Set the values, in character units, in the appropriate fields. You can choose whether the origin is measured at the upper-left corner (the default) or the upper-right corner.

If you would prefer to use pixels rather than character units, choose the Advanced button and set the values using that dialog box instead.

### Miscellaneous Properties (Other Settings)

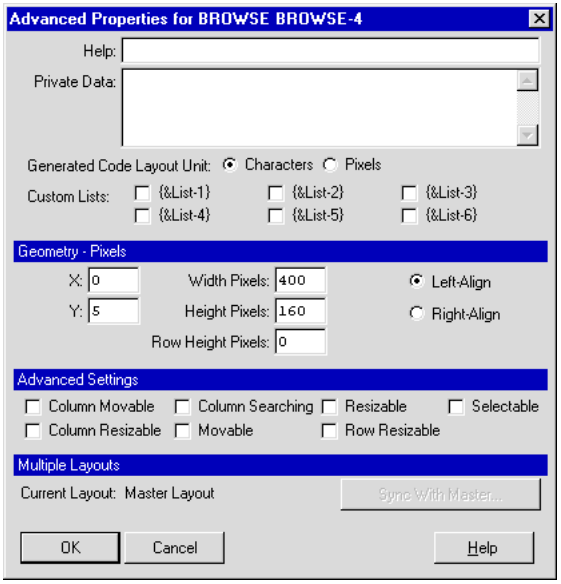
You can make a number of changes to the appearance and behavior of this Browse instance, if the default settings do not meet your needs:

- **Column Scrolling** — Normally set. Clearing this check box causes the horizontal scrolling granularity to be one character. The default is one field (column). This option applies only when running under MS-Windows.
- **Drop Target** — Normally cleared. Setting this check box causes the system to generate an event when the user drags and drops another object onto this one. You must write code to deal with that event.
- **Enable** — Normally set. Clearing this check box causes the Browse to be insensitive.
- **Expandable** — Normally set. Clearing this check box prevents stretching the last column to cover any background exposed by scrolling. In general, when Column Scrolling is set, and the Browse window is scrolled all the way to the right, some background will be exposed because it is unlikely that the last *n* columns will exactly fit the Browse window width. When this option is set, the last column expands cosmetically to the right to mask any exposed background from view. Applies only when running under MS-Windows.
- **Hidden** — Normally cleared. Setting this check box prevents implicit display of the object.
- **Multiple Selection** — Normally cleared. Setting this check box allows the user to select multiple rows. If any of the fields are updatable, you must also clear the No Row Markers check box.
- **No Assign** — Normally cleared. When set, changes to updatable fields are not automatically written back to the database when the user moves to a different row (record).
- **No Box** — Normally cleared. Setting this check box causes the Browse window to display without a border. Under MS-Windows, this border is a single-pixel black line.

- **No Labels** — Normally cleared. Setting this check box prevents display of the column labels.
- **No Tab Stop** — Normally cleared. Setting this check box removes the Browse instance from the TAB-traversal list for the containing Frame. Under MS-Windows, removing the object from the traversal list has some effects that might confuse the user.
- **No Validate** — Normally cleared. Setting this check box disables all validation conditions specified in the Data Dictionary for fields in this Browse.
- **Open the Query** — Normally set. Setting this check box activates the query and populates the fields during initialization.
- **scroll bar Vertical** — Normally set. Clearing this option hides the vertical scroll bar. The user can still scroll using the arrow keys.
- **Separators** — Normally set. Clearing this check box causes the widget to paint itself without lines separating the rows and columns.
- **Title Bar** — Normally cleared. Setting this check box causes the Browse widget to display a full-width Frame-like title bar. Specify the title string if you set this option.

### Advanced Properties

Choose the Advanced button. The Advanced Properties dialog box opens:



You can make a number of changes to the Browse's appearance and behavior by changing the default settings in this dialog box:

- **Help** — The text in this field is displayed in the status bar of the parent window when the Browse has the focus. If the parent window has no status area, this text never displays.
- **Private Data** — AppBuilder writes the contents of this field out as the value of the PRIVATE–DATA attribute. You can write code to use this data in any way that meets your needs.
- **Generated Code Layout Unit** — Controls whether AppBuilder describes the position and size of this Browse in character or pixel units, when it generates source code.
- **Geometry** — Expresses the origin and size of this widget in pixels. You can set a particular origin and size for the object here rather than interactively, if you wish. You can also choose to have the origin be measured at the upper-right rather than the upper-left corner.
- **Column Movable** — Normally cleared. Setting this check box allows the user to reorder the columns by dragging. Such reordering is cosmetic and has no effect on the actual record.
- **Column Resizable** — Normally cleared. Setting this check box allows the user to resize the columns interactively. Resizing the column has no effect on the actual field widths.
- **Column Searching** — Normally cleared. Setting this check box allows the user to search on a selected column (field).
- **Movable** — Normally cleared. Setting this check box, together with the Selectable box, allows the user to move the entire Browse object around within the bounds of the enclosing Frame.
- **Resizable** — Normally cleared. Setting this check box, together with the Selectable box, allows the user to change the size of the Browse object, within the limits of the enclosing Frame.
- **Row Resizable** — Normally cleared. Setting this check box allows the user to change the vertical size (height) of the rows by dragging a horizontal separator line.
- **Selectable** — Normally cleared. Setting this check box allows the use to select the Browse object.

## 5.9 Combo Boxes

The Combo Box object combines a Fill–in with a Selection List. You might use it in cases where:

- The user will have to choose a single item from a list.
- A Radio-button Set is not an appropriate representation.

A Combo Box will generally be a better representation than a Radio Set when the set of choices is large, varies in size, or involves long strings. A Drop–down List or Combo Box will generally be a better choice than a Simple List or Combo Box when space is at a premium, as is often the case. Drop–down Lists/Combo Boxes also visually emphasize the current choice more than Radio Sets or Simple Lists/Combo Boxes do.

AppBuilder offers two different Combo Box objects for your use:

- The basic 4GL Combo Box
- An ActiveX (OCX) Combo Box

### 5.9.1 Creating a Basic Combo Box Instance

Creating and placing a basic Combo Box instance is very simple. Follow these steps:

1 ♦ Click on the basic Combo Box tool icon in the Object Palette:



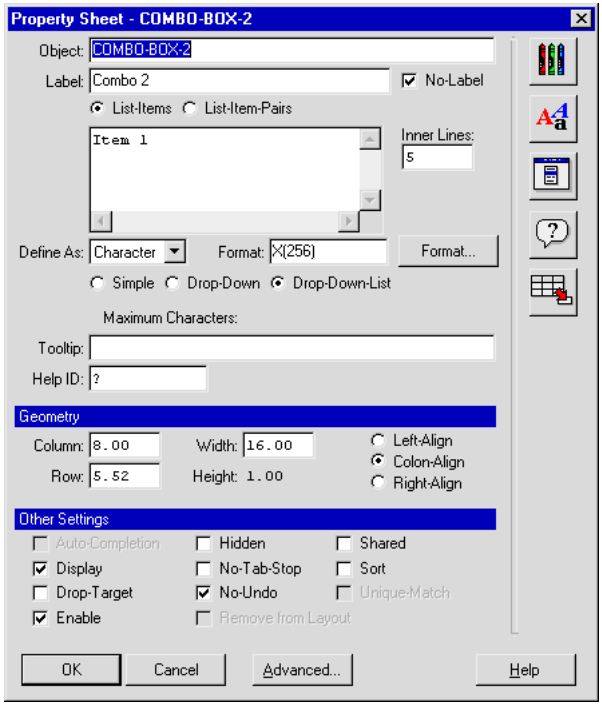
Be sure to select the one shown here. The other Combo Box icon (not shown) represents the OCX version.

2 ♦ Move the mouse cursor over a bare spot in your workspace and click to place the new Combo Box.

3 ♦ Configure and size it.

## 5.9.2 Configuring a Basic Combo Box Instance

To configure your Combo Box object, begin by selecting the object and choosing Tools→ Property Sheet. The dialog box will open:




### Minimal Configuration

Although you can accept the default values for many object properties, you should at least make these changes:

- Change the Object identifier and Label to better represent the role of this Combo Box in your application.
- Type in the items that will populate the list, one item per line. You might need to change the data type first. See the Define As and Format entries in the "Size and Style Configuration" section.
- If appropriate, choose the Advanced button and, when the Advanced Properties dialog box opens, enter the Initial Value.

### Size and Style Configuration

- **No Label** — Normally cleared. Setting this check box turns off display of any label. Setting this check box does not clear the actual label text; the label string remains visible in the property sheet. The only other effect of setting or clearing this box is to alter the values in the Geometry section so that the Combo Box stays at the same XY location within the enclosing Frame.

- **Inner Lines** — Type in the number of lines to be displayed by the selection-list component when it is open. The default value is 5 lines.

- **Define As** — Choose the data type from the list. The default type is CHARACTER, but you can choose DATE, DECIMAL, INTEGER, or LOGICAL instead. AppBuilder will seed the list with a single item of that type.

- **Format** — Enter the formatting string for the data type you have chosen. If you would prefer to pick the format string from a list rather than enter it by hand, choose the Format button.

- **Tooltip** — This Combo Box displays any text you enter here whenever the user allows the mouse cursor to hover over it.

- **Help ID** — If you plan to provide context-sensitive help in your application for this object, type in a unique integer as its identifier.

- **Simple / Drop-down / Drop-down List:**

  – **Simple** — In this style, you must explicitly make the Selection List component visible at design time, if it is to be visible at all. By default, AppBuilder generates the widget with only the Fill–in visible. Drag the bottom handle to reveal as many lines of list as you like. In this style, the Fill–in element is editable.

  – **Drop-Down** — In this style, the user can open and pick from the list, or type a value into the Fill–in component. You can write code to permanently add the typed-in value to the entries in the list, if you wish, or discard the entry after use.

  – **Drop-Down List** — (Default) In this style, the Fill–in component is read-only—the user cannot type into it. The user can only open and pick from the list. The choice then appears in the Fill–in component. Note that in other major widget taxonomies, a Drop-Down List is a type of Selection List, not a type of Combo Box. Note, too, that this style is not the default for the ActiveX Combo Box.

- **Color, Type Style, Popup, String Attributes, Data Field** — Choose the appropriate button to set these properties. For further information, see Appendix A, "Frequently Used Dialogs."

- **Geometry** — Although it is often more convenient to set the XY origin and size of a widget visually in the workspace, you can set those values explicitly here, if you prefer. You can also choose a particular alignment, though the only effect this has is to change the X origin in the source code.

- **Auto-completion** — Normally cleared. Setting this box causes the Combo Box to attempt to complete the entry the user is typing. This option is only meaningful where the Fill–in portion of the Combo Box is editable.

- **Display** — Normally set. Clearing this box prevents this Combo Box from automatically populating its Fill–in during initialization.

- **Drop Target** — Normally cleared. Setting this box causes this Combo Box to experience an event whenever the user drags another object onto it. You must write the appropriate event-handling code.

- **Enable** — Normally set. Clearing this box makes the Fill–in part unresponsive to input.

- **Hidden** — Normally cleared. Setting this box prevents the Combo Box from responding to implicit requests to display itself. It will only honor explicit requests.

- **No Tab Stop** — Normally cleared. Setting this box removes this Combo Box from the enclosing Frame's traversal list. Normally, pressing the key causes focus to move from one object to the next in the list. When this box is set, the object can neither lose nor gain focus when the user presses the **TAB** key. If the widget has the focus, it will ignore the key, and if it does not, focus will cycle through the other members of the traversal list while ignoring this widget.

**CAUTION:** Making this Combo Box Movable or Resizable (Advanced Properties) prevents this object from gaining input focus using the mouse. So if you also set No Tab Stop, you effectively prevent the widget from ever gaining input focus.

- **No Undo** — Cleared. Cannot be set. Progress will always journal changes to the data in this Combo Box.

- **Remove from Layout** — Cleared. Cannot be set unless defining an alternate layout.

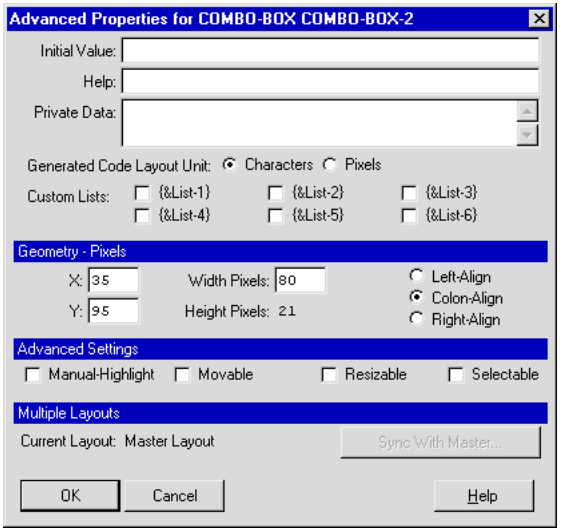
- **Shared** — Cleared. Setting this box causes the Combo Box to be reachable outside the defining procedure.

- **Sort** — Normally cleared. Setting this box causes the Combo Box to always display the items in its drop-down list in sorted order, regardless of their real order.

- **Unique Match** — Normally cleared. (This option is not available unless you set Auto-Completion.) Setting this box causes the Combo Box to wait for a unique substring before completing the entry.

### Advanced Properties

Choose the Advanced button. The Advanced Properties dialog box opens:




You can change any of the values here that do not meet your needs:

- **Initial Value** — You can enter a value here that will appear in the Fill–in component of the Combo Box on startup.

- **Help** — The text you enter here will display in the status area of the enclosing Window, whenever this object has input focus. If you choose to let this Combo Box inherit the help string defined in the Data Dictionary's schema, you will not be able to enter text here, even if there is no help string in the schema. If the Combo Box is assigned to a Dialog Box rather than a Window, or if the Window has no status area, this text will not display.

- **Private Data** — AppBuilder writes this data out as the value of the PRIVATE–DATA attribute. You can write code to make use of this data in any way that meets your needs.

- **Generated Code Layout Units** — You can elect to have AppBuilder define this object in character (the default) or pixel units, in the source code it generates.

- **Custom Lists** — The custom lists are macros belonging to the enclosing Frame. They allow you to treat many objects alike without having to refer to them individually. You can include this object in any or all of the lists.

- **Geometry** — You can set the XY origin and closed size of this Combo Box in pixel units. This section is identical to the corresponding section in the base property sheet apart from the type of units you use.

- **Manual Highlight** — Normally cleared. Setting this box allows you to write code to define a custom highlight effect. Your custom effect will be applied whenever this Combo Box has input focus.

- **Movable** — Normally cleared. Setting this box makes it impossible for the user to give this Combo Box input focus by clicking with the mouse. Instead, the user can use the mouse to drag the Combo Box body—but not the label, which stays where it was—to a different position within the bounds of the enclosing Frame.

- **Resizable** — Normally cleared. Setting this box makes it impossible for the user to give this Combo Box input focus by clicking with the mouse. Instead, if the Selectable box is also set, clicking with the mouse causes handles to appear so that the user can resize the object.

- **Selectable** — Normally cleared. Setting this box allows the user to select this Combo Box.

**CAUTION:** If you make this Combo Box Movable or Resizable, and you also set the No Tab Stop option in the base Property Sheet, you effectively prevent this object from ever gaining input focus.

### 5.9.3 Operating a Basic Combo Box

The Basic Combo Box widget consists of a joined fill-in and selection list. You can manipulate the contents of the list under program control.

#### Clearing the List Buffer

The list of items is stored as a string, pointed at by the LIST-ITEMS attribute. You can clear the list by assigning the empty string:

```
comboExample:LIST-ITEMS = "" .
```

Setting LIST-ITEMS to the empty string clears the buffer immediately.

**CAUTION:** The Combo Box widget is created with a single item in its list. You should always clear that item as part of your initialization process.

You can also use a loop to delete the line items one by one:

```
DO WHILE comboExample:DELETE ( 1 ): /* empty statement */ END.
```

In the example code, the DELETE method is called in a WHILE loop to repeatedly delete line item 1. When the buffer is empty of lines, DELETE will return FALSE and the WHILE loop will terminate.

### Adding a Line Item

Using the INSERT() method, you can add a line item at any offset in the list, regardless of the Sort option.

The ADD-FIRST() and ADD-LAST() methods have a different result depending on whether the Sort option is set. If the option is not set, new items will be added to the top or bottom of the list, depending on which method you call. If the Sort option is set, new items will be inserted in sorted order regardless of which method you call.

The INSERT(), ADD-FIRST() and ADD-LAST() methods all return TRUE if they succeed, FALSE if they cannot perform the insertion.

- Adds a simple item as the new fifth line:

```
comboExample:INSERT ( "Toy Item", 5 ).
```

- Finds the first exact match of "Target Item" and inserts "Toy Item" as a new line above it:

```
comboExample:INSERT ( "Toy Item", "Target Item" ).
```

- Adds a simple item as the new first line:

```
comboExample:ADD-FIRST ( "Toy Item" ).
```

- Adds a simple item as the new last line:

```
comboExample:ADD-LAST ( "Toy Item" ).
```

If you are using simple line items—not item/value pairs—you can add more than one in a single call by combining them into a comma-separated list: "Item 1, Item 2, . . .,Item *n*".

- Adds three simple items as the new 5th, 6th, and 7th lines:

```
comboExample:INSERT ( "Line 5 text,Line 6 text,Line 7 text", 5 ).
```

If you are adding item/value pairs, you can only add one line per call.

- Adds three item/value pairs as the new 5th, 6th, and 7th lines, with values 25, 48, and 16:

```
comboExample:INSERT ( "Line 5", 25, 5).
comboExample:INSERT ( "Line 6", 48, 6).
comboExample:INSERT ( "Line 7", 16, 7).
```

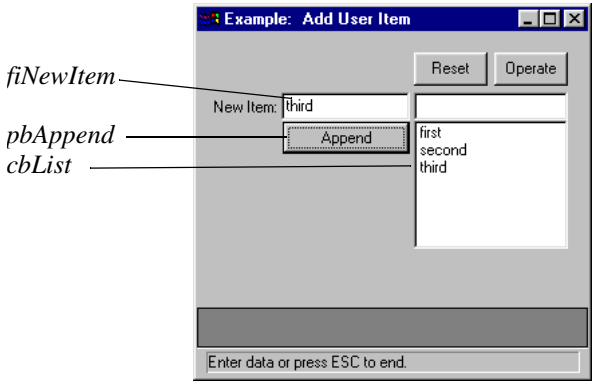
Multiple item/value pairs cannot be added using a single call. This code looks reasonable but would **not** compile successfully:

```
comboExample:ADD-FIRST ( "Line 5", 25, 5,
                          "Line 6", 48, 6,
                         "Line 7, 16, 7 ) .
```

### Adding a User-supplied Line Item

Although a Combo Box appears to be a combined Fill-in and List, you cannot capture user input as new items. The Fill-in's function is only to provide a way to select a list item other than by scrolling to find it.

To allow the user to add items to the list, use a separate Fill-in to capture the text for the new item or item/value pair and call the appropriate insertion method to add it:



This code, if used as the CHOOSE trigger for the pbAppend pushbutton, will move non-empty strings from the Fill-in to the list in the Combo Box:

```
IF fiNewItem:SCREEN-VALUE <> "" THEN DO: /* ignore event if field empty */
   ASSIGN fiNewItem.                     /* set the value of the Fill-in*/
   cbList:ADD-LAST( fiNewItem ).         /* append the new value to list */
   fiNewItem:SCREEN-VALUE = "".          /* clear Fill-in's image... */
   fiNewItem = "" .                      /* ...and its value */
END.
```

### Deleting a Line from the List

Deleting a line requires only that you know the line number. This code will delete line 5 if it exists:

```
comboExample:DELETE ( 5 ).
```

### Finding a Line or Item in the List

- You can determine the line offset of some item using the LOOKUP() method:

  ```
  lineoffset = comboExample:LOOKUP( "Line Item of Interest" ).
  ```

  Note that LOOKUP():

  – Always finds the first instance of the string, if there are duplicates.

  – Returns 0 (zero) if the string is not found

- You can determine the string value of some item using the ENTRY() method:

  ```
  strItemValue = comboExample:ENTRY ( 5 ).
  ```

  Note that ENTRY() will complain and return the undefined value (?) if you pass an argument that does not index an item in the list.

### 5.9.4 Creating an ActiveX Combo Box Instance

Creating and placing an ActiveX Combo Box instance is very simple. Follow these steps:

1 ♦ Click on the OCX Combo Box tool icon in the Object Palette:



Be sure to select the one shown here. The other Combo Box icon (not shown) represents the version defined in the 4GL.

2 ♦ Move the mouse cursor over a bare spot in your workspace and click to place the new Combo Box.

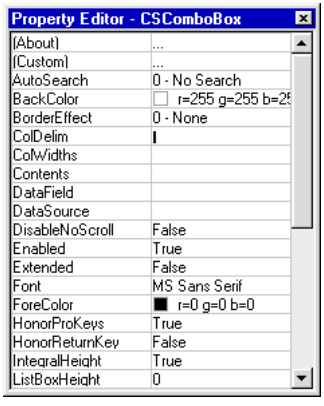
3 ♦ Configure and size it.

### 5.9.5 Configuring an ActiveX Combo Box Instance

ActiveX (OCX) objects are functionally similar to native Progress objects, but written to the Component Object Model (COM) standard defined by Microsoft. To adapt them for use in the Progress environment, AppBuilder automatically supplies a two-layer interface object, the Control Frame. Control Frames translate between the COM world and the 4GL world.

When you select an ActiveX (OCX) object and choose Tools→ Property Sheet, the dialog box that opens in response displays the few properties belonging to the 4GL layer of the Control Frame. See the "ActiveX Control Frame Properties" section in Appendix A, "Frequently Used Dialogs," for further information.

To open the properties dialog box for the Combo Box itself, double-click on the object. The special ActiveX-style properties dialog box opens:



References in your code to the native ActiveX properties of the Combo Box instance must take the indirect form *Com-Handle:ActiveX-Identifier:Property-Identifier.*

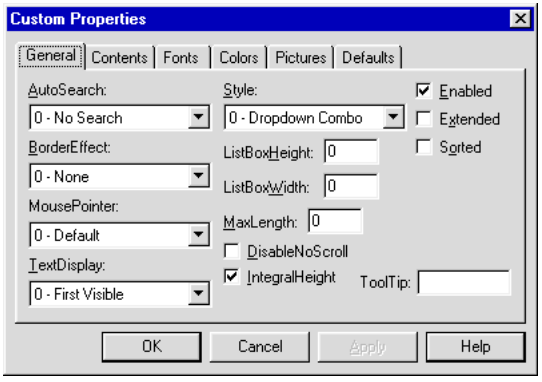
### Minimal Configuration

While you can accept many of the default settings, you will need to make some changes:

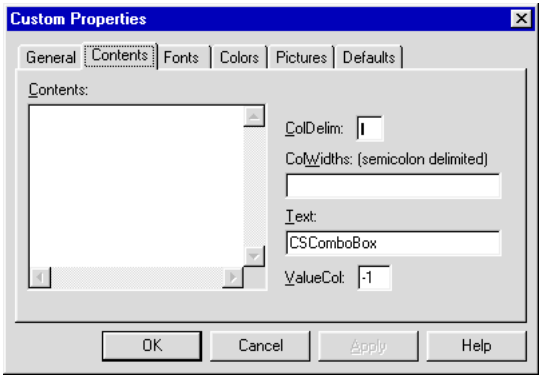
- Replace the default Object identifier for the Control Frame with one that more clearly represents the role of this Combo Box instance in your application. You can do this directly in the main AppBuilder window, when the instance is selected. You need not change the identifier of the actual ActiveX object itself, though you can if you wish.
- Populate the list of choices that the Combo Box will use.

To populate the list of choices, follow these steps:

1 ♦ Click in the [Custom] property data field to expose the ellipsis button, then choose the button to open the Custom Properties dialog box:



2 ♦ Click on the Contents tab to expose that field:




Type in this Combo Box's list of items, one item per line. When finished, choose OK to close the dialog box.

### Additional Property Configuration

Choose from three different styles:

- **0 Dropdown Combo** — Default. The user can open and pick from the list, or type a value into the Fill–in component. Note that this style is not the default style for the 4GL Combo Box.

- **1 Simple Combo** — In this style, you must explicitly make the Selection List component visible at design time, if it is to be visible at all. By default, AppBuilder generates the widget with only the Fill–in visible. Drag the bottom handle to reveal as much of the List element as you like. In this style, the Fill–in element is editable.

- **2 Dropdown List** — In this style, the Fill–in component is read-only—the user cannot type into it. The user can only open and pick from the list. The choice then appears in the Fill–in component. Note that, in other taxonomies, a Drop-Down List is a type of Selection List, not a type of Combo Box Configuration

### Code Configuration

When you run a workspace for the first time after placing your first ActiveX (OCX) object in it, AppBuilder creates the control_load procedure. You can inspect that procedure in the Section Editor, but not edit it.

In that procedure, AppBuilder creates the variable of type COM–HANDLE that you will use when you write event handlers. The identifier of the variable is the same as the identifier you

assign to the Control Frame's Object field, with "ch" prepended. Thus, if you assign the identifier Example to the Control Frame, the COM–HANDLE identifier will be chExample.

The Combo Box stores its current string value in the Value property. To determine the index value of that string (the topmost list item is index 0), pass the Value property to the FindStringExact() method. Presuming you use the Object identifier Example for the OCX Control Frame, this toy code will detect the user's selection, and print out the string and index values:

```
/*
**  Add to the Definitions section
*/

DEFINE VARIABLE sSelection AS CHARACTER INITIAL "".
DEFINE VARIABLE iSelection AS INTEGER INITIAL 0.

/*
**  Add as the Trigger for event OCX.Click
*/

ASSIGN
  sSelection = chExample:CSComboBox:Value
  iSelection = chExample:CSComboBox:FindStringExact( sSelection, -1 ).
MESSAGE "Selection is " sSelection "at index" iSelection .
```

For further information about events, methods, and properties for the three OCX objects that AppBuilder supplies in the Objects Palette, see the online help.

## 5.10 Basic Data Viewers

The basic Data Viewer object, or DB–Fields as it is also called, corresponds in purpose to the SmartDataViewer object. While Browsers present fields from many records at once, in a row/column tabular format, the Viewer objects display fields from a single record using Fill–in widgets, one Fill–in per field. The Viewer object can display any of the standard data types that can be represented by ASCII text.

Unlike a SmartDataViewer, the basic Viewer object manages its own data stream, communicating with the data source on its own behalf rather than tapping into the data stream supplied by a modular data pump such as the SmartDataObject.

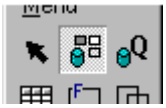
Although you can freely place a basic Data Viewer object in a Smart workspace, for example, a SmartWindow, doing so will not enable it to take advantage of SmartLinks or participate in ADM paging.

### 5.10.1 Creating a Basic Data Viewer Instance

Unlike a SmartDataViewer, a basic Data Viewer (DB–Fields) object is not based on a master, but you must define the query it will use.

To create and place a basic Data Viewer object, follow these steps:

1 ♦ Click on the basic Data Viewer (DB–Fields) tool icon in the Object Palette:



2 ♦ Click on a bare spot in your workspace to begin the placement process. Query Builder will immediately open.

3 ♦ Define the query. See the "Defining and Editing Queries Using Query Builder" section in Chapter 4, "Data-access Objects," for more information.

**NOTE:** While you can elect to jump out of Query Builder and create a free-form query, you cannot later use Query Builder to maintain it. Should you change your mind after confirming that you wish to create a free-form query, your only recourse will be to delete the Viewer object altogether and create a new one.

4 ♦ When you have defined the query terms and chosen the fields the Viewer will display, choose OK to dismiss Query Builder. Your Viewer widgets will appear in a single column in your workspace.

5 ♦ If the default single-column arrangement does not meet your layout needs, select and move the fields, either individually or in subgroups, into a more appropriate arrangement. Since there is no master object involved, the layout you create will have no effect on any basic Data Viewers you place in future.

### 5.10.2 Configuring a Basic Viewer Instance

Besides rearrangement, you can make a number of other changes to the appearance and behavior of your basic Viewer object, if the default settings do not meet your needs. You make such changes via the Property Sheets of the individual Fill–ins.

See "Configuring a Fill–in Instance," in this chapter, for more information.

## 5.11 Editors (Multi-line)

The Editor widget provides the ability to capture large amounts of text in a way familiar to and convenient for the user. You might use an Editor object to capture, for example, free-form comments of undetermined length.

Although they are basic 4GL objects rather than SmartObjects, you can add a considerable amount of additional capability to them through calls to 4GL methods for searching, etc.

### 5.11.1 Creating an Editor Instance

Because the Editor widget is a basic 4GL object, creating one is very simple. Follow these steps:

1 ♦ Click on the Editor tool icon in the Object Palette:



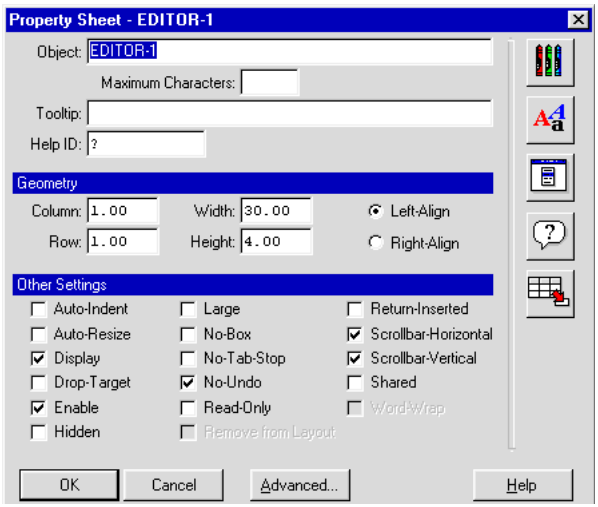
2 ♦ Move the mouse cursor over a bare spot in your workspace and click to place the new Editor object.

3 ♦ Configure and size it.

### 5.11.2 Configuring an Editor Instance

You can change some basic characteristics of the Editor widget's appearance and behavior through its property sheet, if the default settings do not meet your needs. Note, however, that the more sophisticated behaviors, such as searching, cannot be enabled in the property sheet.

To configure your Editor object, begin by selecting the object and choosing Tools→ Property Sheet. The dialog box opens:




### Minimal Configuration

Although you might choose to accept the other default settings, you should make at least the first of these two changes:

- Change the default identifier to one that more closely reflects the role of this object in your application.
- Turn off the Horizontal scroll bar and turn on Word Wrap. The default style produces a single long line of input unless the user explicitly uses the **RETURN** key. This might not be the most appropriate style for the needs of your application.

### Edit-buffer Size

The default size of an Editor widget's buffer is 20KB under MS-Windows. You can reduce memory requirements by forcing a smaller buffer size. Enter the desired value in the Maximum Characters field.

### Help

Add one or both of these help elements:

- Add a Tooltip string, if appropriate. The Editor will display that string whenever the user allows the mouse cursor to hover over the widget.
- If you will provide context-sensitive help for this object, add a unique integer Help ID.

### Geometry

AppBuilder allows you to adjust the position and size of your widgets interactively, by dragging their handles. The changes you make in this way are reflected in the Geometry values. You can also change these values explicitly, here.

AppBuilder normally generates source code that identifies the upper-left corner of the object as its origin. You can choose to have the upper-right corner identified as the origin instead. This alignment setting does not change the actual position of the object in any way.

### Miscellaneous Properties (Other Settings)

You can change as many of these properties as you like:

- **Auto-indent** — Normally cleared. Setting this box causes a new line to inherit the indenting of the preceding line. This is now a universal feature in programmers' editors, and is often a feature, in some form, in word-processing editors as well.
- **Auto-resize** — Normally cleared. Setting this box causes the Editor to automatically resize, if necessary, to keep the same amount of text in view despite changes in size or style of type.
- **Display** — Normally set. Clearing this box prevents the Editor from making its contents visible when it enables itself for input. The edit buffer will appear to the user to be empty.
- **Drop Target** — Normally cleared. Setting this box causes the Editor to experience an event whenever the user drops some other object onto it. You must write code to handle the event.
- **Enable** — Normally set. Clearing this box prevents this object from accepting the input focus.
- **Hidden** — Normally cleared. Setting this box prevents this object from displaying itself in response to implicit requests. If you set this box, you will need to include explicit code to make the object visible.
- **Large** — Normally cleared. Setting this box changes the size of the edit buffer from the 20KB default to some larger size consistent with system resources. Applies only to MS-Windows.
- **No Box** — Normally cleared. Setting this box prevents the Editor from displaying an outline and causes it to change its background color to be the same as that of the application. If you also turn off the scroll bars, the object will be nearly invisible. You can restore some visibility by forcing the background color to white. See the "Choosing Colors" section in Appendix A, "Frequently Used Dialogs."

- **No Tab Stop** — Normally cleared. Setting this box removes the Editor from its parent Frame's traversal list. An object not in the traversal list can neither gain nor lose focus when the user presses the **TAB** key.
- **No Undo** — Normally set. Clearing this box causes the Editor to journal all changes made to its buffer.
- **Read-only** — Normally cleared. Setting this box prevents the Editor from allowing changes to its buffer contents, even though it will accept the input focus.
- **Remove from Layout** — Cleared. Cannot be set unless editing an alternate layout.
- **Return Inserted** — Normally cleared. When this box is cleared, if the Editor is a child of a Dialog Box that has a default button, a RETURN event chooses the default button. In all other cases, a RETURN event inserts a hard return at the cursor. Applies only to the MS-Windows platform.
- **Scrollbar Horizontal/Vertical** — Normally set. Clearing one or both of these boxes turns off the respective scroll bar(s). Turning off the horizontal scroll bar automatically sets the Word Wrap box.
- **Shared** — Normally cleared. Setting this box makes this widget and its edit buffer available to other procedures.
- **Word Wrap** — Normally cleared. Setting this box causes words to be wrapped to the next line if they would otherwise be clipped by the right edge of the Editor's client area. This box is set by default when you turn off the horizontal scroll bar.

### Advanced Properties

The Editor widget has only the standard Advanced Properties, but you can change those that do not meet your needs. See the "Advanced Properties" section in Appendix A, "Frequently Used Dialogs" for more information.

## 5.12 Fill–ins (Single-line Editors)

The Fill–in widget provides the ability to capture and/or display a small amount of text, typically a single token or short string. Fill–ins do not allow embedded line breaks. You can choose to have text captured by a Fill–in undergo automatic type conversion and validation.

### 5.12.1 Creating a Fill–in Instance

Because the Fill–in widget is a basic 4GL object, creating one is very simple. Follow these steps:

1 ♦ Click on the Fill–in tool icon in the Object Palette:



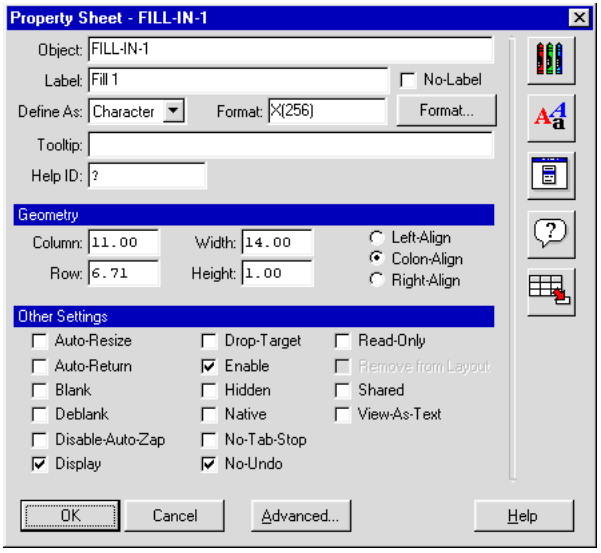
2 ♦ Move the mouse cursor over a bare spot in your workspace and click to place the new object.

3 ♦ Configure and size it.

### 5.12.2 Configuring a Fill–in Instance

You can change some basic characteristics of the Fill–in widget's appearance and behavior through its property sheet, if the default settings do not meet your needs.

To configure your Fill–in object, begin by selecting it and choosing Tools→ Property Sheet. The dialog box opens:



### Minimal Configuration

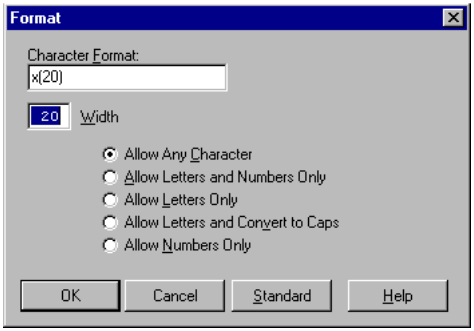
Although you can accept many of the default settings, you should make at least three changes:

- Change the AppBuilder-generated Object identifier to one that more accurately reflects the role of this Fill–in in your application.
- Change the generic label to one that will be meaningful to the users of your application. Or, if appropriate, set the No Label check box to hide the label. Setting this check box does not clear the actual label text; the label string remains visible in the property sheet. The only other effect of setting or clearing this box is to alter the values in the Geometry section so that the Fill–in stays at the same XY location within the enclosing Frame.
- Click the Advanced button and set the initial value, if any, that the Fill–in is to display. By default, the initial value is empty/undefined.

### Internal Data Format

You can use a Fill–in to capture any data type whose value can be entered from the keyboard and represented on the display as ordinary text. The Fill–in will do a limited amount of syntax-checking for you if you appropriately set the values for Define As and Format.

- **Define As** — Choose the data type from the list. The default type is CHARACTER, but you can choose DATE, DECIMAL, INTEGER, LOGICAL, or RECID instead. RECIDs are internal record identifiers.
- **Format** — Enter the formatting string for the data type you have chosen. If you would prefer to pick the format string from a list rather than enter it by hand, choose the Format button to open a dialog box specific to the type you have chosen. Shown here is the dialog that displays for type CHARACTER:



### Help

You can supply helpful information to your users in several ways. Here are two of the most common:

- Add a Tooltip string. Under MS-Windows, Tooltips display whenever the user allows the mouse cursor to hover for a few seconds over an object for which a Tooltip is defined.
- Add a unique integer as the identifier for context-sensitive help, if you plan to provide such help as part of your application.

### Geometry

Reflects the XY origin and size, in character units, of the Fill–in. By default, the Fill–in is colon-aligned. You can change that to left- or right-alignment, though the only noticeable effect that will have is to change the X origin value.

### Miscellaneous Properties (Other Settings)

These properties modify the widget's appearance or behavior in various ways. You can change as many as you like:

- **Auto-Resize** — Normally cleared. Setting this box causes the object to automatically change its displayed size to agree with the current type size.
- **Auto-Return** — Normally cleared. Setting this box causes focus to move to the next object in the traversal list once this Fill–in has accepted as many characters as it can.
- **Blank** — Normally cleared. Setting this box prevents the Fill–in from echoing input back to the display. This is useful for password fields and similar applications.
- **Deblank** — Normally cleared. Setting this box causes the Fill–in to automatically discard any leading blanks from input.
- **Disable Auto-zap** — Normally cleared. Setting this box prevents the Fill–in from automatically clearing its input field whenever it gets the focus.
- **Display** — Normally set. Clearing this box prevents the object from automatically showing the current value of the associated variable during initialization. To the user, the value would appear undefined.
- **Drop Target** — Normally cleared. Setting this box causes the Fill–in to experience an event message whenever the user drops another object onto this one. You must write the appropriate event-handling code.
- **Enable** — Normally set. Clearing this box makes the Fill–in unresponsive to input.

- **Hidden** — Normally cleared. Setting this box prevents the Fill–in from responding to implicit requests to display itself. It will only honor explicit requests.
- **Native** — Normally cleared. Setting this box causes the Fill–in to vary its behavior according to the underlying platform (MS-Windows, for example) rather than behaving the same way regardless of platform.
- **No Tab Stop** — Normally cleared. Setting this box removes this Fill–in from the enclosing Frame's traversal list. Normally, pressing the key causes focus to move to the next object in the list. When this box is set, the Fill–in can neither lose nor gain focus when the user presses the **TAB** key. If the widget has the focus, it will ignore the key, and if it does not, focus will cycle through the other members of the traversal list while ignoring this widget.

**CAUTION:** Making this Fill–in Movable or Resizable (Advanced Properties) prevents this Fill–in from gaining input focus using the mouse. So if you also set this property, you effectively prevent the widget from ever gaining input focus.

- **No Undo** — Normally set. Clearing this box causes the Fill–in to journal all changes.
- **Read-Only** — Normally cleared. Setting this box prevents the user from changing the content of the Fill–in.
- **Remove from Layout** — Cleared. Cannot be set unless defining an alternate layout.
- **Shared** — Normally cleared. Setting this box makes the value of this Fill–in available to other procedures.
- **View as Text** — Normally cleared. Setting this box causes the Fill–in to display itself as though it were static text: read-only, and without any border or 3D effect.

### Advanced Properties

A Fill–in widget has only the standard Advanced Properties, but you can change those that do not meet your needs. See the "Advanced Properties" section in Appendix A, "Frequently Used Dialogs," for more information.

## 5.13 Radio-button Sets

The Radio-set widget provides a way to force a single choice from a small set of possibilities. Unlike a logical set that you might construct using Toggle Boxes, a Radio–button Set must always have exactly one member—no more, no fewer—selected. Thus, you cannot use Radio-button Sets when you wish to allow no selection or multiple selection.

The Radio–set widget will represent a value of any standard type, including CHARACTER. Typically, though, you would use a Radio–button Set for values that are basically nontext-numeric, symbolic-in nature.

By default, AppBuilder supplies a set of three buttons, arranged vertically. Using the property sheet, you can easily change the orientation to horizontal, and increase or decrease the number of members in the set.

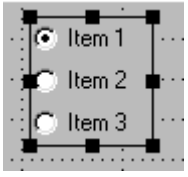
### 5.13.1 Creating a Radio-set Instance

Because the Radio–set widget is a basic 4GL object, creating one is very simple. Follow these steps:

1 ♦ Click on the Radio–set tool icon in the Object Palette:



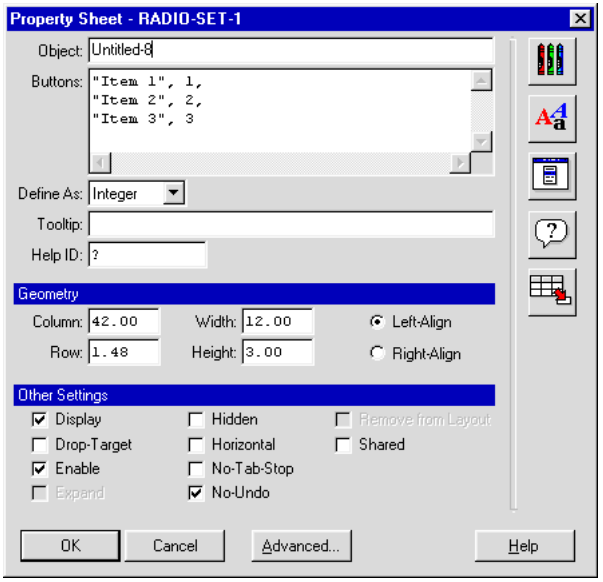
2 ♦ Move the mouse cursor over a bare spot in your workspace and click to place the new object:




3 ♦ Configure and size it. Note that you can change the spacing between the buttons by dragging the group handles.

## 5.13.2 Configuring a Radio-set Instance

To configure your Radio–set object, begin by selecting it and choosing Tools→ Property Sheet. The dialog box opens:




### Minimal Configuration

Although you can accept many default settings, you should make at least these changes:

- **Object** — Change the default identifier to one that more clearly reflects the role of this Radio set in your application.
- **Define As** — By default, the Radio–set widget represents a variable of type INTEGER. You can choose a different data type, if more appropriate. Open the drop-down list and make your choice. Note that the button definitions in the list above this drop-down change to agree with your choice here.
- **Buttons** — Each button is defined by a pair of values:
  - Label string, delimited by double quotes
  - Item value, consistent with the data type

  Use a comma as separator between the label string and the item value, and between the item value for one button and the label string for the next button.

Changing the number of buttons is very simple. To make a set with fewer buttons, delete the unwanted value pairs. To make a larger set, add value pairs.

- **Initial Value** — By default, the initial value is undefined. That might not be at all obvious because of the way a Radio–button Set displays itself. It is good programming practice always to explicitly initialize variables, but initialization is particularly important in such cases as this.

  To set the initial value, click the Advanced button. When the Advanced Properties dialog box opens, type an appropriate value into the Initial Value field.

### Help

You can add one or two types of help here:

- **Tooltip** — The Radio–button Set will display this string whenever the user allows the mouse cursor to hover over the object's frame.

- **Help ID** — If you plan to provide context-sensitive help, type in a unique integer identifier.

### Geometry

Reflects the XY origin and size, in character units, of the widget's bounding box. By default, the object's origin is considered to be the upper-left corner. You can elect to have AppBuilder consider the origin to be the upper-right corner instead, though this will not change the object's position in any way.

### Miscellaneous Properties (Other Settings)

A Radio–button Set has comparatively few properties, but you can change any that do not meet your needs.

- **Display** — Normally set. Clearing this box prevents the object from automatically displaying its initial value, if you have set an initial value. Note that this class of widget always appears to display an initial value, even if you have not set one. It is good practice always to initialize variables such as this.

- **Drop Target** — Normally cleared. Setting this box causes the Radio set to experience an event when the user drops some other object on it. You must write the code to handle the event, and the dropped object.

- **Enable** — Normally set. Clearing this box prevents the Radio set from accepting the input focus when offered. Without the input focus, the user cannot change the Radio set value.

- **Expand** — Normally cleared and not settable. Available only with Horizontal orientation. Setting this box regularizes the space between buttons by using the size of the longest label as the inter-button space. When cleared, the space between buttons varies with the length of each button's label.
- **Hidden** — Normally cleared. Setting this box causes the widget to ignore implicit requests to show itself. If you set this box, you must include explicit code in your application to make this object visible to the user.
- **Horizontal** — Normally cleared. Setting this box changes the orientation of the buttons from vertical to horizontal, and allows you to set the Expand box, if you wish.
- **No Tab Stop** — Normally cleared. Setting this box removes this object from the enclosing Frame's traversal list. An object not in the traversal list can neither gain nor lose the input focus when the user presses the **TAB** key.
- **No Undo** — Normally set. Clearing this box causes changes to the object's value to be journaled.
- **Remove from Layout** — Cleared. Cannot be set unless defining an alternate layout.
- **Shared** — Normally cleared. Setting this box makes this object's storage and value available outside the defining procedure.

### Advanced Properties

A Radio–set widget has only the standard Advanced Properties, but you can change those that do not meet your needs. See the "Advanced Properties" section in Appendix A, "Frequently Used Dialogs," for more information.

## 5.14 Selection Lists

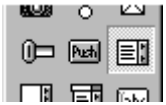
The Selection–List widget allows you to present users with a list of values represented by text strings. Selection Lists serve a similar function to Radio–button Sets and Combo Boxes: grouping and presenting a set of potential choices. Unlike those objects, Selection Lists permit multiple selection. Selection Lists (or Combo Boxes) are more appropriate than Radio–button Sets when the number of items in the set cannot be determined in advance, or where long text strings are involved.

As with Radio–button Sets, you can assign arbitrary values to the line items in the list.

### 5.14.1 Creating a Selection–List Instance

Because the Selection–List widget is a basic 4GL object, creating one is very simple. Follow these steps:

1 ♦ Click on the Selection–List tool icon in the Object Palette:

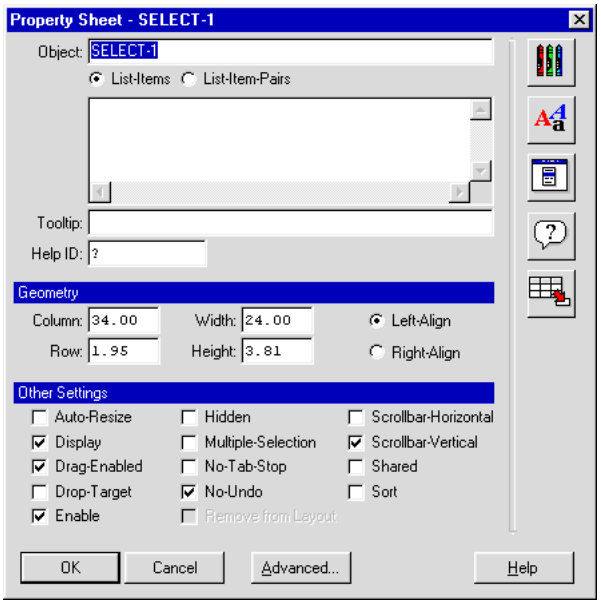


2 ♦ Move the mouse cursor over a bare spot in your workspace and click to place the new object.

3 ♦ Configure and size it.

### 5.14.2 Configuring a Selection–List Instance

To configure your Selection–List object, begin by selecting it and choosing Tools→ Property Sheet. The dialog box opens:



### Minimal Configuration

Although you can accept many default settings, you should make at least these changes:

- **Object** — Change the object identifier to one that more clearly relates to the role of this List object in your application.
- **Items versus Item Pairs** — Choose whether the Selection–List object should interpret the content you enter in the field below as independent, listable strings only, or string + value pairs. If you elect Items, the value of the variable will the selected string(s) themselves. If you elect Pairs, the value will be the value portion of the selected item(s).
- **Content** — Type in the selectable items to be displayed by the object. They can be either strings (Items) or comma-separated string + value pairs (Item Pairs). Type in only one item or item pair per line. If you are using Item Pairs, remember to separate the string from the value by a comma.

### Help

You can provide your users additional help through these settings:

- **Tooltip** — The Selection List displays this string whenever the user allows the mouse cursor to hover over the object's frame.
- **Help ID** — If you plan to provide context-sensitive help, type in a unique integer identifier.

### Geometry

Reflects the XY origin and size, in character units, of the Selection List's bounding box. By default, the object's origin is considered to be the upper-left corner. You can elect to have AppBuilder consider the origin to be the upper-right corner instead, though this will not change the object's position in any way.

### Miscellaneous Properties (Other Settings)

There are a number of other changes you can make to the Selection List's appearance and behavior, if the default settings do not meet your needs:

- **Auto-resize** — Normally cleared. Setting this box causes the Selection List to change its dimensions to keep the same number of list items in view when, for example, the font changes.
- **Display** — Normally set. Clearing this box prevents the Selection List from automatically displaying its content items during initialization.

- **Drag-enabled** — Normally set. Clearing this box prevents the user from being able to select a number of consecutive lines by holding down the **SHIFT** key and dragging the mouse pointer down the list.
- **Drop Target** — Normally cleared. Setting this item causes the Selection List to experience an event whenever the user drops another object onto it. You would have to write code to handle the event and deal with the object dropped.
- **Enable** — Normally set. Clearing this box prevents the Selection List from accepting the input focus when offered. Without the input focus, the user cannot select a listed item.
- **Hidden** — Normally cleared. Setting this box prevents the object from responding to implicit requests to display itself. You must include code to explicitly paint the object, when this box is set.
- **Multiple Selection** — Normally cleared. Setting this box allows the user to select more than one listed item.
- **No Tab Stop** — Normally cleared. Setting this box removes the object from the parent Frame's traversal list. An object not in a traversal list can neither gain nor lose focus when the user presses the **TAB** key. If the object has focus, it keeps it. If it does not have it, focus circulates among only the objects that are in the traversal list, bypassing the excluded object.
- **No Undo** — Normally set. Clearing this box causes changes to be journaled.
- **Remove from Layout** — Cleared. Cannot be set unless working on an alternate layout.
- **Scrollbar Horizontal/Vertical** — Normally Vertical only. Set or clear to change the kind of scroll bar(s) the object displays.
- **Shared** — Normally cleared. Setting this box makes the Selection List's contents and variable available to procedures other than the one in which you defined the object.
- **Sort** — Normally cleared. Setting this box presents the listed items to the user in sorted order.

### Advanced Properties

A Selection–List widget has only the standard Advanced Properties, but you can change those that do not meet your needs. See the "Advanced Properties" section in Appendix A, "Frequently Used Dialogs," for more information.

## 5.15 Slider Controls

The Slider-Control widget is a virtual knob-and-track device with integrated numeric readout. Moving the knob along the track sets the associated INTEGER value to a proportional value within the range you have defined on the track. Conversely, setting the value of the associated variable moves the knob to the corresponding position along the track, offering the user a degree of analog, at-a-glance information that the number displayed by the readout does not.

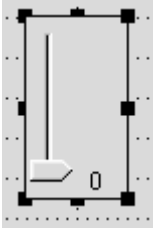
### 5.15.1 Creating a Slider-Control Instance

Because the Slider-Control widget is a basic 4GL object, creating one is quite simple. Follow these steps:

1 ♦ Click on the Slider-Control tool icon in the Object Palette:




2 ♦ Move the mouse cursor over a bare spot in your workspace and click to place the new object.

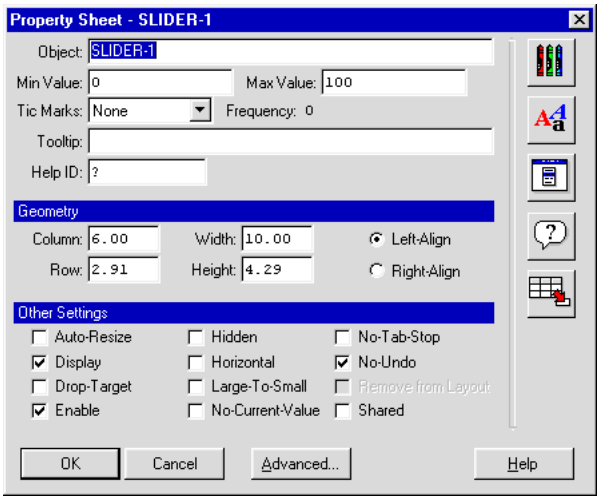



3 ♦ Configure it, and adjust its size and position appropriately.

## 5.15.2 Configuring a Slider-Control Instance

You can change some basic characteristics of the Slider-Control widget's appearance and behavior through its Property Sheet, if the default settings do not meet your needs.

To configure your Slider-Control object, begin by selecting it and choosing Tools→ Property Sheet. The dialog box opens:




### Minimal Configuration

Although you can accept many default settings, there are some you should change:

- **Object** — Change the object identifier to one that more closely matches the role of this Slider Control in your application.
- **Min/Max Values** — By default, the Slider Control represents the subrange 0–100. Unless this already meets your needs, define the subrange you want. The Slider will represent any signed 16-bit integer (–32768 to 32767).
- **Initial Value** — By default, the Slider Control is initialized to 0, or to the lowest value in the subrange if 0 is outside the subrange. To initialize to some other value, click the Advanced button and, when that dialog box opens, type in the initial value you desire.
- **Tic Marks** — By default, the scale (virtual slot) is unmarked. You can choose to add tic marks to either or both sides. If you do, you can also choose the value interval (default is 10, for a 0–100 scale).

### Help

You can provide additional help to your users with these settings:

- **Tooltip** — The Slider Control displays this string whenever the user allows the mouse cursor to hover over the object's frame.
- **Help ID** — If you plan to provide context-sensitive help, type in a unique integer identifier.

### Geometry

Reflects the XY origin and size, in character units, of the Slider Control's bounding box. By default, the object's origin is considered to be the upper-left corner. You can elect to have AppBuilder consider the origin to be the upper-right corner instead, though this will not change the object's actual position in any way.

### Miscellaneous Properties (Other Settings)

There are a number of changes you can make to the Slider Control's appearance and behavior if the default settings do not meet your needs:

- **Auto-resize** — Normally cleared. Setting this box causes the Slider Control to automatically resize itself as needed.
- **Display** — Normally set. Clearing this box prevents the Slider Control from automatically revealing its initial value. Note however that by its nature the object will always give the appearance of having been initialized.
- **Drop Target** — Normally cleared. Setting this box causes the Slider Control to experience an event whenever the user drops another object onto it. You would have to write the code to handle the event and deal with the object dropped.
- **Enable** — Normally set. Clearing this box prevents the Slider Control from accepting the input focus when offered. This would mean the user could not change the Slider Control's value.
- **Hidden** — Normally cleared. Setting this box prevents the Slider Control from responding to implicit requests to show itself. You would have to include code to explicitly request it to appear.
- **Horizontal** — Normally cleared. Setting this box causes the Slider Control to orient its track horizontally rather than vertically.

- **Large-to-small** — Normally cleared. Setting this box reverses the default direction of the values. Normally, the lowest value is at the bottom or left of the track. With this box set, the bottom or left end has the highest value, instead.
- **No Current Value** — Normally cleared. Setting this box turns off the integral numeric readout.
- **No Tab Stop** — Normally cleared. Setting this box removes this instance from the enclosing Frame's traversal list. An object not in the traversal list can neither gain nor lose focus when the user presses the **TAB** key.
- **No Undo** — Normally set. Clearing this box causes this Slider Control to journal all user changes to its value.
- **Remove from Layout** — Cleared. Cannot be set except when defining an alternate layout.
- **Shared** — Normally cleared. Setting this box makes the Slider widget and its variable available to other procedures.

### Advanced Properties

A Slider–Control widget has only the standard Advanced Properties, but you can change those that do not meet your needs. See the "Advanced Properties" section in Appendix A, "Frequently Used Dialogs," for more information.

## 5.16 Spin Controls (ActiveX)

The Spin-Control widget consists of two buttons marked with arrowheads, optionally attached to a readout field that displays an INTEGER value. You can think of the buttons as being a form of scroll bar, without the actual bar or thumb. Pressing one of the buttons causes the value in the readout fields to change by some amount you set (default 1) within the subrange –10000 to 10000. Unlike a scroll bar or a Slider Control, a Spin Control offers no analog, at-a-glance information. You might use a Spin Control in cases where you wish to allow the user to set some value that changes in integer steps other than 1.

See the "Configuring an ActiveX Object Instance" section in Chapter 8, "Other Objects," for an example of using the Spin Control.

### 5.16.1 Creating a Spin-Control Instance

Creating a Spin Control is quite simple. Follow these steps:

1 ♦ Click on the Spin-Control tool icon in the Object Palette:



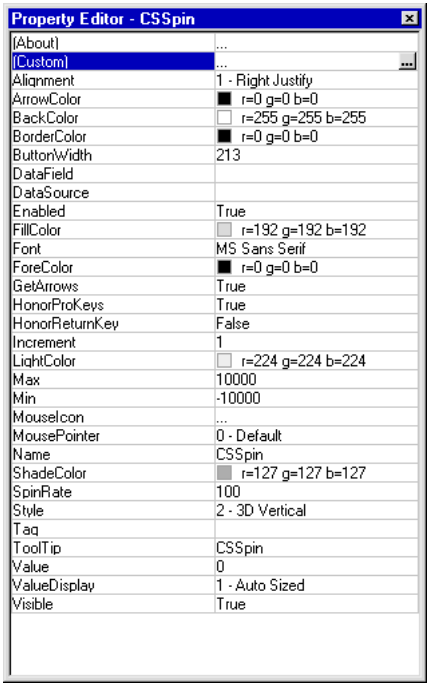
2 ♦ Move the mouse cursor over a bare spot in your workspace and click to place the new object. It will almost certainly be too large, so resize it temporarily:




3 ♦ Configure it, and make final adjustments to its size and position.

### 5.16.2 Configuring a Spin-Control Instance

The Spin Control has a number of properties you can change if the default settings do not meet your needs. Double-click the instance to open its property sheet:



### Minimal Configuration

Although you can accept many default settings, you should make at least these few changes:

- Replace the default identifier (shown in AppBuilder's main window) with one that more clearly identifies the role of this object in your application. This identifier belongs to the Control Frame interface. For more information about Control Frame objects, see the "ActiveX Control Frame Properties" section in Appendix A, "Frequently Used Dialogs."
  **Max and Min** — Normally 10000 and –10000. Set these properties to the upper and lower INTEGER bounds of the range you want the Spin Control to use. You cannot set a value smaller than –10000 or larger than 10000. Note that if you set Min to be larger than Max, the meaning of the buttons will reverse, which may confuse the user.

- **Increment** — Normally 1. Set this property to be the size of the INTEGER change to be made per button press. This is an unsigned (absolute) value.

### Optional Configuration

You may wish to make changes to some or all of these settings, or to others not discussed here:

- **Value** — Normally 0. Set this property to initialize the object; read it to detect user changes.

- **Spin Rate** — Normally 10. This property represents the number of increments per second when the user holds down one of the buttons. Thus, if the increment value is 100, and you set the Spin Rate to 12, you can expect the value to change by 1200 over a second's time, when a button is held down.

- **GetButtons** — Normally TRUE. Setting this property to FALSE prevents the object from responding to keyboard arrow-key presses. Normally, pressing one of the arrow keys has the same effect as pressing one of the Spin Control buttons.

- **ValueDisplay** — Normally AutoSized. Setting this property to None prevents the object from displaying the integrated readout. Turning off the readout in this way is useful if you wish to bind the buttons to some other object for display.

- **Style** — Normally 3D Vertical. There are four possible settings, combining Normal versus 3D, and Horizontal versus Vertical. Setting this property to Normal flattens out the 3D effect, and setting it to Horizontal changes the orientation of the buttons such that they point left and right rather than up and down.

  Combining Horizontal orientation with reversed Min and Max meanings may be appropriate in some international contexts.

- **Alignment** — Normally Right Justify. Applies to how the integrated readout displays the object's INTEGER value. Possible settings are Left Justify, Center, and Right Justify.

## 5.17 Toggle Boxes (Check Boxes)

The Toggle Box always represents two-valued, either/or information. You can visually group together many Toggle Boxes in a logical relationship that you define, but, unlike Radio Buttons, Toggle Boxes have no interdependence and are unaware of one another's settings. Setting one box has no effect on any other unless you write the code to create and manage that effect.

Radio-button sets are best for representing a group of mutually exclusive states; Check-Box sets are best for representing a group of independent states.

### 5.17.1 Creating a Toggle–Box Instance

Because Toggle Boxes are 4GL objects, creating one is quite simple. Follow these steps:

1 ♦ Click on the Check Box tool icon in the Object Palette:



2 ♦ Move the mouse cursor over a bare spot in your workspace and click to place the new object:

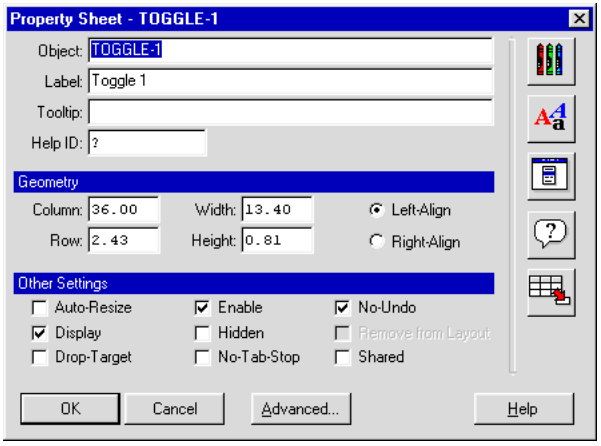



3 ♦ Configure it, and adjust its size and position appropriately.

## 5.17.2 Configuring a Toggle–Box Instance

You can change some basic characteristics of the Toggle–Box widget's appearance and behavior through its property sheet, if the default settings do not meet your needs.

To configure your Toggle–Box object, begin by selecting it and choosing Tools→ Property Sheet. The dialog box opens:




### Minimal Configuration

Although you can accept other default settings, you should make at least three changes:

- **Object** — Change the object identifier to one that better reflects the role of this Toggle Box in your application.
- **Label** — Change the label to a string that will help the user of your application understand what the Toggle Box selects.

  Note that it will be better for your users if you do not create cross-connections in their heads: the cleared state should always represent the not/off/false/negative condition, not the default condition. Initialize the box to whatever the default condition is, whether set or cleared.
- **Initial Value** — By default, the Toggle Box initializes itself to the cleared state. To change that, click the Advanced button, and when the Advanced Properties dialog box opens, type yes into the Initial Value field.

### Help

You can offer additional help to your users via these properties:

- **Tooltip** — The Check Box displays this string whenever the user allows the mouse cursor to hover over the object's frame.
- **Help ID** — If you plan to provide context-sensitive help, type in a unique integer identifier.

### Geometry

Reflects the XY origin and size, in character units, of the object's bounding box. By default, the object's origin is considered to be the upper-left corner. You can elect to have AppBuilder consider the origin to be the upper-right corner instead, though this will not change the object's position in any way.

### Miscellaneous Properties (Other Settings)

There are a number of changes you can make to the Toggle Box's appearance and behavior if the default settings do not meet your needs:

- **Auto-resize** — Normally cleared. Setting this box causes the Toggle Box to automatically resize itself as needed.
- **Display** — Normally set. Clearing this box prevents the Toggle Box from automatically revealing its initial value. Note however that it always appears to have an initial value.
- **Drop Target** — Normally cleared. Setting this box causes the Toggle Box to experience an event whenever the user drops another object onto it. You would have to write the code to handle the event and deal with the object dropped.
- **Enable** — Normally set. Clearing this box prevents the Toggle Box from accepting the input focus when offered. This would mean the user could not change the Box's value.
- **Hidden** — Normally cleared. Setting this box prevents the Toggle Box from responding to implicit requests to show itself. You would have to include code to explicitly request the object to appear.
- **No Tab Stop** — Normally cleared. Setting this box removes the Toggle Box from the enclosing Frame's traversal list. An object not in the traversal list can neither gain nor lose focus when the user presses the **TAB** key. This can confuse users.
- **No Undo** — Normally set. Clearing this box causes the Toggle Box to journal all user changes to its value.

- **Remove from Layout** — Cleared. Cannot be set except when defining an alternate layout.
- **Shared** — Normally cleared. Setting this box makes the widget and its variable available to other procedures.

### Advanced Properties

A Toggle–Box widget has only the standard Advanced Properties, but you can change those that do not meet your needs. See the "Advanced Properties" section in Appendix A, "Frequently Used Dialogs," for more information.

# 6

# Data-communication Objects

The world is increasingly a wired one, where the limitations imposed by physical distance are being overcome in part by high-speed, global communication. As more companies partner with customers on the other side of the globe, effective data communication becomes vital to staying in business.

This chapter describes the data-transformation and communication objects AppBuilder supplies for your use:

- SmartB2BObject
- XML Schema Mapping Tool
- SmartSenders and SmartReceivers
- SmartRouters
- SmartProducers and SmartConsumers

## 6.1 SmartB2BObjects

The SmartB2BObject helps you make your applications capable of business-to-business communication. It handles transformation between the Progress internal data representation and the emerging standard for inter-business communication, the eXtended Markup Language (XML). Figure 6–1 illustrates how the SmartB2BObject fits into an application that can communicate with another business over the Internet.

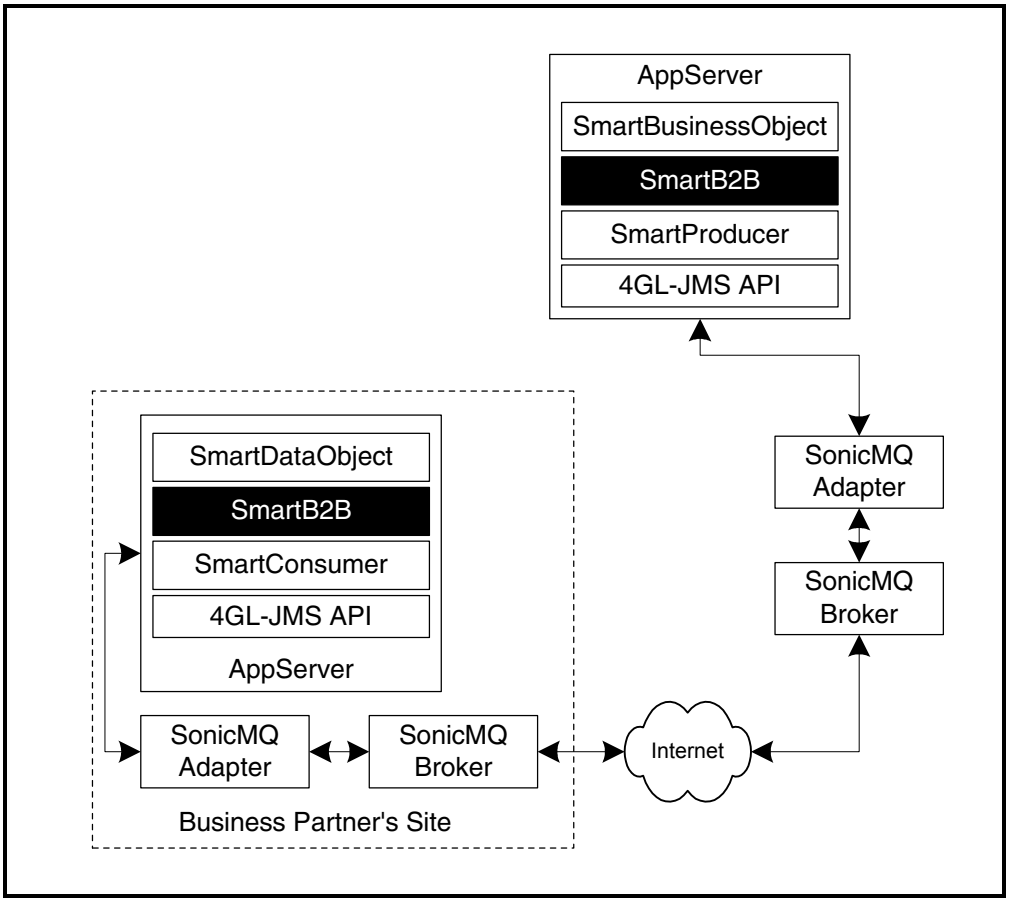



**Figure 6–1: SmartB2BObject in Inter-business Communications**

XML is related to HTML, the HyperText Markup Language of the Web. Both XML and HTML are descendents of SGML, the Standard Generalized Markup Language used in high-end publication editors.

A detailed description of XML is well beyond the scope of this document, but you can find specifications and descriptions in a number of sites on the Web. The standards body for XML, as for HTML, is the WorldWide Web Consortium (W3C), an international organization. The W3C is hosted in the Americas by the Massachusetts Institute of Technology in Cambridge, Massachusetts, USA, in Europe by the Institut National de Recherche en Informatique et en Automatique (INRIA) at several locations in France, and in Asia by Keio University at their Shonan Fujisawa Campus in Japan.

The W3C sponsors several ongoing engineering committees, including the XML Schema Working Group. The consortium's Web site has information about these working groups, including instructions for joining them if you want to help with their work. The Web site also has primary instructional materials, standards documents, pointers to tools such as schema validators, and links to other sites. Their URL is `http://www.w3.org`.

### 6.1.1 XML Schema and Mapping Files

Although two business partners might use an identical set of logical documents, their documents generally have different physical formats and map to databases that are organized differently. Rarely will two businesses use the identical format for their Purchase Orders, for example, or have identically-named tables and fields. This creates a problem for automatic handling. Although human beings are extremely good at recognizing when two different physical formats are logically equivalent, computers are not. For this reason, some way is needed to transform a logical document from the sender's local format into the receiver's local format. The most general solution to this problem is to create a shareable intermediate representation. XML, emerging as an international standard, is well-suited to that role.

XML schema are defined using a notation called "XML Data, subset reduced." A number of organizations have arisen that act as repositories for shareable schema and there are already schema defined for many common documents. You can discover some of them through the WWW Consortium Web site and others by using a Web search engine.

An XML schema for some logical unit of communication (a purchase order, an invoice, a catalog, etc.) is defined in advance and shared with business partners who will receive that kind of document. XML schema serve the same purpose as database schema—they define how raw data should be interpreted. They deal with form rather than content.

After a schema for a document has been agreed upon, each business creates an additional XML file that describes how to map between that common, shared schema and their own local data format. Typically, different mapping files will be used for inbound and outbound documents, even within a single site.

Once those mapping files have been created and debugged, communication becomes a relatively straightforward operation.

**CAUTION:** At present, there is no automatic safeguard against schema version skew. It is possible for a particular schema to be changed without notice, with an updated map file being created only at one end of the communication channel. In that case, miscommunication will certainly occur, possibly in ways that are important but hard to detect. Although you cannot eliminate this problem, you can reduce it by maintaining and checking for version information in any schema you create.

Figure 6–2 illustrates how the schema, the map file, and the data interact in inter-business messaging.

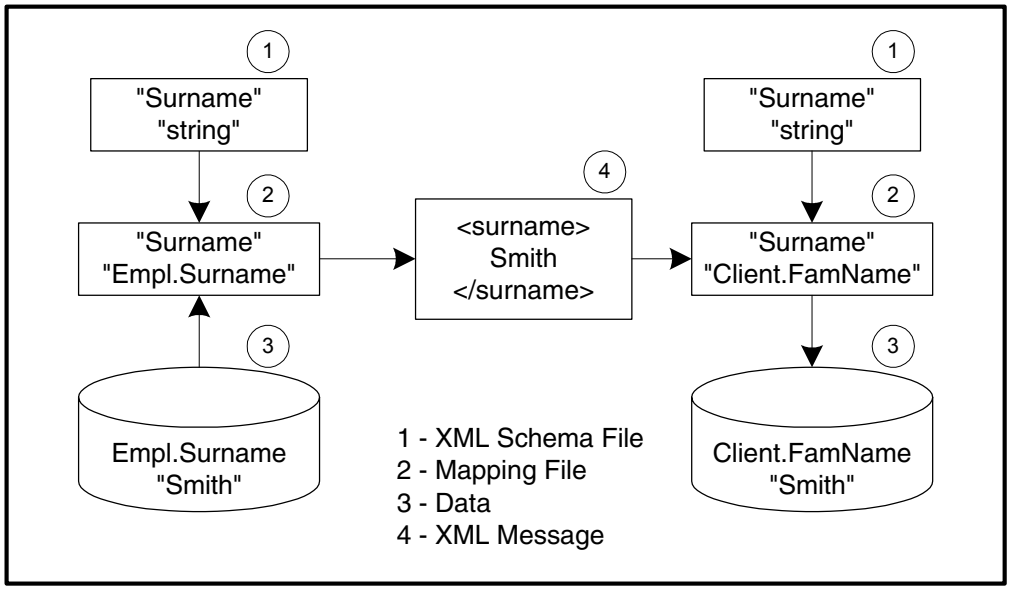



**Figure 6–2: Simple One-to-One XML Mapping in Data Communication**

On the originating side (the left side in Figure 6–2), the value for the Surname (family name) field in the Empl table is "Smith". The XML schema specifies a Surname field of type string. The map file relates the XML Surname node to the Empl.Surname field.

The SmartB2BObject would use that mapping to extract the value "Smith" from the data and insert it into the message. When the message is received (the right side in the figure), the SmartB2BObject there parses the message, encounters Surname, and checks its own map file, where it sees that Surname maps to the Client.FamName field. It then passes the value "Smith" to the SmartDataObject handling that table.

Note that neither the table nor the field need have the same name on both sides. That is the purpose of XML mapping: to make such differences irrelevant.

Figure 6–2 illustrated the simplest mapping: an XML field to a database field. Figure 6–3 illustrates a more complex possibility: deriving a value for the message by filtering local data.

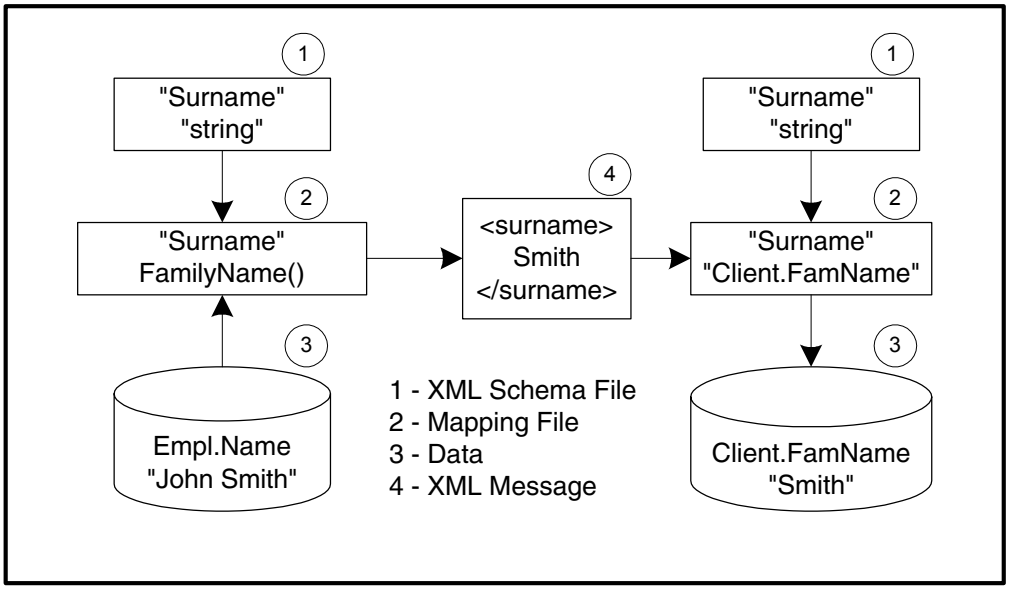



**Figure 6–3: Mapping an XML Node to a Function**

In this example, the node definition in the XML schema is the same as in Figure 6–2: Surname, defined as type string. Here, though, there is no surname field in the Empl table. The whole name is stored as a single string. The mapping in this case is to a dedicated function FamilyName() that reads the Name field and returns the surname substring.

Where does that dedicated function come from? You create it.

You can create mappable procedures and functions within the SmartB2BObject itself, or in the SmartBusinessObject or SmartDataObject that cooperates with the SmartB2BObject. In general, good programming practice suggests that you put special procedures and functions inside the object to which they are most closely related. In the illustrated case (Figure 6–3), that would be the SmartDataObject, since a FamilyName() function would be the logical equivalent of a calculated field.

### 6.1.2 Mapping Options

Mapping is very like translating from one human language to another. It can be simple or difficult depending on the complexity of the original document and your skill at preserving the underlying meaning.

**CAUTION:** No type-checking is performed at any stage of the mapping process. You yourself must ensure that the values you produce by mapping or conversion are appropriate and correct.

You have a number of mapping options to aid you in your document conversion. You can choose to map a schema node to a:

- Column being managed by a data-access object.
- Function you created in the SmartB2BObject itself.
- Procedure you created within the data-access object.
- Function you created within the data-access object.
- Whole data-access object.

These options are discussed more fully in the following sections.

### Mapping to a Column

This is the simplest and probably the most common mapping. You can use it whenever a column in the data stream corresponds directly to a schema element.

Calculated fields/columns can complicate matters. On the outbound side, mapping a calculated column is identical to mapping an ordinary column. However, since calculated fields have no physical existence in a database, you would have to write special code to make sense of an inbound mapping. Usually, calculated fields are read-only; writing them in a meaningful way requires that you handle the result as a special case, decomposing and folding it back into the original fields in some way that fits your business context.

### Mapping to a Function in the SmartB2BObject

If you need to transform or convert a column value in some general way before sending/storing it, you can create a function to do that inside the SmartB2BObject itself. For example, you might create a function to convert dates to some special format, or present an integer value in some unusual base. The Mapping Tool will display any such functions as options when you elect to map to a column.

These conversion functions always have a single parameter:

- For inbound documents, the value of the message element is passed as the parameter and the return value of the function is stored into the data field.
- For outbound documents, the situation is exactly opposite: the data field is passed as the parameter and the return value of the function is used as the message element.

### Mapping to a Procedure in the Data-access Object

When you map an XML node to a procedure, you are actually mapping to the procedure's parameter. The parameter must be an INPUT parameter for inbound documents and an OUTPUT parameter for outbound documents. It must also be the **only** parameter. You must handle all 4GL data operations inside the body of the procedure.

### Mapping to a Function in the Data-access Object

When you map to a function, the exact mapping depends on the direction:

- For inbound documents, you are mapping the schema node to an INPUT parameter. You must handle all assignments to the data field within the body of the function; the return value will be discarded.

- For outbound documents, you are mapping the return value of the function to the schema node. These functions can have no parameters. Handle all data reads within the body of the function.

### Mapping to a Whole Data-access Object.

Mapping a whole object to a schema node has a different meaning depending on whether you are creating an inbound or an outbound map.

On the outbound side, the meaning is an implied one: iteration.

Consider the subtree example shown here:




The fields custnum, name, state, and so forth have obvious mappings to fields in the Customer table. To map those fields to these nodes, you must identify as a resource the SmartDataObject that supplies records from the Customer table.

If the schema allows multiple instances of the Customer element to be generated, and you map the object itself to the Customer node here, then the SmartB2BObject treats that as a request to fetch **all** Customer records as part of building the message body. It will loop through all Customer records, extracting values for each of the mapped fields (custnum, name, and so forth) and creating XML for them.

The net result of that loop will be multiple Customer record elements, explicitly laid out in the message body. It might look something like this, if the only fields mapped were custnum and name:

```
<Customer>
   <custnum> 1 </custnum>
   <name> Smith </name>
</Customer>
<Customer>
   <custnum> 2 </custnum>
   <name> Garcia </name>
</Customer>
<Customer>
   <custnum> 3 </custnum>
   <name> Hardingfele </name>
</Customer>
```

Which means that, on the inbound side, no loop is required—simply going through the message in a linear way will traverse all included elements.

On the inbound side, the result of processing the incoming XML is typically operations on a database. When you elect to map a data object to an element, you must identify what sort of operation is desired or acceptable:

- **Update (may exist)** — Any particular record need not exist. If it does exist, it will be updated and if it does not exist, it will be created.
- **Update (must exist**) —The record must exist and will be updated as part of the message-consuming process. If the record does not exist, the data handler will report an error.
- **Create only**— The record must not already exist; it is created as part of the message-consuming process. If the record already exists, the data handler will report an error.
- **Delete** — The record must exist; it is deleted as part of the message-consuming process. If the record does not exist, the data handler will report an error.
- **Find** — This operation is used for positioning the current-record pointer in the database.

**NOTE:** It is also possible to identify the operation within a node of the message itself. Exactly how this is to be done is not well-defined and may vary across document schema. If you face this situation, you must write code to handle it as a special case.

To continue the example: if on the inbound side you map the Customer object to the Customer element with a type of Create, then every time the receiving SmartB2BObject sees the <Customer> token in the received message, it will build a Create request using field data from

<custnum>, <name>, and so forth. When it reaches the </Customer> token, it will forward the request to the local SmartDataObject that manages that table.

Note that, in this example, Customer cannot be the root node of the schema tree if you intend to generate multiple records; XML only allows one root node. If Customer were the root node, only data from the current record would be transformed into XML.

However, if Customer is not the root node of the schema tree, then **all** Customer records will be processed. If you want only a subset of all records to be used, you must take steps to ensure that only those records appear in the data stream. Consider using a SmartFilter for that purpose (see the "SmartFilters" section in Chapter 4, "Data-access Objects" for details).

**CAUTION:** If you map an object to some field/column that does not necessarily have unique values across all records, Find, Delete, and Update operations will act on the first acceptable record found. That might not be what you intend or desire.

### 6.1.3 Progress XML Schema Mapping Tool

Every time the SmartB2BObject converts some document to or from XML, it must read the logical associations from a map file. You or the user of your application must create each such map file specifically for a particular document schema and direction (inbound or outbound). This can be a non-trivial programming task. Progress Software provides an editor—the XML Mapping Tool— to make that task easier.

Figure 6–4 shows the Mapping Tool's window.

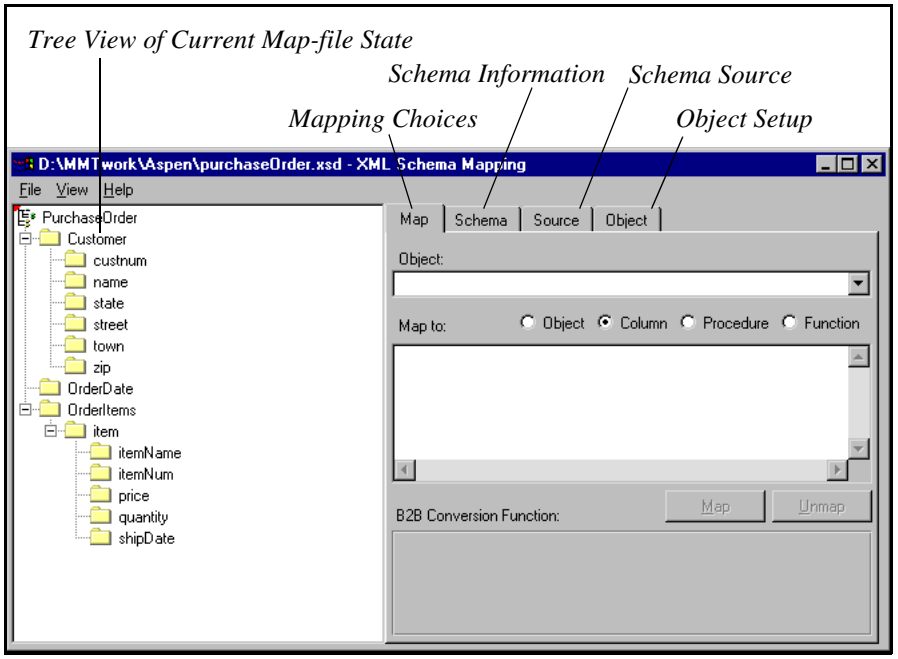



**Figure 6–4: XML Mapping Tool Window**

The Mapping Tool's window is divided into two spaces—a tree view on the left and a tab control with four tabs on the right.

The tree view displays the current state of the map file. Each line represents a mappable element of the document. As you make mapping choices, the tree view changes to reflect them. When every node displays a mapping, your map file is complete.

The four tabs on the right contain the editing interface:

- The Map tab is where you will do most of the mapping work.
- The Schema tab provides read-only information about the current schema file.
- The Source tab offers a read-only view of the raw schema source.
- The Object tab allows you to identify the local SmartObjects you will use when mapping nodes.

### 6.1.4 Starting the XML Mapping Tool

You can start the XML Mapping Tool in either of two ways:

- Click the tool icon at the end of the PRO*Tools toolbar:

  When the Mapping Tool window opens, choose File→ Open and select one of the three types of files the mapping editor understands. Those files are:

  – **Outbound map file** — Identified by the filename extension .xmp (**X**ML **M**ap for **P**roducers). This file is used to transform outgoing data from the local internal representation into XML format for transmission.

  – **Inbound map file** — Identified by the filename extension .xmc (**X**ML **M**ap for **C**onsumers). This file is used to transform incoming data from the XML representation into the local internal format for processing and storage.

  – **XML schema file** — Identified by the filename extension .xsd (**X**ML **S**chema **D**ocument). This file represents the generic, standard, nonproprietary document. It is used by all parties as the basis for creating their inbound and outbound map files. You cannot edit a schema file using the XML Mapping Tool.

- Choose File→ Open in AppBuilder and select a map or schema file. AppBuilder will automatically call the Mapping Tool on your behalf.

**NOTE:** The XML Mapping Tool only supports schema files that conform to the W3C specification. It does not attempt to handle other formats such as Microsoft's or BizTalk's. Nor does it handle the older Document Type Description (DTD) files.

### 6.1.5 Creating an XML Map File

The basic process in creating a map file is straightforward:

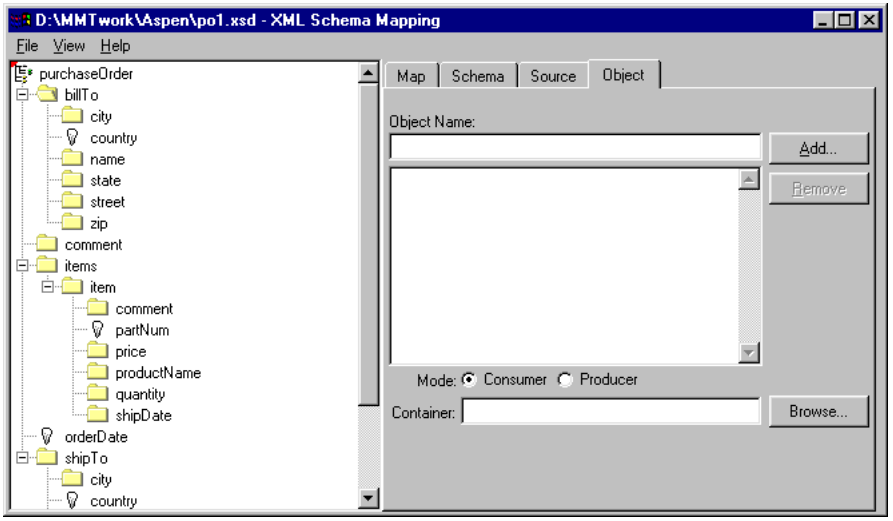
1. Start the Mapping Tool and select the SmartObjects that will handle this class of document.

2. Work through the fields—nodes— of the document in an orderly way, mapping each one to the corresponding object, database field, or transforming routine.

### Selecting SmartObjects for Mapping

The SmartB2BObject always transforms data on behalf of some other object. You must identify at least one such object—typically a SmartDataObject or SmartBusinessObject—before you can begin mapping nodes. Follow these steps:

1 ♦ Choose the Object tab. The view changes:




2 ♦ Choose the Add button. Select an object that can serve as a DATA-SOURCE or DATA-TARGET, depending on the type of map file you are creating. Its ObjectName property will appear in the fill-in and its filename will appear in the list box.

**NOTE:** You can edit any ObjectName here, but no two listed objects can have the same ObjectName. If you change an ObjectName, the new name must be locally unique (in other words, unique within the list you are creating here).

3 ♦ Repeat Step 2 until you have identified all the objects you will use.

**NOTE:** If you intend to use conversion functions that reside in the SmartB2BObject, you must Add that object to this list as well.

If you make a mistake when adding an object, select it in the list and choose the Remove button. Note that you can remove an object from the list at any time. Even if you have already mapped nodes to the object, the mappings will go away cleanly if you remove the object name from the list.

4 ♦ Select the mode of this map file: inbound (consumer) or outbound (producer).

5 ♦ Choose the Browse button and identify the SmartContainer that holds the SmartB2BObject that will use this map file. You can omit this step if this is the only document type your application will produce/consume.

6 ♦ Choose File→ Save to save your work.

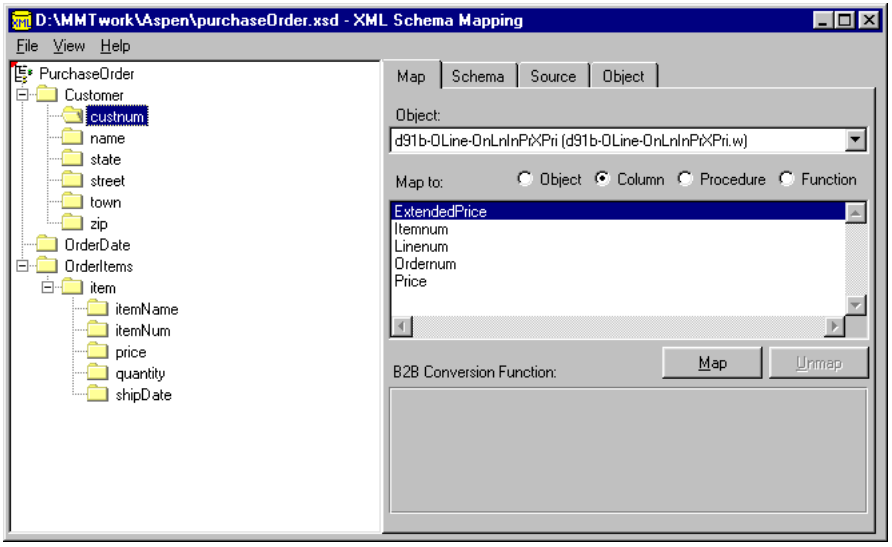
You are now ready to begin the mapping process itself.

### Mapping Schema Nodes

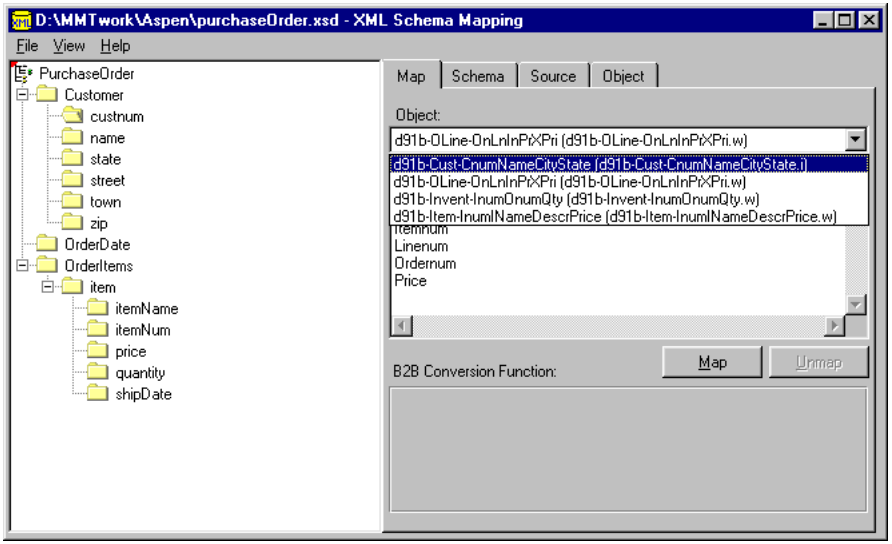
As the SmartB2BObject parses some logical document, it consults the map file you created for that document type. That map file must contain a mapping association for every element of the document. As the SmartB2BObject identifies an element in the document, it creates a transformation based on the association you supplied.

To create an association, follow these steps:

1 ♦ Select a node in the tree view. Click the Map tab, if necessary:



2 ♦ Pull down the Object list and select the object that will supply the association for this node:

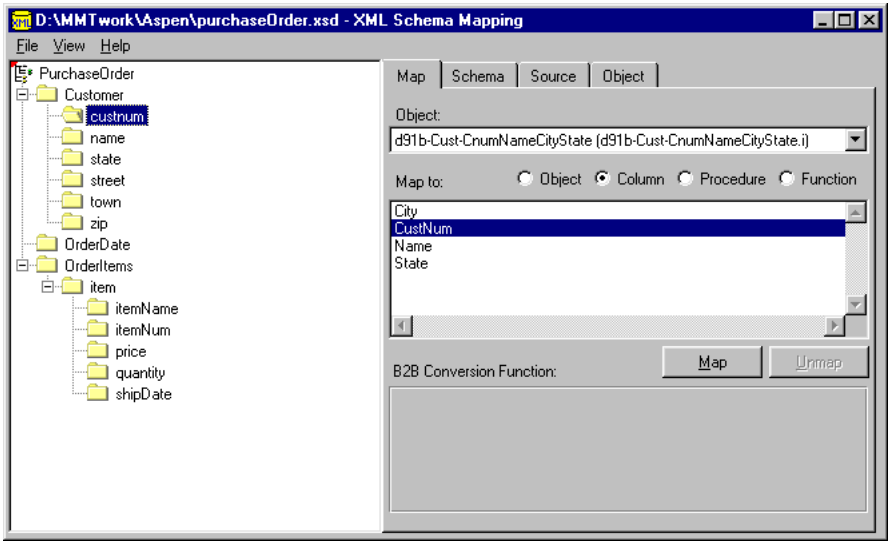



3 ♦ Select the appropriate radio button to refine the association. You can choose to map the node to:

- The whole object, if you intend to loop through multiple records (outbound side) or to CREATE, FIND, or DELETE the node, or UPDATE it or its children (inbound)
- Some particular field supplied by the object, if the node does not have children
- Some procedure or function, if the node does not have children

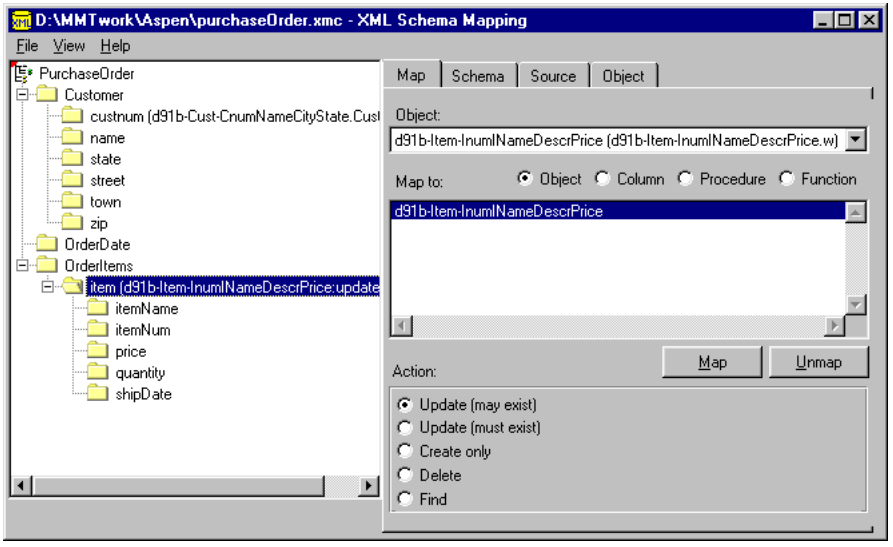
See the "Mapping Options" section for more information.

4 ♦ The display changes to reflect your choice:

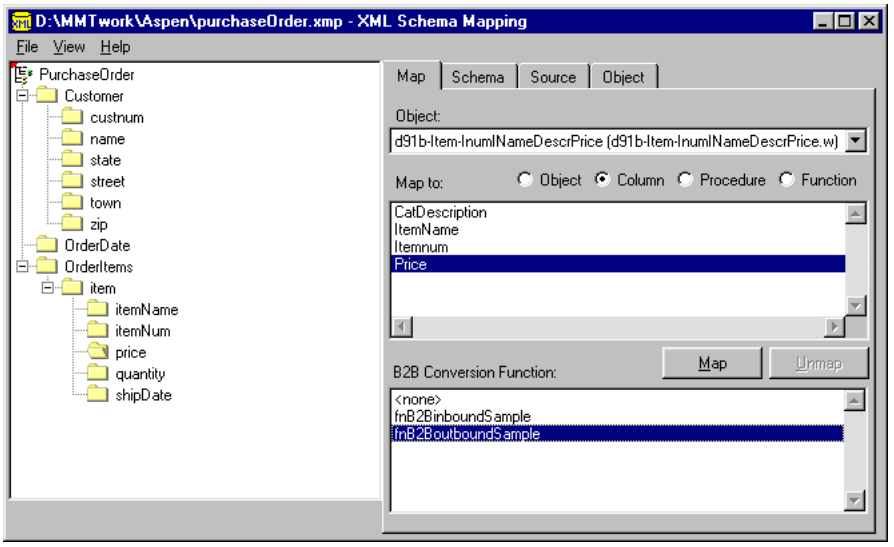



Complete the association by selecting the exact mapping ( see the "Mapping Options" section for more information):

- If mapping the whole node in an inbound document, select the action to be performed (Update–may exist, Update–must exist, Create only, Delete, Find):



- If mapping to or from a column that you will filter through a SmartB2BObject function, select the column identifier before you select the function identifier:




- If mapping to a procedure, or to a function not in the SmartB2BObject, select it from the list that appears after you choose the radio button.

### 6.1.6 Creating a Master SmartB2BObject

The SmartB2BObject is supplied for your use in two forms: a template for creating custom master objects and a precompiled master from which you can create generic instances. Unlike other template-based SmartObjects, creating a custom master SmartB2BObject does not involve a Wizard.

To create a new master object, follow these steps:

1 ♦ Click the SmartB2BObject tool icon in the Object Palette:



2 ♦ When the Choose dialog box opens, choose New.

**NOTE:** You could also start the process by choosing File→ New and selecting the SmartB2BObject item from the list, or by right-clicking the icon in the Palette and selecting New SmartB2BObject from the menu that appears.

3 ♦ The new SmartB2BObject master appears immediately. Select it, open the Section Editor, and add all the code needed to complete this custom master.

4 ♦ Choose File→ Save and give the new master a descriptive filename. Note that SmartB2BObject filenames conventionally begin with `sbb`.

The new master object is now available for your use.

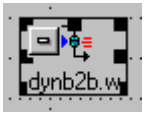
### 6.1.7 Creating and Placing a SmartB2BObject Instance

Creating a SmartB2BObject instance is much like creating an instance of any other SmartObject. To add a SmartB2BObject instance to your organizer workspace, follow these steps:

1 ♦ Click the SmartB2BObject tool icon in the Object Palette. When the Choose dialog box opens, select the appropriate master and click OK.

**NOTE:** To select the dynamic object, right-click the tool icon instead. The dynamic object is listed last in the menu that appears.

2 ♦ The SmartB2BObject has no run-time representation, so you can position it anywhere in the workspace. For your future convenience, consider selecting a spot that will otherwise remain empty. When you have selected a spot, click to release. The design-time representation appears:

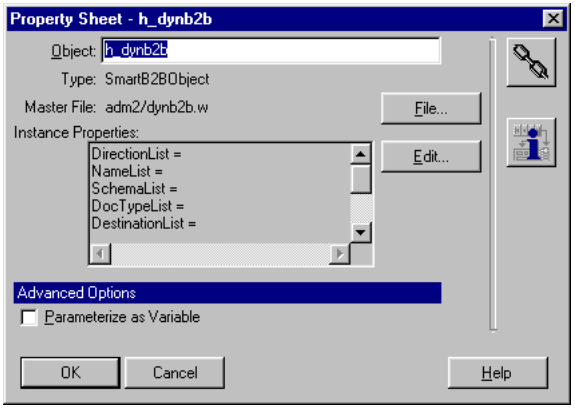



3 ♦ AppBuilder opens an Advisor window if it finds opportunities to create SmartLinks to other objects. Accept the Advisor's recommendations, if they seem appropriate.

4 ♦ Configure the instance. Save your work.

### 6.1.8 Configuring a SmartB2BObject Instance

If you have already created the map file, configuring a SmartB2BObject instance is straightforward. To begin the configuration process, click the instance's menu button and choose Properties. This dialog box appears:




Then follow these steps:

1 ♦ Change the Object identifier to more clearly reveal the role of this object in your application.

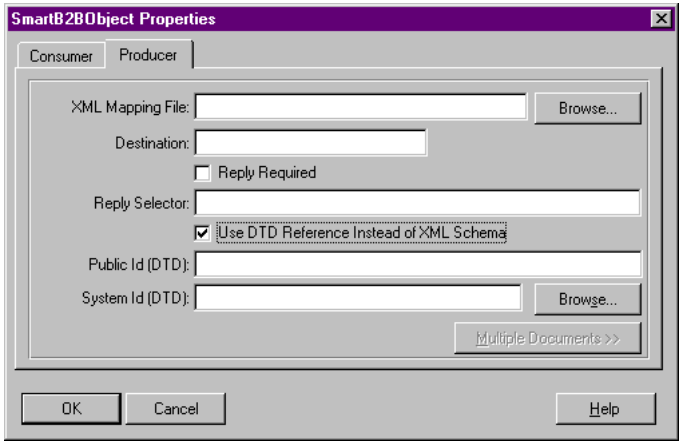
2 ♦ If you wish to use this object as a placeholder, set the Parameterize as Variable check box. For information about using instances as placeholders, see the "Creating a SmartObject Placeholder" in Appendix A, "Frequently Used Dialogs."

3 ♦ Choose OK and save your work.

You are now ready to configure the instance properties. The SmartB2BObject's instance properties vary depending on whether the object will process incoming or outgoing documents. In either case, begin by clicking the menu button and choosing Instance Properties from the menu that opens.

### Outgoing Instance Properties

If you have already linked this object to a SmartProducer, the dialog box should open with the Producer tab selected. If not, select it now:




Then follow these steps:

1 ♦ At a minimum, configure these fields to use XML:

- **XML Mapping File** — Type in the path to the file that this object will use when transforming documents.
- **Destination** — The logical identifier of the outbound document's destination. At present this must be a Queue or Topic served by the Progress SonicMQ system.

If you prefer to use a DTD (Document Type Definition) file, set the Use DTD checkbox. This will enable the Public ID and System ID fields. Fill them in.

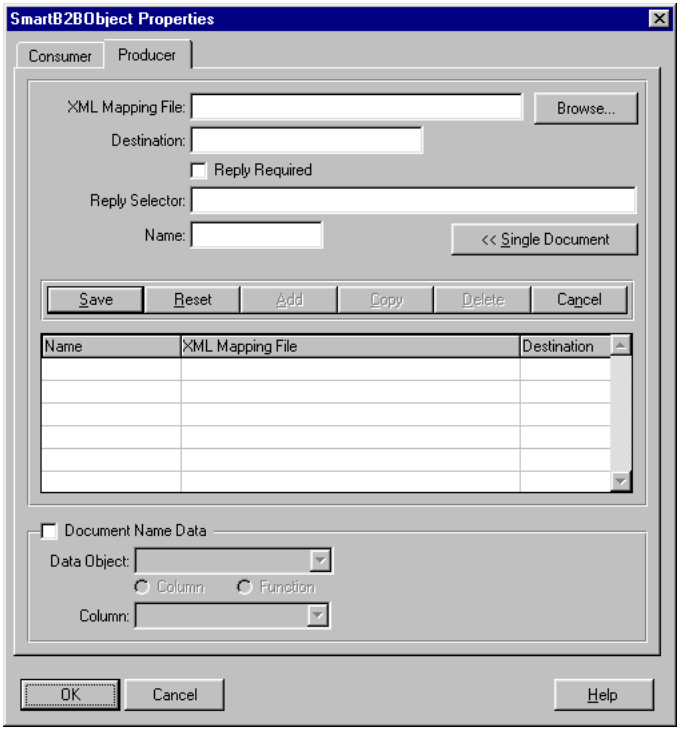
- **Public ID (DTD)** — (Optional) Enter the global public identifier of the DTD document.
- **System ID (DTD)** — (Mandatory) Enter the path to the actual DTD file. Click the Browse button if you wish to search for the file.

2 ♦ Examine these fields and change those that do not meet your needs:

- **Reply Required** — Defaults to cleared. Setting this check box will notify recipients that you wish a reply sent to you whenever they receive a document of this type. You must add code to handle these replies.
- **Reply Selector** — If you wish to filter incoming replies according to values in reply-header fields or properties, enter the expression here. The SmartProducer

object will perform the test on incoming replies and discard from the inbound Queue/Topic any messages that fail the test. Example: `company_name = "Smith" AND priority > 4`. A reply from the Smith company with priority of 4 or less would be discarded without notice. The syntax is a subset of the syntax defined in SQL-92.

3 ♦ If this SmartB2BObject will create different documents from the same data stream, or send to different real-world or JMS destinations, choose the Multiple Producers pushbutton. Additional fields come into view:




4 ♦ For each document type/destination pair, fill in the fields identified in Step 1 and Step 2.

5 ♦ To specify where to find the name reference at run time, set the Document Name Data check box. Three additional fields activate themselves. Change their values as appropriate:

- **Data Object** — Select the SmartDataObject or other data-handling object that handles the data for the new document type.
- **Column, Function** — Defaults to Column. Selecting Function means that the Document/Direction Name value is returned by a function in the data object rather than by one of its columns.
- **Column/Function** — [The label on this fill-in reflects the current radio-button selection.] Select the column or function in the data object that will return the Document/Direction Name.

6 ♦ Choose the Save button to add the new document information to the list in the browser widget.

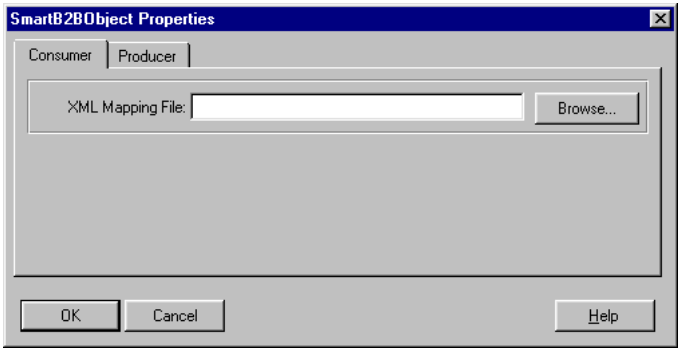
**NOTE:** If you decide not to add this entry, choose Cancel instead of Save. If you decide you made a mistake in saving an entry, choose the Delete button.

7 ♦ To add an additional entry, click the Add button and repeat Step 4 through Step 6.

8 ♦ When you have added all the document types/destinations you desire, save your work.

### Incoming Instance Properties

If you have already linked this object to a SmartConsumer, the dialog box should open with the Consumer tab selected. If not, select it now:




To configure this object for consuming, follow these two steps:

1 ♦ Identify the mapping file.

2 ♦ Save your work.

## 6.2 SmartSender and SmartReceiver Objects

The SmartB2BObject is designed to transform the representation of data between the internal 4GL format and industry-standard XML. But what if you need no such transformation? What if you only want to send, for example, some unstructured block of ordinary text?

You can use the SmartSender (or, on the inbound side, the SmartReceiver) in place of the SmartB2BObject if you do not need the SmartB2BObject's special capabilities. The capabilities of the SmartSender and SmartReceiver are whatever you choose to make them. Figure 6–5 illustrates where the objects fit in with other SmartObjects.

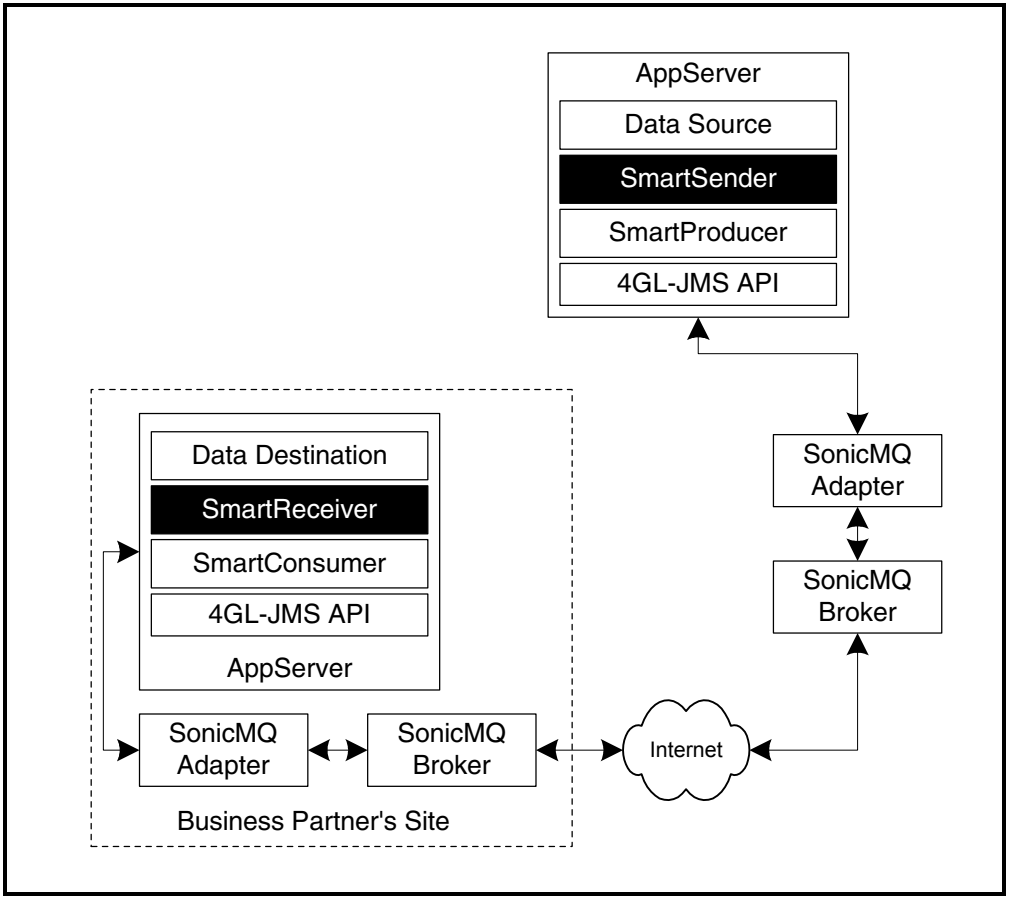



**Figure 6–5: The SmartSender and SmartReceiver in B2B Communication**

The SmartSender and SmartReceiver objects are supplied for your use as templates with stubbed handler routines. After writing 4GL procedures to complete the handler stubs, you can connect these objects to the Smart message-interface objects in exactly the same way you would connect a SmartB2BObject. The SmartSender will then perform any transformation you have included in the handler routines, package the content, and call the SmartProducer to send it off.

Similarly, the SmartReceiver will accept an inbound message from the SmartConsumer, perform the handling process you define, and pass the result on to some other SmartObject for storage or further handling.

Because you provide the handling logic, you can make these objects as simple or complex in their behavior as you desire.

You could handle plain, unformatted text by making your handler routines mere pass-throughs, their only function being to copy the unaltered text into or out of the message body. Or, you could implement special encryption/decryption procedures, graphics reformatting, or even your own special-purpose XML logic. You can implement any handling you like.

What you do with a SmartSender/SmartReceiver is entirely up to you.

### 6.2.1 Creating and Completing a SmartSender Master

To create a new SmartSender master, follow these steps:

1 ♦ Click the SmartSender tool icon on AppBuilder's Palette:



2 ♦ When the Choose dialog box opens, click the New button. After a moment, a new SmartSender object appears:



3 ♦ Open the Section Editor and add handler code to complete the `sendHandler` and `receiveReplyHandler` procedures. Use calls from the 4GL–JMS API.

The `sendHander` procedure deals with your outbound messages. Use it to set properties and create message bodies. See the "Example of sendHandler Code" section.

The `receiveReplyHandler` procedure deals with incoming replies. Use it to extract properties and evaluate message bodies. See the "Example of receiveReplyHandler Code" section.

4 ♦ Save the completed master using a descriptive filename. Note that SmartSender filenames conventionally begin with `sms`.

Creating a SmartReceiver master involves comparable steps, though the names of the handler routines are `receiveHandler` and `sendReplyHandler`, respectively. See the "Example of receiveHandler Code" and "Example of sendReplyHandler Code" sections.

### 6.2.2 Placing and Configuring a SmartSender Instance

Once you have created a master object, placing an instance is a straightforward process. Follow these steps:

1 ♦ Click the SmartSender tool icon on AppBuilder's Palette:



2 ♦ When the Choose dialog box opens, select the master object and click OK. Move your mouse cursor over your workspace and click to release the object.

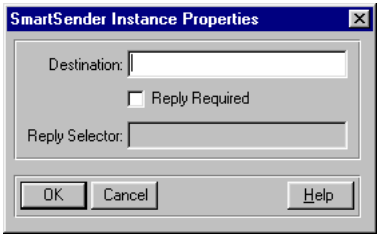
3 ♦ Save your work.

You are now ready to configure the instance. Follow these steps:

1 ♦ Click the instance's menu button and choose Properties. When the standard SmartObject Property Sheet opens, change the Object identifier (the instance handle) to more clearly reflect the role of this object in your application.

2 ♦ If you wish to use this instance as a placeholder, set the Parameterize as Variable check box. For information about placeholders, see the "Creating a SmartObject Placeholder" in Appendix A, "Frequently Used Dialogs," and the *Progress ADM 2 Guide*.

3 ♦ Click OK to close the Property Sheet. Click the menu button again and choose Instance Properties. The dialog box opens:




4 ♦ Set the Destination property to the identifier or the SonicMQ Topic (Publish-and-Subscribe) or Queue (Point-to-Point) to which your message is to be sent.

You can choose to set two other properties, if you wish:

- **Reply Required** — Defaults to cleared. Setting this check box notifies the recipient that you wish a reply to your message. Your `receiveReplyHandler` routine must deal with any such reply.
- **Reply Selector** — If you wish to filter incoming replies according to values in reply-header fields or properties, enter the expression here. The SmartConsumer object will perform the test on incoming replies and discard from the inbound queue/topic any messages that fail the test. Example: `company_name = "ABC" AND priority >= 4`. Any reply received from company ABC with a priority of 3 or less will be discarded without notice. The syntax is a subset of the syntax defined in SQL-92.

5 ♦ Click OK to close the dialog box. Save your work.

Placing and configuring a SmartReceiver instance involves comparable steps. The major difference is that the SmartReceiver has no Instance Properties dialog box.

## 6.3 SmartRouters

The SmartRouter object is designed to distribute incoming documents to the appropriate SmartB2BObject for transformation. It is only required in applications that can handle more than one document type. Figure 6–6 illustrates where the SmartRouter fits into the document-handling process.

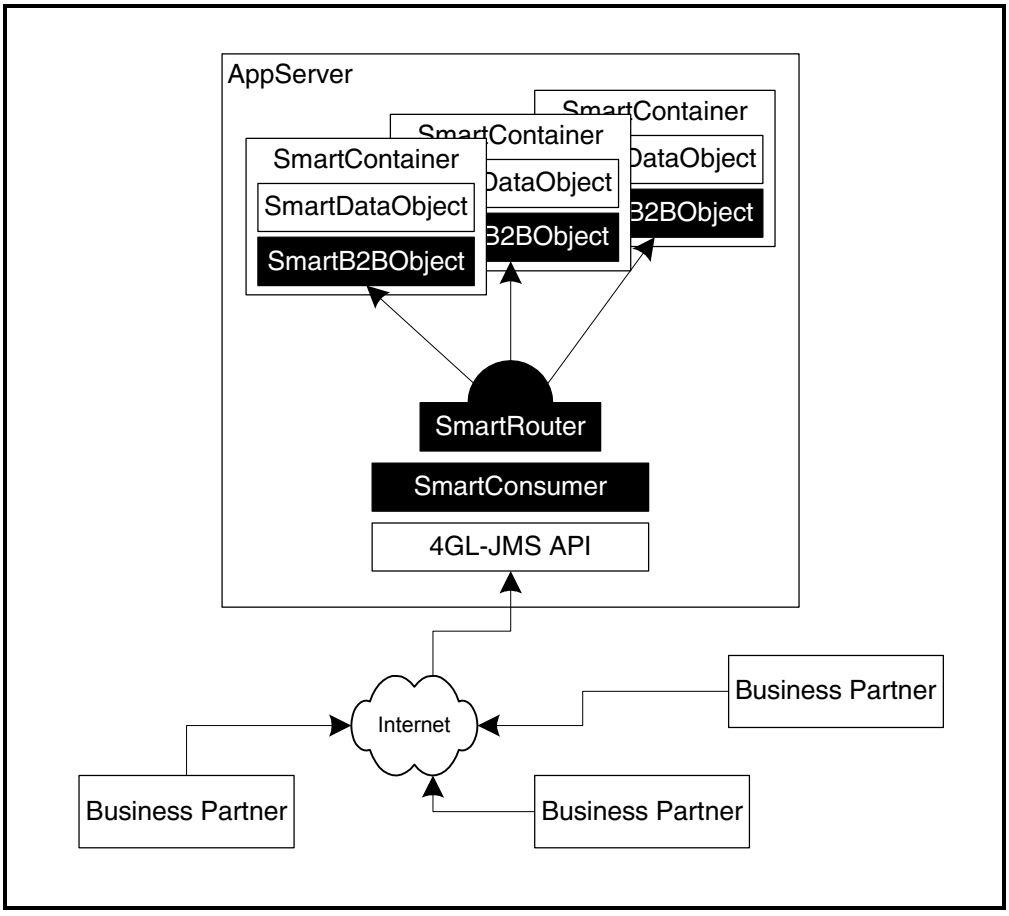



**Figure 6–6: SmartRouter Distributing Incoming Documents**

### 6.3.1 Handling Multiple Document Types

You can create a complete message-handling system using very few SmartObjects, if your needs are uncomplicated. Figure 6–7 gives a schematic view of such a system.

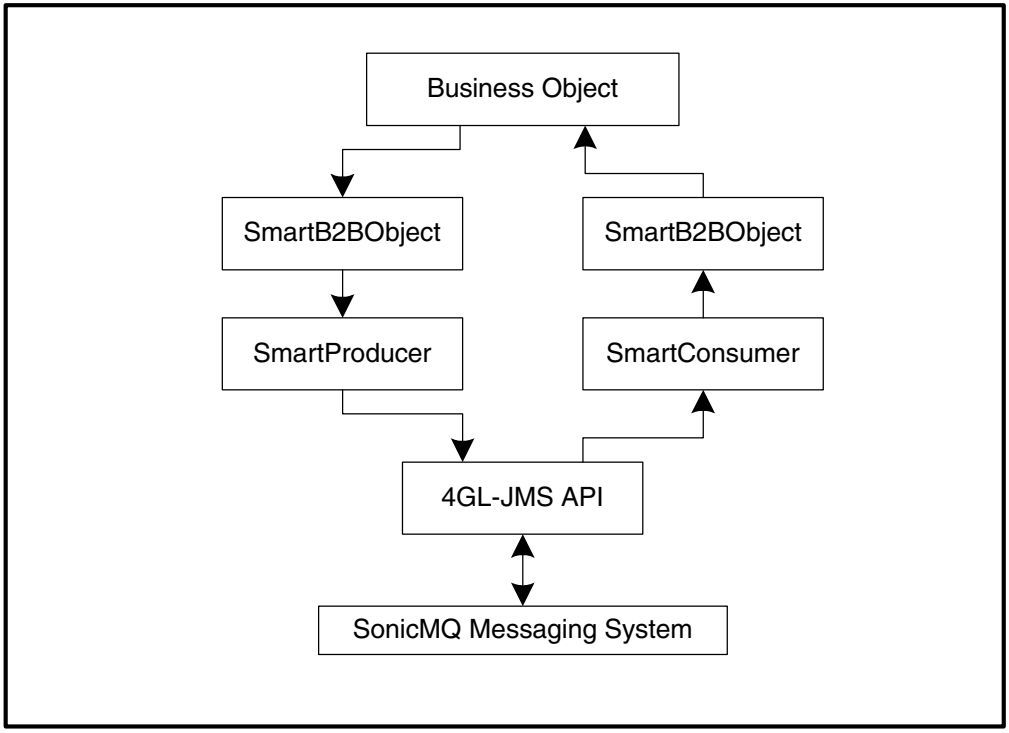



**Figure 6–7: Basic Message-handling Blocks**

The business object would typically be a SmartDataObject or SmartBusinessObject. The SmartB2BObjects could also be a SmartSender/SmartReceiver pair. Regardless, the basic layout is very straightforward when all inbound or outbound messages are handled alike.

In some business situations, however, you might have to handle several different incoming logical-document types—in Figure 6–8, Purchase Order, Sales Enquiry, and Customer Support Form—all packaged identically as XML messages. Since at present each document type is handled by a dedicated SmartB2BObject instance, you would need a way to route them appropriately.

The SmartRouter object provides exactly that service.

Figure 6–8 illustrates how the SmartRouter relates to the SmartConsumer and SmartB2B objects.

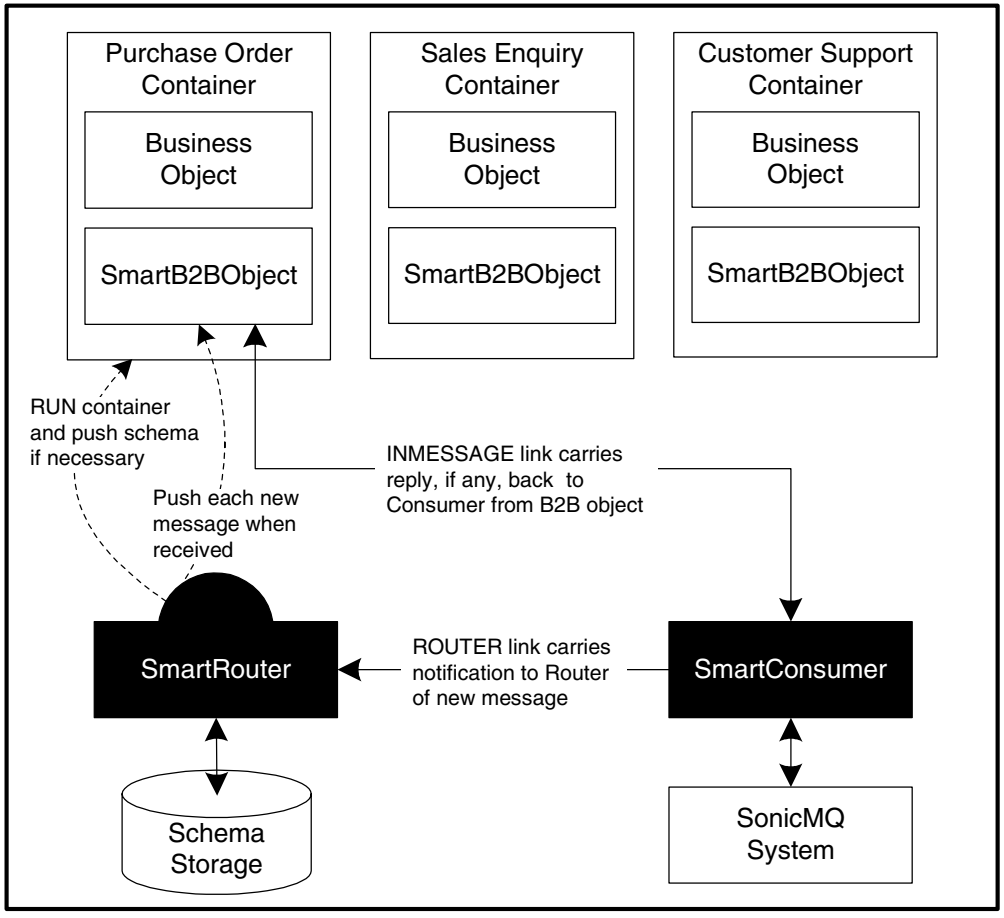



**Figure 6–8: Message Routing Using SmartRouter**

When a new message comes in, the SmartRouter determines which SmartB2BObject should do the transformation and starts that container, if necessary. It then identifies the SmartB2BObject to the SmartConsumer and hands off the message file to the SmartB2BObject. After that, it has nothing more to do with that message. The SmartConsumer and SmartB2BObject communicate directly from that point onward, as they do when a SmartRouter is not involved.

### 6.3.2 Callable Document-handlers

The major difference between using and not using a SmartRouter is in the packaging. As Figure 6–8 illustrates, the SmartB2BObject/SmartDataObject pairs that handle a particular document type are packaged separately. Similarly, a SmartRouter and SmartConsumer can be packaged together, if desired.

The package used in each case is the Simple SmartContainer.

The Simple SmartContainer is a specialized organizer object. It has no run-time representation. You would typically use it only when creating packages such as those illustrated in Figure 6–8. Being able to package components without a user interface is particularly useful when your message-consuming process is to run in batch mode, without human intervention.

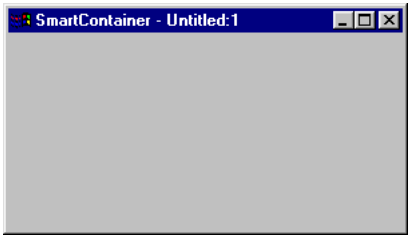
### 6.3.3 Creating a Document-handling Package

Creating a unit to handle inbound documents is a straightforward process. Follow these steps:

1 ♦ Create a SmartB2BObject—the SmartRouter cannot communicate with a SmartReceiver—to transform the object. See the "SmartB2BObjects" section for details. Save the object.

2 ♦ Create a data handler—a SmartDataObject or SmartBusinessObject—to process the transformed document. See the *Progress ADM 2 Guide* for information about the SmartDataObject, or the "SmartBusinessObject" section for details about that object. Save the object.

3 ♦ Choose File→ New and select the Simple SmartContainer. A workspace of that type appears:




4 ♦ Place and configure the transformer and data handler objects that you created. No advisor will appear to offer to create SmartLinks between these objects: they do not communicate using the SmartLink mechanism.

5 ♦ Save your work. Repeat this process until you have created a handler for each incoming document type.

### 6.3.4 Creating a Document-routing Package

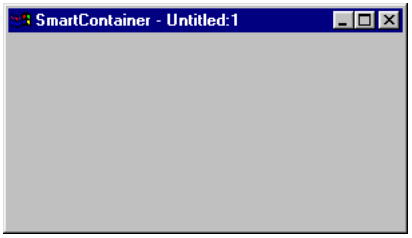
If your customer's business practices allow, you can create a system that processes incoming documents in batch mode, without human intervention.

To do this, you would create separate containers to perform the document routing and the document handling. At run time, you start the routing package in batch mode and allow it to call the document-handling packages as documents arrive for processing. For information about creating document-handling packages, see the "Creating a Document-handling Package" section.

This example assumes that you will use the invisible Simple SmartContainer as the organizer. If you require a user interface, use the SmartWindow instead.

To create a document-routing package, follow these steps:

1 ♦ Choose File→ New and select the Simple SmartContainer. A workspace of that type appears:



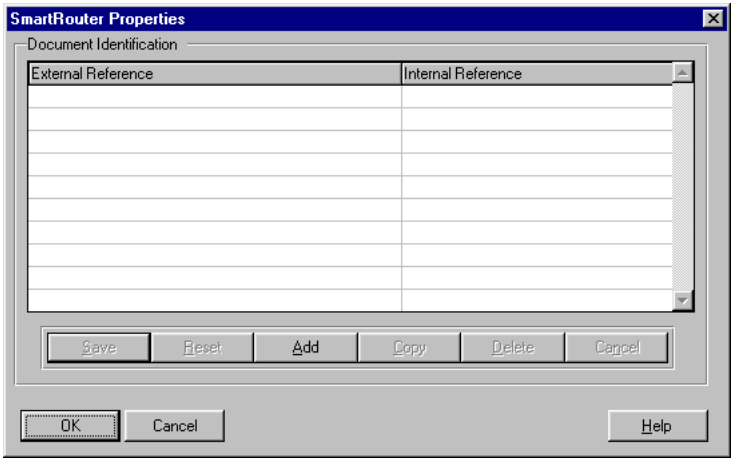

2 ♦ Place a SmartConsumer object and configure it to monitor the appropriate inbound Queue or Topic. See the "Creating and Placing a SmartConsumer Instance" and "Configuring a SmartConsumer Instance" sections for details.

3 ♦ Click the SmartRouter tool icon:



4 ♦ Place the instance in the workspace. An Advisor will appear and offer to create a SmartLink of type ROUTER from the SmartConsumer. Accept the offer.

5 ♦ Click the menu button and choose Instance Properties. The dialog box opens:




6 ♦ Click the Add button. This makes the External Reference field editable. Enter the URL of the schema file for some document type this Router must recognize. Note that the filename of the schema must be the same locally as it appears in the URL, though the paths to them will probably be very different.

7 ♦ Click the corresponding Internal Reference field to activate it. Enter the PROPATH-relative path to the corresponding map file. Thus, if the full path is `d:\abc\def\mapfile.xmc` and PROPATH contains `d:\abc`, you should enter `def\mapfile.xmc` here.

8 ♦ If you are satisfied with this entry, click the Save button to write the entry out to disk. Otherwise, click Reset to clear the fields, or Cancel to clear the fields and leave edit mode.

9 ♦ Repeat Step 6 through Step 8 until you have created entries for each document type this Router instance must recognize. Click OK to close this dialog box. Now, when run, this Router will be able to recognize and appropriately distribute those document types.

10 ♦ Save the package using a distinctive filename. Note that filenames for Simple SmartContainers conventionally begin with `c`.

## 6.4 SmartProducers and SmartConsumers

Progress Software provides you with two message-handling objects—the SmartProducer and SmartConsumer. The SmartProducer sends messages and the SmartConsumer receives them. With them, your ADM-compliant application can use the reliable Progress SonicMQ message service for communication within and between businesses. Figure 6–9 illustrates the place of the SmartProducer and SmartConsumer objects in the message-handling process.

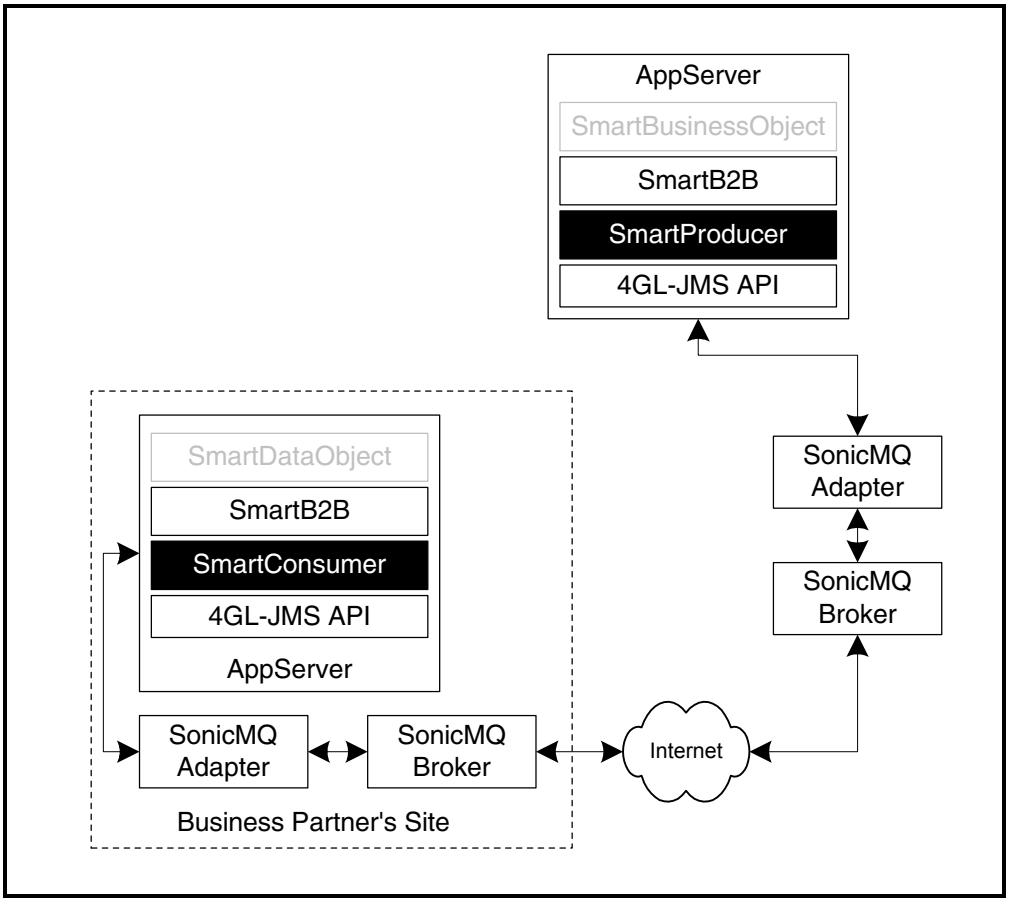



**Figure 6–9: The Smart Message-interface Objects**

SonicMQ is Progress Software Corporation's implementation of the industry-standard Java Message Service (JMS). The SmartProducer and SmartConsumer communicate with the SonicMQ broker through the 4GL–JMS API.

### 6.4.1 Creating and Placing a SmartProducer Instance

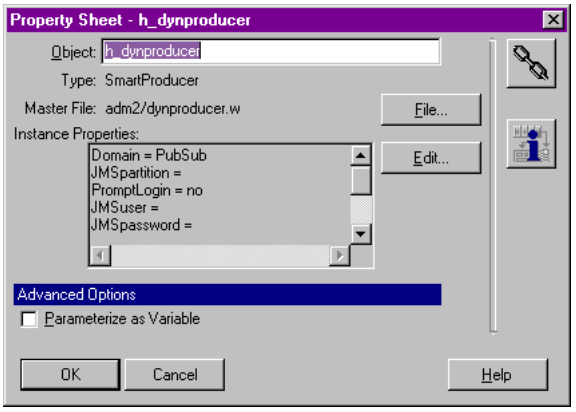
The SmartProducer is supplied as a precompiled master. Creating and placing an instance is a straightforward process. Follow these steps:

1 ♦ Click the SmartProducer tool icon in the Object Palette:



2 ♦ Position the mouse cursor over an empty spot in your workspace and click to place the object. If you have one or more SmartB2BObjects or SmartSenders in place, AppBuilder opens an Advisor dialog box and offers to create OUTMESSAGE SmartLinks. Examine the offers and accept those that are appropriate.

3 ♦ Click the object's menu button and choose Properties. The Property Sheet dialog box opens:



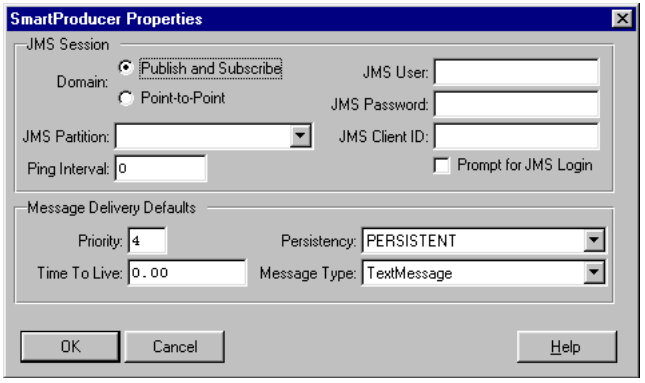

4 ♦ Change the Object identifier (instance handle) to one that more accurately describes the role of this SmartProducer in your application.

5 ♦ Configure the instance. Save your work.

### 6.4.2 Configuring a SmartProducer Instance

There are only a limited number of instance properties you need to configure. Follow these steps:

1 ♦ Begin by clicking the instance's menu button and choosing Instance Properties. The SmartProducer instance properties dialog box opens:




2 ♦ Change the default values that do not meet your needs:

- **Domain** — Defaults to Publish-and-Subscribe. Setting this value to Point-to-Point creates a single copy of the message. Although a number of entities might have access to the message, the first entity to read it removes it from the queue. With the Publish-and-Subscribe mechanism, a single copy of the message can be read by all subscribing entities. This allows implementation of a form of multicasting.

  You cannot modify the Domain value at run time.

- **JMS Partition** — Select a JMS Partition from the dropdown list. If there are no JMS Partitions listed, you can define as many as you like using the Service Parameter Maintenance Tool, available on the PRO*Tools toolbar. For information about how to use that tool, see the online help.

- **JMS User** — Set this field to identify the user of the JMS Broker connection.

- **JMS Password** — Set this field to be the password that corresponds to the JMS User identifier.

- **JMS Client ID** — Set this field to a value unique to this client. This field is used to resolve ambiguities in cases where multiple clients might be using the same subscription and user names.

**NOTE:** If you do not know the User, Password, and Client ID values at design time, you can check the Prompt-for-JMS-Login box and the values will be collected at run time.

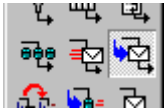
- **Ping Interval** — Defaults to 0 (pinging off). Setting this field to some integer value (representing seconds) causes the connection to the SonicMQ broker to be tested at that rate for the duration of the JMS session. It is good practice not to ping frequently, since pinging decreases overall system performance.
- **Prompt for JMS Login** — Defaults to cleared. Setting this check box causes the messaging object to prompt the user for login.
- **Priority** — Defaults to 4. The value is read whenever a new message is sent. It establishes the priority for that message in the range 0 (lowest) to 9 (highest). You can modify the value at run time under program control.
- **Time To Live** — Defaults to 0 (no expiration). This field is evaluated for every new message. It establishes the length of time in milliseconds that the message can remain unread before being discarded as stale. You can modify the value at run time under program control.
- **Persistency** — Defaults to PERSISTENT (message retained in the broker's database). Other values supported are:
  - NON_PERSISTENT (message not retained)
  - NON_PERSISTENT_ASYNC
  - UNKNOWN
- **Message Type** — Defaults to TextMessage. Other types supported are:
  - **BytesMessage** — A stream of raw (uninterpreted) bytes. This is the default type when connecting to a SmartB2BObject; most other types require extra code.
  - **HeaderMessage** — Known simply as type Message within the SonicMQ system. Use for shutdown messages.
  - **XMLMessage** — A string representing an XML tree that can be parsed as a document. **Do not use this type** when connected to a SmartB2BObject.

- **MapMessage** — A set of name/value pairs, where names are of type string and values are Java primitives. You can read the entries in sequence or randomly by reference to the name.
- **StreamMessage** — A stream of Java primitives, read sequentially.

### 6.4.3 Creating and Placing a SmartConsumer Instance

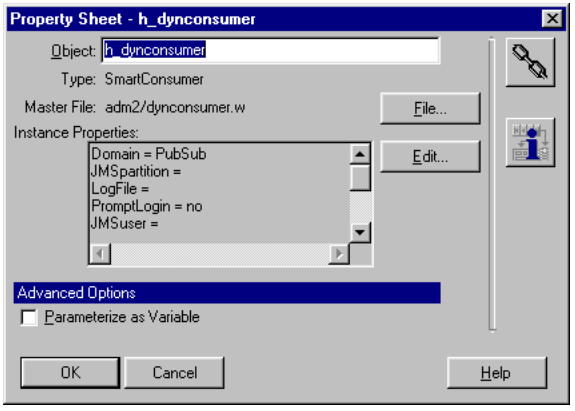
The SmartConsumer is supplied as a precompiled master. Creating and placing an instance is a straightforward process. Follow these steps:

1 ♦ Click the SmartConsumer tool icon in the Object Palette:



2 ♦ Position your mouse cursor over an empty spot in your workspace and click to place the object. If you have a SmartRouter in place, AppBuilder opens an Advisor dialog box and offers to create a ROUTER SmartLink. If, instead, you have a SmartB2BObject or SmartReceiver in place, the Adviser will offer to create an INMESSAGE link. In either case, accept the offer unless it seems inappropriate.

3 ♦ Click the object's menu button and choose Properties. The Property Sheet dialog box opens:



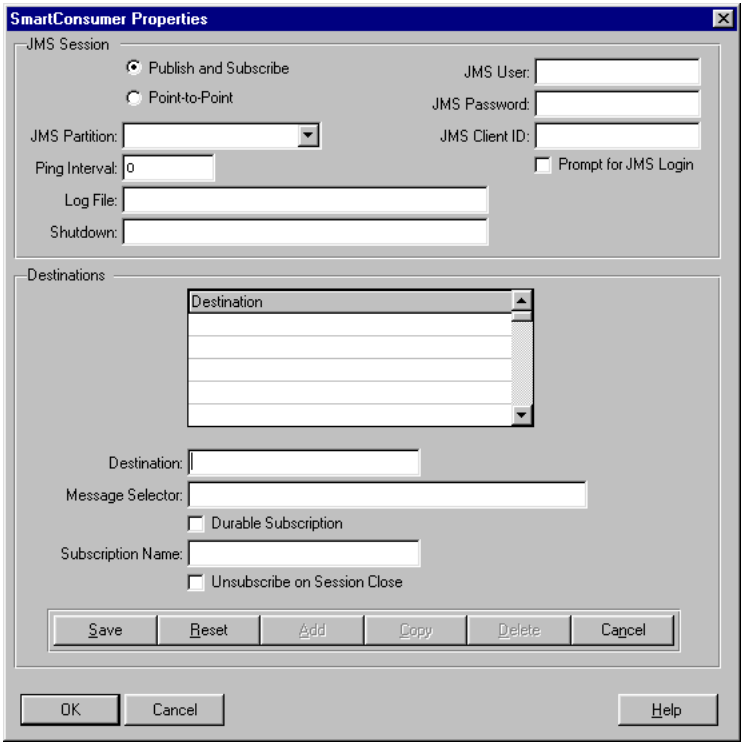

4 ♦ Change the instance handle to one that more accurately describes the role of this SmartConsumer in your application.

5 ♦ Configure the instance. Save your work.

### 6.4.4 Configuring a SmartConsumer Instance

There are only a limited number of instance properties you need configure. Follow these steps:

1 ♦ Begin by clicking the instance's menu button and choosing Instance Properties. The instance-properties dialog box opens (all fields shown as enabled for illustration):




2 ♦ Change the default values that do not meet your needs:

- **Domain** — Defaults to Publish-and-Subscribe. Setting this value to Point-to-Point creates a single copy of the message. Although a number of entities might have access to the message, the first one that reads the message deletes it from the pipeline. With the Publish-and-Subscribe mechanism, all subscribing entities can read the message. This allows implementation of a form of multicasting.

  The Domain value is not modifiable at run time.

- **JMS Partition** — Select a Partition from those listed. If there are no JMS Partitions listed, you can define as many as you like using the Service Parameter Maintenance Tool available on the PRO*Tools toolbar. For information about how to use that tool, see the online help.

- **Ping Interval** — Defaults to 0 (pinging off). Setting this field to some integer value in seconds causes the connection to the SonicMQ broker to be tested at that rate for the duration of the JMS session. It is good practice not to ping frequently, since pinging decreases system performance.
- **Log File** — Defaults to empty. Set this field to the pathname of the error-log file that will be used when running in batch mode.
- **Shutdown** — Defaults to empty. Set this field to the identifier for the Topic (Publish-and-Subscribe) or Queue (Point-to-Point) to be used for shutting down this SmartConsumer instance when running in batch mode. Sending any message to this destination causes this SmartConsumer object to begin the shutdown process.
- **JMS User** — Set this field to identify the user of the JMS Broker connection.
- **JMS Password** — Set this field to be the password that corresponds to the JMS User identifier.
- **JMS Client ID** — Set this field to a value unique to this client. This field is used to resolve ambiguities in cases where multiple clients might be using the same subscription and user names.

**NOTE:** You can supply the User, Password, and Client ID values at design time if you know them. Otherwise, check the Prompt-for-JMS-Login box and the object will prompt the user for them at run time.

- **Prompt for JMS Login** — Defaults to cleared. Setting this check box causes the object to prompt the user for login.
- **Destination** — Set this field to create/update an identifier for the destination Topic/Queue.
- **Message Selector** — If you wish to filter incoming messages according to values in header fields or properties, enter the expression here. The SmartConsumer object will perform the test on incoming messages and discard from the inbound queue/topic any messages that fail the test. Example: `company_name = "ABC" AND priority < 4`. Any message received from company ABC with a priority of 3 or less will be discarded without notice. The syntax is a subset of the syntax defined in SQL-92.
- **Durable Subscription** — Defaults to cleared. Enabled only when Domain is Publish-and-Subscribe. Set this check box to preserve the current subscription across JMS sessions.

- **Subscription Name** — Enabled only for Durable Subscriptions. Set this field to create/update a subscription name.
- **Unsubscribe on Session Close** — Defaults to cleared. Enabled only for Durable Subscriptions. Setting this check box causes the selected subscription to be cancelled when the JMS session ends.

### 6.4.5 Batch Mode: Starting and Stopping a SmartConsumer

Three properties are generally required before running SmartConsumer object in batch mode, if SonicMQ security checking is turned on. They are JMSUser, JMSPassword, and ClientID. Strictly speaking, ClientID is not required except when needed to distinguish between sessions using the same user identifier and password.

At present, you can set those properties in the instance properties dialog box or from the command line using the `-param` startup parameter:

```
prowin32 -b -p containerfilename -param "jmsuser,jmspassword,clientid" .
```

In the example shown, *containerfilename* would be the filename of a Simple SmartContainer that contains a SmartConsumer object and a SmartRouter object. The values you pass in the parameter string override any values set at design time using the Instance Properties dialog box.

#### Stopping a SmartConsumer

The most orderly way to shut down a SmartConsumer object running in batch mode is to send a message to the designated shutdown topic or queue. You identify that topic or queue at design time in the Shutdown property.

The shutdown message needs no body; its receipt by the shutdown Queue or Topic is sufficient.

When the SmartConsumer detects reception of a message in that Queue/Topic, it sends a shutdown message to its container object. Shutting down the container object shuts down all objects in the container (including the SmartConsumer itself) as well as any containers that were started by the SmartRouter to which the SmartConsumer is connected.

To set up and test a shutdown mechanism, follow these general steps:

1 ♦ Create a Simple SmartContainer

2 ♦ Create and place a SmartProducer in the SmartContainer. Set Message Type to HeaderMessage and configure other properties appropriately.

3 ♦ Create and place a SmartSender in the SmartContainer. If an Advisor offers to create an OUTMESSAGE link to the SmartProducer, accept the offer; otherwise, create the link using the SmartLinks editor. Set the Destination property to be the shutdown destination for the consumer, for example "Shutdown". Any message this SmartProducer sends will then be a shutdown message.

4 ♦ Open a Section Editor window and create a local override procedure called initializeObject. Insert this RUN statement as the procedure body:

```
RUN sendMessage IN THIS-PROCEDURE.
```

5 ♦ Save your work using a distinctive filename, for example `stopconsumer`. To test this shutdown mechanism, enter this statement on the command line:

```
prowin32 -b -p stopconsumer.w
```

## 6.5 The Message-handling Relationship: Internal Details

The process for handling a message is very much the same regardless of whether the message is coming in or going out.

### Outbound Message and Reply-handling Process

Sending a message requires cooperation between the SmartProducer and an object serving as the OUTMESSAGE-SOURCE (OMS). Generally you would use either a SmartB2BObject or a SmartSender as OMS. Figure 6–10 illustrates the process.

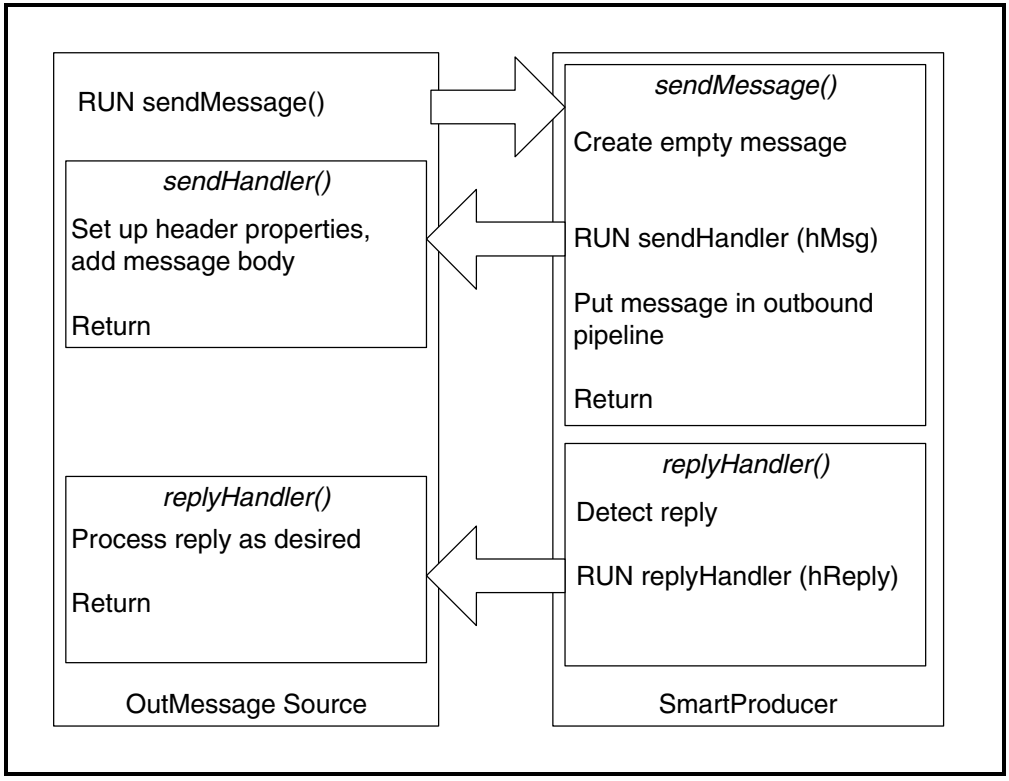



**Figure 6–10: Outbound Message-handling Process**

1. When a message is to be sent, the OMS calls sendMessage in the SmartProducer.
2. The sendMessage() routine in the SmartProducer creates an empty message. It then calls sendHandler() in the OMS, passing the handle of the message it created.
3. The sendHandler() routine in the OMS sets the message header properties, adds a body to the message, and returns.
4. The sendMessage() routine in the SmartProducer now has a complete message to send and calls the relevant routines in the 4GL–JMS API to start the message on its way.
5. When a reply comes in, replyHandler() in the SmartProducer calls replyHandler() in the OMS, passing it the handle of the reply.
6. The replyHandler() routine in the OMS gets the ID of the original message and the reply properties and body. It performs any desired processing on that information and returns.

### Inbound Message-Handling and Reply Process

Receiving a message requires cooperation between the SmartConsumer and an object serving as the INMESSAGE-TARGET (IMT). Generally you would use either a SmartB2BObject or a SmartReceiver as IMT. Figure 6–11 illustrates this process.

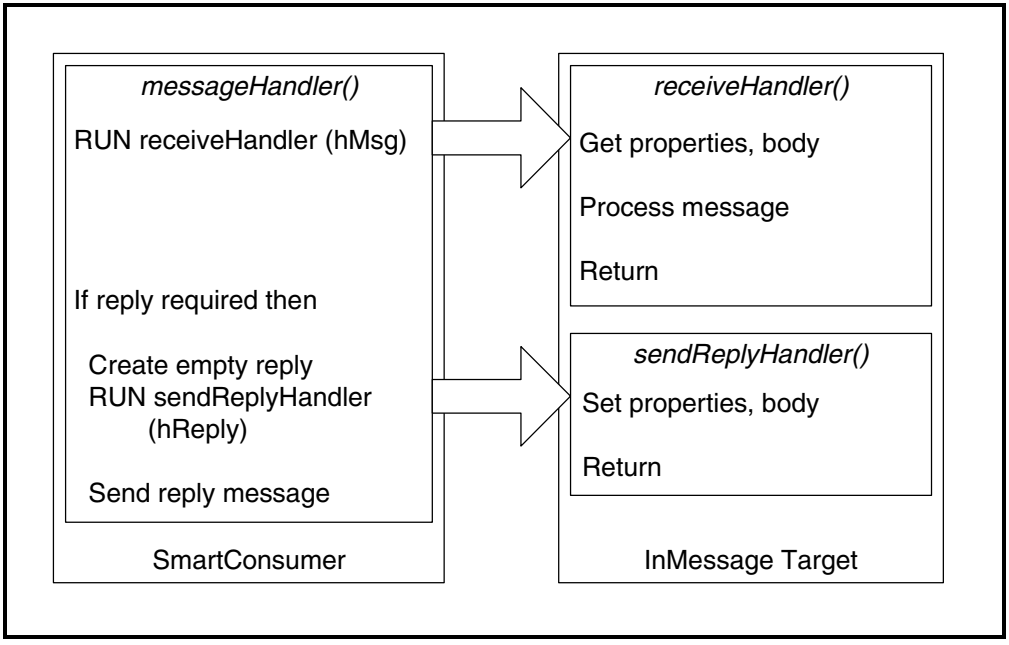



**Figure 6–11: Inbound Message-handling Process**

If you use a SmartRouter to distribute incoming messages, the SmartRouter acts as an IMT proxy, accepting calls from the SmartConsumer and passing them on transparently to the real IMT. The brief description below ignores the role of the SmartRouter:

1. When a message is received, the messageHandler() routine in the SmartConsumer calls the receiveHandler() routine in the IMT, passing the handle of the received message.

2. The receiveHandler() routine in the IMT extracts header properties and the message body, performs any desired processing, and returns.

3. If the message requires a reply, the messageHandler() routine in the SmartConsumer creates an empty reply message and calls sendReplyHandler() in the IMT, passing the handle of the reply it created.

4. The sendReplyHandler() routine in the IMT sets the header properties, adds a body, and returns.

5. The messageHandler() routine in the SmartConsumer now has a complete reply message to send and calls the relevant routines in the 4GL–JMS API to start it on its way.

### 6.5.1 Example of sendHandler Code

This code might be used to create a large text message in a SmartSender:

```
/*
** Declare a handle for the message
*/
DEFINE INPUT PARAMETER hMessage AS HANDLE NO-UNDO.

/*
** Set the message properties
*/
DYNAMIC-FUNCTION( 'setBooleanProperty':U IN hMessage,
         INPUT 'backordered':U,
         INPUT TRUE ).
DYNAMIC-FUNCTION( 'setDoubleProperty':U IN hMessage,
         INPUT 'minorderamount':U,
         INPUT 1500.00 ).
/*
** Append text blocks from tables to create message body
*/
FOR EACH table
 DYNAMIC-FUNCTION( 'appendText':U ) IN hMessage, table.datafield ).
END.
```

### 6.5.2 Example of sendHandler Override Code

This code might be used in a sendHandler() override in a SmartB2BObject:

```
DEFINE INPUT PARAMETER hMessage AS HANDLE NO-UNDO.

/*
** Set custom message properties (example only)
*/
  DYNAMIC-FUNCTION('setStringProperty':U IN hMessage,
      INPUT 'city':U,
      INPUT cCity).

/*
** Run the SmartB2BObject's version of sendHandler
** to assign the body of the message
*/
RUN SUPER.
```

### 6.5.3 Example of receiveReplyHandler Code

Any object that acts as an OUTMESSAGE-SOURCE to a SmartProducer must include a receiveReplyHandler() procedure. That procedure handles replies from message recipients. It might perform some or all of these functions:

- Identify the original message.
- Extract any important properties from the reply.
- Identify the reply type and obtain the reply body in an appropriate format.

This is an example of code that performs those functions for some OUTMESSAGE-SOURCE object other than a SmartB2BObject, for example a SmartSender:

*(1 of 2)*

```
DEFINE INPUT PARAMETER hReply AS HANDLE NO-UNDO.
/*
** Declare storage for obtaining the properties
*/
DEFINE VARIABLE cOriginalMsgID AS CHARACTER NO-UNDO.
DEFINE VARIABLE cReplyType AS CHARACTER NO-UNDO.
DEFINE VARIABLE cPropertyNames AS CHARACTER NO-UNDO.
DEFINE VARIABLE cProperty AS CHARACTER NO-UNDO.
DEFINE VARIABLE cPropType AS CHARACTER NO-UNDO.
DEFINE VARIABLE iThisProp AS INTEGER NO-UNDO.
DEFINE VARAIBLE cCharProp AS CHARACTER NO-UNDO.
DEFINE VARIABLE dDecProp AS DECIMAL NO-UNDO.
DEFINE VARIABLE iIntProp AS INTEGER NO-UNDO.
DEFINE VARIABLE lLogicProp AS LOGICAL NO-UNDO.
DEFINE VARIABLE mReplyBody AS MEMPTR NO-UNDO.

/*
** Identify the original message using the reply handle, get property names
*/
cOriginalMsgID = DYNAMIC-FUNCTION('getJMSCorrelationID':U IN hReply).
cPropertyNames = DYNAMIC-FUNCTION(‘getPropertyNames’:U IN hReply).

/*
** Loop through all properties, extracting their values according to type
*/
DO iThisProp = 1 TO NUM-ENTRIES(cPropertyNames):
 cProperty = ENTRY(iThisProp, cPropertyNames).
 cPropType = DYNAMIC-FUNCTION(‘getPropertyType’:U IN hReply,
                INPUT cProperty).
 CASE cPropType:

  WHEN ‘String’:U THEN DO:
   cCharProp = DYNAMIC-FUNCTION(‘getCharProperty’:U IN hReply,
                INPUT cProperty).
   /*
   ** Insert code here to process this character property
   */
  END.

  WHEN ‘Boolean’:U THEN DO:
   lLogicProp = DYNAMIC-FUNCTION(‘getLogicalProperty’:U IN hReply,
                INPUT cProperty).
   /*
   ** Insert code here to process this logical property
   */
  END.
```

*(2 of 2)*

```
  WHEN 'Byte':U OR WHEN 'Short' OR WHEN 'Int' THEN DO:
   iIntProp = DYNAMIC-FUNCTION('getIntProperty':U IN hReply,
                 INPUT cProperty).
   /*
   ** Insert code here to process this integer property
   */
  END.

  WHEN 'Long':U OR WHEN 'Float' OR WHEN 'Double' THEN DO:
   iDecProp = DYNAMIC-FUNCTION('getDecProperty':U IN hReply,
                 INPUT cProperty).
   /*
   ** Insert code here to process this integer property
   */
  END.

 END. /* case cproptype */
END. /* do iThisProp */

/*
** Identify the reply type
*/
cReplyType = DYNAMIC-FUNCTION('getMessageType':U IN hReply).

/*
** Extract the reply body in the appropriate format
*/
CASE cReplyType:

 WHEN 'BytesMessage':U THEN DO
  mReplyBody = DYNAMIC-FUNCTION('getMemptr':U IN hReply).
 END. /* when bytesmessage */

 WHEN 'TextMessage':U OR WHEN 'XMLMessage':U THEN DO:
  DO WHILE (DYNAMIC-FUNCTION('endOfStream':U IN hReply) = FALSE:
   cReplyBody = DYNAMIC-FUNCTION('getTextSegment':U IN hReply).
   /*
   ** Insert code here to put returned text segments into a form,
   ** such as temp-table records, that can be read by the routine
   ** that will process the body of this reply
   */
  END. /* do while */
 END. /* when textmessage */

END. /* case cReplyType */
```

### 6.5.4 Example of receiveHandler Code

This code might be used in an INMESSAGE-TARGET object other than a SmartB2BObject, for example a SmartReceiver. Note the similarities to the receiveReplyHandler code:

*(1 of 2)*

```
DEFINE INPUT PARAMETER hMessage AS HANDLE NO-UNDO.
/*
** Declare storage
*/
DEFINE VARIABLE cMessageType AS CHARACTER NO-UNDO.
DEFINE VARIABLE cMessageID AS CHARACTER NO-UNDO.
DEFINE VARIABLE mMessageBody AS MEMPTR NO-UNDO.

DEFINE VARIABLE cProperty AS CHARACTER NO-UNDO.
DEFINE VARIABLE cPropertyNames AS CHARACTER NO-UNDO.
DEFINE VARIABLE cPropType AS CHARACTER NO-UNDO.
DEFINE VARIABLE iThisProp AS INTEGER NO-UNDO.
DEFINE VARAIBLE cCharProp AS CHARACTER NO-UNDO.
DEFINE VARIABLE dDecProp AS DECIMAL NO-UNDO.
DEFINE VARIABLE iIntProp AS INTEGER NO-UNDO.
DEFINE VARIABLE lLogicProp AS LOGICAL NO-UNDO.
/*
** Loop through all properties, identifying types and extracting values
** appropriately
*/
cPropertyNames = DYNAMIC-FUNCTION('getPropertyNames':U IN hMessage).
DO iThisProp = 1 TO NUM-ENTRIES( cPropertyNames ):
 cProperty = ENTRY(iThisProp, cPropertyNames).
 cPropType = DYNAMIC-FUNCTION('getPropertyType':U IN hMessage,
                INPUT cProperty).
 CASE cPropType:

  WHEN 'String':U THEN DO:
   cCharProp = DYNAMIC-FUNCTION('getCharProperty':U IN hMessage,
                  INPUT cProperty).
    /*
    ** Insert code for processing a string property
    */
  END.

  WHEN 'Boolean':U THEN DO:
   lLogicProp = DYNAMIC-FUNCTION('getLogicalProperty':U IN hMessage,
                INPUT cProperty).
    /*
    ** Insert code for processing a logical property
    */
  END.
```

*(2 of 2)*

```
  WHEN 'Byte':U OR WHEN 'Short':U OR WHEN 'Int':U THEN DO:
   iIntProp = DYNAMIC-FUNCTION('getIntProperty':U IN hMessage,
                 INPUT cProperty).
    /*
    ** Insert code for processing an integer property
    */
  END.

  WHEN 'Long':U OR WHEN 'Float':U OR WHEN 'Double':U THEN DO:
   dValue = DYNAMIC-FUNCTION('getDecProperty':U IN hMessage,
                INPUT cProperty).
    /*
    ** Insert code for processing a decimal property
    */
  END. /* when decimal */
 END. /* case cPropType */
END. /* DO iThisProp */

/*
** Determine message type; extract body appropriately
*/
cMessageType = DYNAMIC-FUNCTION('getMessageType':U IN hMessage).
CASE cMessageType:

 WHEN 'BytesMessage':U THEN DO:
  mMessageBody = DYNAMIC-FUNCTION('getMemptr':U IN hMessage).
 END.

 WHEN 'TextMessage':U OR WHEN 'XMLMessage':U THEN DO:
  DO WHILE (DYNAMIC-FUNCTION('endOfStream':U IN hMessage) = FALSE:
   cMessageBody = DYNAMIC-FUNCTION('getTextSegment':U IN hMessage).
   /*
   ** Insert code here to put extracted text segments into a form,
   ** such as temp-table records, that can be read by the routine
   ** that will process the body of this message
   */
  END. /* do while */
 END. /* when text message */

END. /* case cMessageType */
```

### 6.5.5 Example of receiveHandler Override Code

This code might be used in a receiveHandler override in a SmartB2BObject:

```
DEFINE INPUT PARAMETER hMessage AS HANDLE NO-UNDO.
/*
** Declare storage
*/
DEFINE VARIABLE lBackOrdered AS LOGICAL NO-UNDO.

/*
** Get a specific property value
*/
lBackOrdered = DYNAMIC-FUNCTION('getCharProperty':U IN hMessage).

RUN SUPER.
```

### 6.5.6 Example of sendReplyHandler Code

This code might be used in the sendReplyHandler procedure of some INMESSAGE-TARGET object other than a SmartB2BObject, for example a SmartReceiver:

```
DEFINE INPUT PARAMETER hReplyMessage AS HANDLE NO-UNDO.
/*
** Set message properties
*/
DYNAMIC-FUNCTION('setBooleanProperty':U IN hMessage,
         INPUT 'backordered':U,
         INPUT TRUE).
DYNAMIC-FUNCTION('setDoubleProperty':U IN hMessage,
         INPUT 'minorderamount':U,
         INPUT 1500.00).
/*
** Assign the message body
*/
RUN setMemptr IN hReplyMessage
 (INPUT mMessage,
 INPUT 1,
 INPUT 200).
```

### 6.5.7 Example ofsendReplyHandler Override Code

This code might be used in a SmartB2BObject's sendReplyHandler override:

```
DEFINE INPUT PARAMETER hReplyMessage AS HANDLE NO-UNDO.
/*
** Set custom message properties
*/
DYNAMIC-FUNCTION('setStringProperty':U IN hMessage,
         INPUT 'city':U,
         INPUT cCity).
/*
** Run the SmartB2BObject's version of sendReplyHandler
** to assign the body of the message
*/
RUN SUPER.
```

# 7

# Control Objects

Menus, toolbars, pushbuttons, and similar objects serve as the controls and control panels of an application. They allow the user to navigate through databases, shift between functional layouts, set values for fields, and choose when and whether to make changes to database records.

This chapter discusses these objects:

- SmartToolbar
- SmartPanels
- Basic Pushbutton

## 7.1 Introduction to Control Objects

Nearly every nontrivial application offers many ways to control its behavior and, perhaps, appearance. Because of the way human memory works, grouping these controls according to their function is a standard way to improve their ease of use. You will probably have experienced this principle yourself.

Menu bars, typically with pull-down menus attached, were originally the only means provided for gaining access to an application's functions. Toolbars were developed as a type of supplemental menu bar for frequently used features. They gradually became a defacto standard after graphics editors and similar applications popularized the pushbutton as an action and tool interface.

As part of the ADM toolset, AppBuilder offers several control clusters that embody Smart technology and are able to control other SmartObjects:

- SmartToolbar
- SmartPanels for
  - Navigation
  - Saves and Updates
  - Commits

AppBuilder also offers the basic 4GL Pushbutton. You can use it to build your own control groups, if you choose.

## 7.2 SmartToolbars

The SmartToolbar object optionally combines a standard menu bar with a full-width toolbar, and provides access to database navigation and record-management functions. You can use a SmartToolbar in place of the standard menu bar in a SmartWindow, if you wish. The SmartToolbar offers the same functions that the larger, separate Navigation and Update SmartPanel button arrays provide between them.

The SmartToolbar has some limitations:

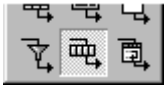
- You can only use the full SmartToolbar in a SmartWindow or SmartFrame. If you place one in a SmartDialog, only the toolbar component will activate, since a Dialog cannot have a menu bar even as part of a SmartObject.
- You can use only one SmartToolbar per organizer instance. If you need additional controls, you must create and group them by hand.

- The width of a SmartToolbar does not automatically change to agree with changes in the organizer object. You must change the width under program control, or manually.
- If you use the menu component, you cannot use the standard menu editor to populate it. You must write code to add any additional menu items that you desire.

### 7.2.1 Placing a SmartToolbar

AppBuilder supplies a single version of the SmartToolbar. If you do not need to add additional functions, using it in your application is very straightforward. Follow these steps:

1 ♦ Click on the SmartToolbar tool icon in the Object Palette:



2 ♦ Move the mouse cursor over a bare spot in your workspace and click to place the object. A menu bar and its corresponding toolbar have a well-defined position in a window, so you need not be careful where you place it—it establishes itself in the right place automatically, in some cases even moving other objects out of the way.

3 ♦ Configure it, and adjust its size appropriately.

Figure 7–1, Figure 7–2, and Figure 7–3 show the SmartToolbar in different states with both components and all supplied functions enabled. The Filter button appears insensitive because there was nothing connected to it at the time these images were taken.




**Figure 7–1: SmartToolbar Fully Enabled**




**Figure 7–2: SmartToolbar in Record-changed State**

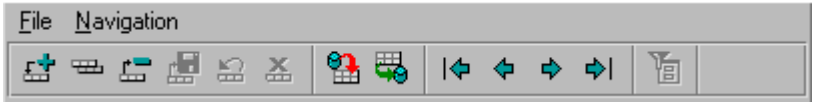
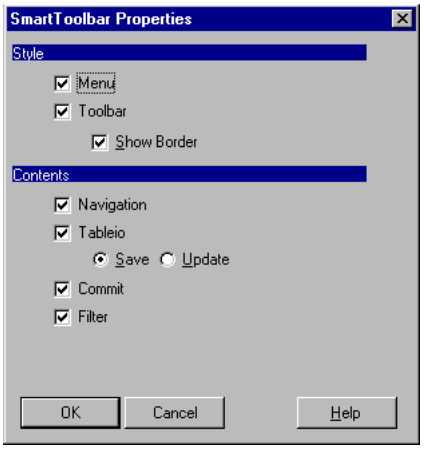



**Figure 7–3: SmartToolbar in Transaction-pending State**

### 7.2.2 Configuring a SmartToolbar Instance

A default SmartToolbar has a few properties you can change without writing code. To open the Properties dialog box, select Instance Properties from the menu-button menu. The dialog box opens:




The settings shown by this dialog box produce the toolbar in Figure 7–1. All but two of the options are set by default:

- **Menu** — Normally set. Clearing this box turns off the menu-bar component. Turning off the SmartToolbar's menu-bar component does **not** allow you to define or use the menu bar that is a feature of the Window widget.
- **Toolbar** — Normally set. Clearing this box turns off the toolbar component. Of course, you might not feel there is much reason to use the SmartToolbar at all, if you turn off this component.
- **Show Border** — Normally set. Clearing this box turns off the sunken border (implemented using a Rectangle) that appears surrounding the toolbar.
- **Navigation** — Normally set. Clearing this box turns off the four buttons that allow the user to change the current-record pointer. It also removes the entire Navigation menu and the Navigation link to the SmartDataObject, if there is one.
- **TableIO** — Normally set. Clearing this box turns off the six buttons that allow the user to update records and the database. It removes the corresponding items from the File menu and the TableIO link to the SmartDataViewer, if there is one.

- **Commit** — Normally cleared. Setting this box turns on the Undo and Commit transaction-processing buttons, and adds those items to the File menu.

- **Filter** — Normally cleared. Setting this box activates the Filter button and adds a corresponding item to the File menu. See the "Adding a Filter-control Window" section in this chapter, for information about making use of this button.

### 7.2.3 Creating a New SmartToolbar

AppBuilder supplies a single, precompiled SmartToolbar that can replace all the SmartPanels that AppBuilder also supplies. If you choose to use the Toolbar in your application, you can elect to enable more or fewer of those virtual SmartPanels.

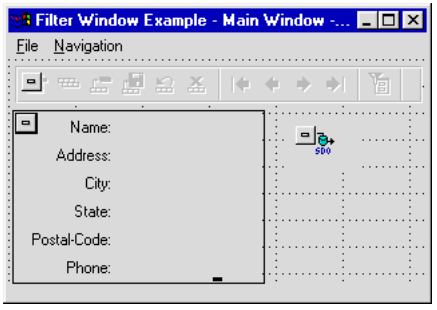
If you want to make other changes to the SmartToolbar, you must write code. See the *Progress ADM 2 Guide* for further information.

### 7.2.4 Adding a Filter-control Window

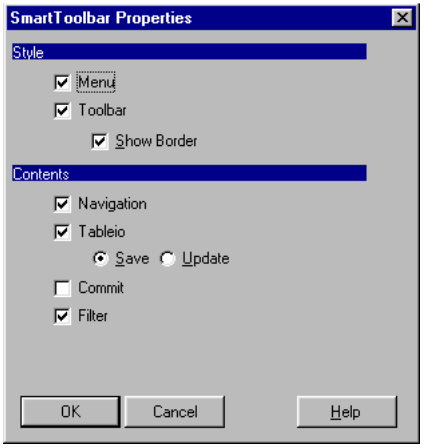
Displaying a SmartFilter object necessarily takes up a certain amount of space on the display. Since display space is typically in short supply, you might prefer to allow the user to control whether the Filter is visible. One way to do this is to put the Filter in a separate, callable SmartWindow. ADM makes this choice easier to implement by supplying a dedicated Filter pushbutton as part of the SmartToolbar object.

Follow these steps to create a sample application that uses a Filter Window:

1 ♦ Create a new SmartWindow instance to be the main window of this toy application. Populate it with a SmartDataObject, SmartDataViewer, and SmartToolbar. Allow the Advisor to create SmartLinks for you. Adjust sizes and positions as necessary. Your result should look similar to this:

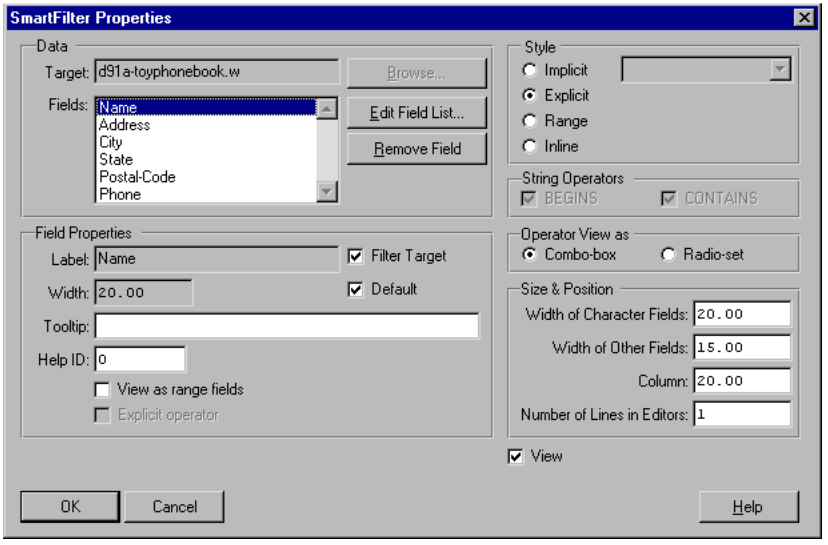


2 ♦ Click on the Toolbar's menu button and choose Instance Properties. When the dialog box opens, set the Filter check box:



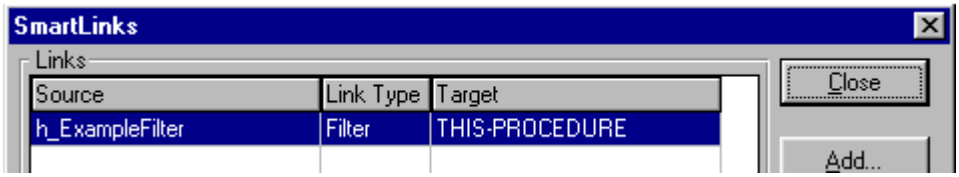

3 ♦ Choose OK to dismiss the dialog box. Create a SmartWindow to be your Filter window. Populate it with a SmartFilter.

4 ♦ Click on the Filter instance's menu button and choose Instance Properties. The dialog box opens:



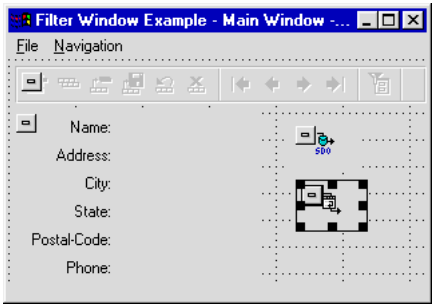

5 ♦ Identify the data source and the filterable fields. For the purposes of this example, accept the default style and view-as settings. Choose OK.

6 ♦ Click on the Filter's menu button and choose SmartLinks. Add a Filter link from the Filter object (represented by the handle identifier) to the Window object (represented by THIS–PROCEDURE):



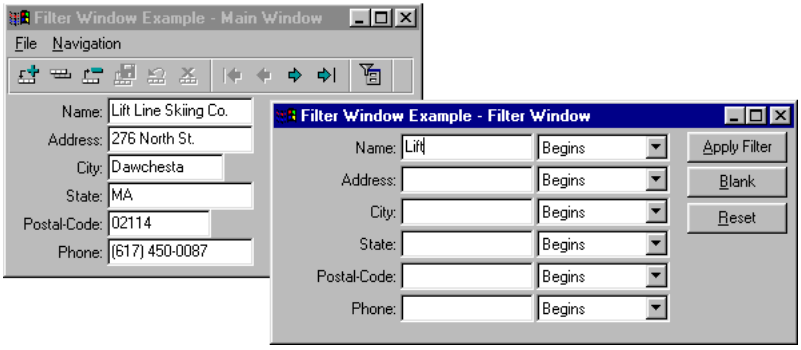

7 ♦ Close the SmartLink editor. Resize the Filter Window as necessary. Save it using a distinctive name.

8 ♦ Click on the SmartContainer tool in the Object Palette. When the Choose dialog box opens, select the Filter Window that you just saved, and choose OK. Place the object on any blank spot in your main Window. Allow the Advisor to create the links it suggests. Your result should now look similar to this:




9 ♦ Click on the menu button of the Filter Window instance you just placed. Choose Instance Properties. When the dialog box opens, clear the View check box. This will prevent the Filter Window from automatically displaying at startup. Choose OK to dismiss the dialog box.

10 ♦ Run your sample application. When it starts up, click the Filter button to open the Filter Window. Verify that you can filter the data stream:




The Filter button only opens the Filter Window. To close the Filter Window, you must click the Close button (the X) at the right end of its title bar.

## 7.3 SmartPanels

A SmartPanel is a task-oriented array of Pushbuttons that incorporates ADM Smart technology. As is true of all SmartObjects, SmartPanels are modular and communicate with other SmartObjects through SmartLinks. SmartPanel instances are immediately available as precompiled objects from the Object Palette. They are loosely coupled to theSmartObjects they control and, because they are not directly involved in the operation of a data stream, are not based on a master object.

AppBuilder offers several such predefined SmartPanels for your use:

- **Navigation (with Icon Labels)** — A four-button array with iconic labels that follow the arrowhead convention made familiar by tape recorders and VCRs. The four buttons select, respectively: the first record in a data stream; the record immediately preceding the current record, if possible; the record immediately following the current record, if possible; the final record in the data stream.

- **Navigation (with Text Labels)** — The same four-button array, but with text (First, Prev, Next, Last) rather than icon labels.

  **NOTE:** The scroll bar provides a navigational interface to a SmartDataBrowser. Adding a navigation array creates the possibility of confusion: since the SmartPanel is not able to communicate with the scroll bar, they can get out-of-sync. If you choose to add a SmartPanel for navigation, consider turning off the scroll bar in the Property Sheet for the Browse widget.

- **Update (Standard)** — A six-button array with text labels. The six buttons are Save/Update, Reset, Add, Copy, Delete, and Cancel.

  You must use this SmartPanel, or the equivalent buttons in a SmartToolbar, to enable changes to the database when using a SmartDataViewer for data display. You do not need it to make simple updates when using a SmartDataBrowser; a SmartDataBrowser that allows update will automatically perform the update when the user leaves the changed row. Use an Update SmartPanel with a SmartDataBrowser if you want the user to be able to add and delete records, or make more sophisticated updates.

- **Commit** — A two-button array used for transaction processing. The two buttons are Commit and Undo.

  A transaction is any group of operations that must be completed or discarded as an *atomic* unit. Failing to treat a transactional group atomically generally results in logical rather than physical data corruption. For example, after a credit purchase, successfully updating the inventory and shipping queue without also updating the customer's balance will cause an inventory leak. In this example, an update must be performed on all of the tables or on none of them, if the system is to remain in good order.

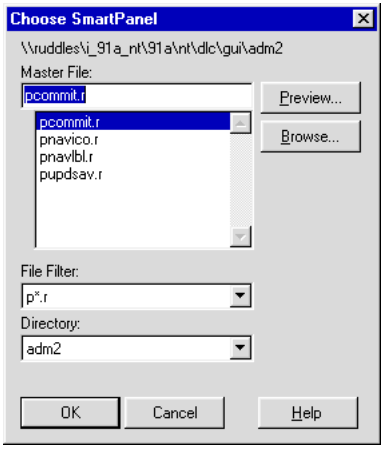
### 7.3.1 Creating and Configuring a SmartPanel Instance

Creating a SmartPanel is quite easy. After you determine which SmartPanel will meet your needs, follow these steps:

1 ♦ Click on the SmartPanel icon in the Object Palette:



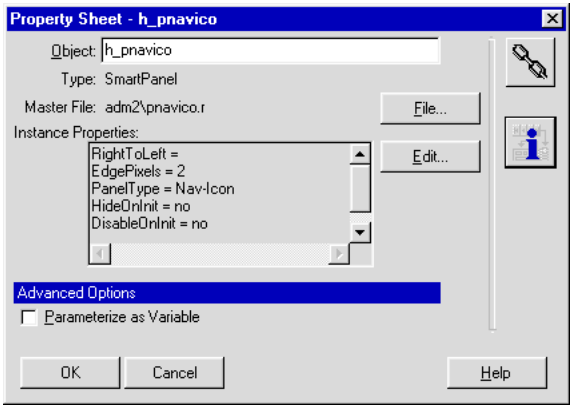
2 ♦ When the Choose SmartPanel dialog box opens, select the needed SmartPanel and choose OK:




**NOTE:** If you prefer, you can right-click on the tool icon, instead, and select the desired SmartPanel from the menu that opens there.

3 ♦ Position the mouse cursor over a bare spot in your workspace and click to place the object. If you have other SmartObjects in place, AppBuilder might open an Advisor dialog box and offer to create SmartLinks. Examine the offer and accept it if the proposed SmartLinks seem appropriate.

4 ♦ Click on the object's menu button and choose Properties. The Property Sheet dialog box opens. The sheet for the iconic Navigation Panel is shown here:



5 ♦ Change the instance handle to one that more accurately describes the role of this SmartPanel in your application.

6 ♦ The properties for appearance and behavior vary from Panel to Panel. Click the Edit button to inspect them (you can also reach them directly by choosing Instance Properties from the menu-button menu). Change those that do not meet your needs.

### Navigation SmartPanel Properties

A Navigation SmartPanel has three properties. You can change those that do not meet your needs:

- **Show Border** — Normally set. Clearing this check box turns off the border (implemented as a Rectangle) around the buttons.
- **Edge Pixels** — Normally 2. You can set the value to as much or as little as you like, with differing visual effects. Note that the number of pixels between the edge of the button and the inner edge of the rectangle does not change—the Rectangle's border grows inward toward the buttons rather than outward, in other words.
- **Internationalization** — (Applies only to the iconic Navigation Panel.) To conform your software to cultural conventions, you can choose which button—leftmost or rightmost—will set the pointer to the first record in a data stream. You can often determine the convention for a particular culture by looking at the direction in which the local language is written and read.

### Save/Update SmartPanel Properties

A Save/Update SmartPanel has only two properties. Change either or both, if you wish:

- **TableIO Type** — Normally Save. Setting this option to Update requires the user to first choose the Update button before the fields will make themselves editable.

  Save causes the fields to be editable by default. In either case, once the user has edited a field, choosing the Update button (now labelled Save) writes the changes back to the database immediately. If you also have a transaction (Commit/Undo) Panel installed, your application will journal the changes so that it can undo them again, if the user elects not to eventually commit.

- **Behavior of Add Button** — Normally prompts for one new record. Setting to Add Multiple causes your application to prompt for an arbitrary number of new records without requiring the user to choose the Add button again.

### Transaction SmartPanel Properties

There are no programmer-modifiable instance properties for the Commit (Transaction) SmartPanel, apart from the object identifier.

## 7.4 Basic Pushbuttons

Pushbuttons came into existence along with the use of electricity in the 1800s. Simple switches, they are such a convenient user-interface device that they have now become nearly invisible because of how universally they are used.

In a GUI, a Pushbutton is equivalent in power and generality to a menu item. Indeed, you can construe a Pushbutton as being a self-contained, single-item menu. Since Pushbuttons are normally always visible, they offer more immediacy and convenience than menu items at the cost of taking up more space. Because they do take up more space, you will probably want to use them sparingly.

Because of how they are used in everyday life, Pushbuttons in a GUI are associated with invocations, actions, and major state changes, not with minor state changes such as setting a single value. Using them in a way that runs counter to this expectation can make your applications harder and less satisfying for people to use.

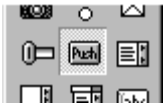
AppBuilder supplies a default, undefined Pushbutton object as well as a number of Pushbuttons preconfigured for common applications:

- **View Page** — Applies only to ADM paging (see the "Simple SmartContainers" section in Chapter 3, "Organizer Objects."). Causes objects associated with a certain page number to appear. You supply the number as an argument to the call in the trigger code. Already-visible objects assigned to other ADM pages do not disappear.
- **OK, Cancel** — Accepts/discards changes and closes the parent Dialog Box or Window.
- **Done** — Calls the application's shutdown routine.
- **Help** — Calls the help system.
- **First, Prev, Next, Last** — Moves the current-record cursor to the record indicated by the button.
- **Custom Size/Color** — (Default) Creates a generic Pushbutton with no action defined on it. You must add the action code yourself in the CHOOSE trigger.

### 7.4.1 Creating a Pushbutton Instance

Because they are such simple objects, you can create and configure Pushbuttons very easily. To include a Pushbutton in your application, follow these steps:

1 ♦ Right-click on the Pushbutton icon in the Object Palette:




When the menu opens, select the appropriate type of Pushbutton.

**NOTE:** If you prefer to use the default (undefined) Pushbutton, simply left-click on the icon instead; no menu will open.

2 ♦ Move your mouse cursor over a bare spot in your workspace window and click to place the new Pushbutton.

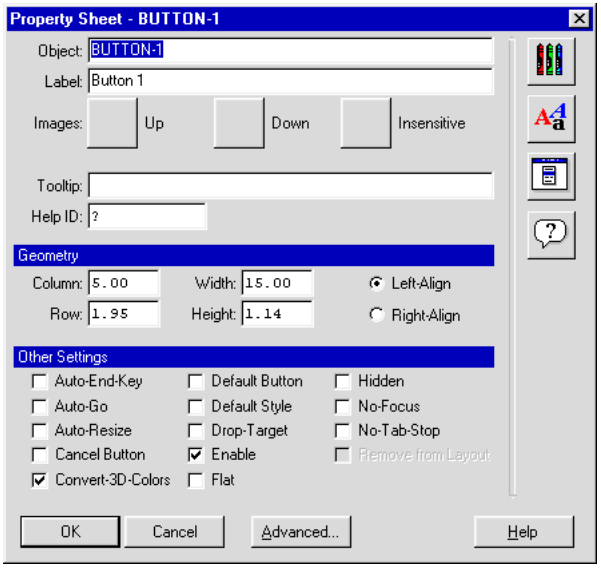
3 ♦ Configure, position, and size it appropriately.

### 7.4.2 Configuring a Pushbutton Instance

To configure a Pushbutton, you must:

- Customize the appearance to present appropriate cues to the user.
- Write trigger code to respond to the CHOOSE event.

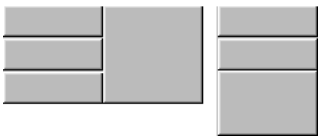
To begin customizing the Pushbutton's appearance, select it and choose Tools→ Property Sheet. The dialog box opens:




### Minimum Configuration

While you can accept many default settings, at a minimum you should:

- Change the object identifier to one that more closely represents the role of this Pushbutton in your application.
- Set the identifying label to a string or icon that will be meaningful to the user. See the "Setting a Pushbutton Label" section in this chapter for instructions.
- Resize the Pushbutton to fit it to its surroundings in a way that minimizes the space it takes up. For the best visual effect, you should make all related Pushbuttons the same size, and align them carefully. If one of them is more significant than the others, you can indicate that greater significance by making that Pushbutton unmistakably larger in one dimension, as in these examples:



### Miscellaneous Configuration (Other Settings)

These properties modify appearance or behavior. Change settings that do not meet your needs:

- **Auto-End Key** — Normally cleared. Setting this box causes the ENDKEY condition when the user chooses the Pushbutton.
- **Auto-Go** — Normally cleared. Setting this box causes the GO condition when the user chooses the Pushbutton.
- **Auto-Resize** — Normally cleared. Setting this box causes the Pushbutton to automatically change size when its label changes size.
- **Cancel Button** — Normally cleared. Setting this box causes a CHOOSE event for this Pushbutton when the user presses the **ESC** key. In other words, when this box is set, pressing the **ESC** key and choosing this Pushbutton are equivalent actions. No matter which the user does, the CHOOSE trigger code you write executes.
- **Convert 3D Colors** — Normally set. Only relevant if you use an iconic rather than a text label for this Pushbutton. Clearing this box allows Progress to modify the colors used by the image you use as a label and use system colors instead. This might produce an undesirable visual effect.
- **Default Button** — Normally cleared. Setting this box causes this Pushbutton to receive all **RETURN** key function events, regardless of whether it is selected at the time. Note that, while AppBuilder allows you to set this option for only one Pushbutton per Frame—it clears any earlier setting—it cannot detect the same setting in other Frames.
- **Default Style** — Normally cleared. Setting this box adds the thicker margin around the Pushbutton that indicates to the user that the object receives the **RETURN** key events.
- **Drop Target** — Normally cleared. Setting this box causes the Pushbutton to register an event when the user drags an object onto it. You must write the code needed to handle the event.
- **Enable** — Normally set. Clearing this box causes the Pushbutton to ignore attempts to choose it.
- **Flat** — Normally cleared. Setting this box causes the Pushbutton to display its label without any border indicating the clickable area, except when the cursor is inside that boundary.
- **Hidden** — Normally cleared. Setting this box causes the Pushbutton not to respond to implicit requests to display itself. It will respond only to explicit requests.

- **No Focus** — Normally cleared. Setting this box causes the Pushbutton not to indicate that it has the input focus, although the user can select it in the normal way.
- **No Tab Stop** — Normally cleared. Setting this box removes the Pushbutton from the enclosing Frame's traversal list. An object with this option set neither gains nor gives up the input focus in response to the user pressing the **TAB** key.

### Advanced Properties

The Pushbutton object has only the standard Advanced Properties. See the "Advanced Properties" section in Appendix A, "Frequently Used Dialogs," for further information.

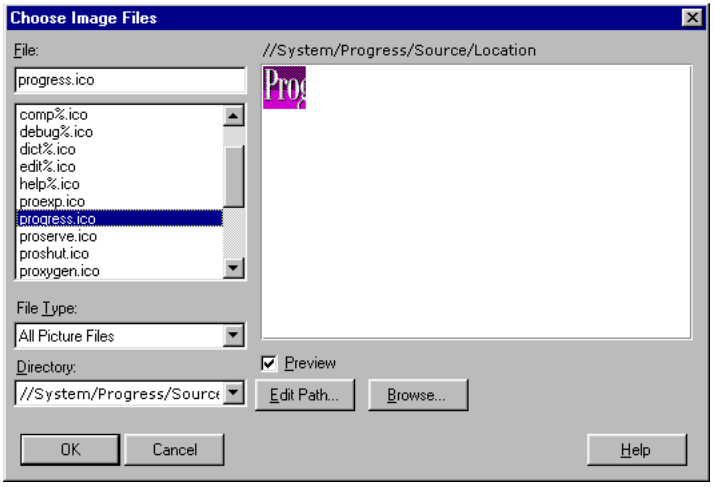
### Setting a Pushbutton Label

You can choose to label your Pushbuttons with text or an iconic image, but you cannot use both on the same object.

To use a text label, simply type the new string into the Label field. You can even do this without opening the property sheet, if you wish. For your convenience, AppBuilder displays below the toolbar in its main window both the identifier and the label of a Pushbutton object whenever it is selected.

To use an icon (and override any text label), follow these steps:

1 ♦ Double-click on the square button marked Up. The dialog box opens:



2 ♦ If the image file you wish to use is listed. select it and choose OK. Note that if you set the Preview check box, the image you select will display in the open area to the right of the listing. Note, too, that if you are looking for a particular type of file (.ico, .bmp, etc.) you can choose to filter for only those files by selecting the appropriate item from the File Type drop-down list.

3 ♦ If the image file is not listed, you can choose the Browse button to look for them. If you will often search certain other directories for image files, you can add the other directories to the drop-down list in the Directories combo box. Choose Edit Path to add a directory, or to modify the order in which they are listed.

# 8

# Other Objects

The other-objects category is a catch-all. Objects are included here only because they do not fit neatly into any other category.

The objects discussed here are:

- Simple (Undefined) SmartObject
- ActiveX (OCX) Control Objects in general
- Static Image
- Static Text
- Timer Object (ActiveX)

## 8.1 Simple (Undefined) SmartObjects

AppBuilder supplies a number of ADM SmartObjects for your use. As a developer, however, you will probably encounter situations where the current set of SmartObjects falls short. To solve a particular problem, you will need some SmartObject that does not exist. How will you find a solution?

You have two choices:

- Solve the immediate problem by using 4GL objects to fill in the gap, linking them into the larger web of SmartObjects in your application. This is a point solution, but it is a somewhat lower-cost one.
- Define and create your own SmartObject. The extra generality required might cost more in design time, but you would then be in a position to reuse the object indefinitely and gain the same advantages in development and maintenance costs that you get from other SmartObjects.

AppBuilder supplies the unfinished Simple SmartObject to make development quicker and less effortful. You create one by choosing it from the list under File→ New.

For detailed instructions about ADM and creating new SmartObjects, see the *Progress ADM 2 Guide*.

## 8.2 ActiveX (OCX) Objects

ActiveX (OCX) controls are objects that are conceptually similar to 4GL widgets and ADM SmartObjects, but written to Microsoft's Component Object Model (COM) standard. ActiveX objects supply encapsulated functionality of various kinds, and are available from a number of third-party vendors. The COM standard is the current successor to the older VBX standard.

**NOTE:** AppBuilder supplies only three ActiveX objects that you can use freely in your applications. In general, using an ActiveX object requires a separate licence from the supplier of the object.

AppBuilder allows you to select and position ActiveX objects in your GUI workspace in the same way you select and position native Progress 4GL and SmartObjects. To make this work, AppBuilder supplies a two-layer interface between the ActiveX object itself and the Progress 4GL world. The two layers are a Control Frame widget and a Control–Frame COM object. They translate between the Microsoft and Progress standards.

Although the Control Frame interface is too general a mechanism to make the differences between ActiveX objects and 4GL fully transparent to you, this section provides a general overview of how to create and configure instances.

For a full technical discussion of ActiveX objects in Progress applications, see the *Progress External Program Interfaces* manual.

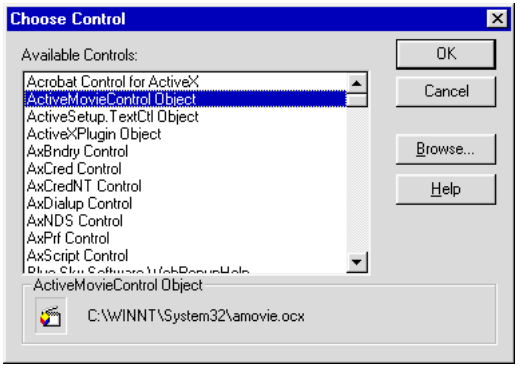
### 8.2.1 Creating an ActiveX Object Instance

Although an ActiveX object is very different, internally, to a 4GL object or SmartObject, the basic process of selecting and placing an OCX object is similar to the process for a native Progress object. Follow these simple steps:

1 ♦ Click on the OCX tool icon in the Objects Palette:




2 ♦ A dialog box opens. The dialog box lists all ActiveX controls found within reach of your development system. Because AppBuilder has no way of knowing whether you have a licence to use a particular OCX object, you should not assume that any particular object is available for inclusion in your applications.




3 ♦ Select the object you wish to use, and choose OK. Move your mouse cursor over a bare spot in your workspace and click to place the instance.

4 ♦ Configure and size the object.

### 8.2.2 Configuring an ActiveX Object Instance

Because OCX controls are not native to the Progress system, configuring them is a slightly more complicated process than configuring a comparable 4GL widget. AppBuilder generates a control_load() procedure to handle generic initialization of OCX objects. That procedure performs three tasks:

1. Creates a COM–HANDLE variable and initializes it with the handle of the COM-object layer of the Control Frame
2. Calls the LoadControls() method
3. Calls the optional procedure initialize-controls()

At this point, you must take over. You must perform some number of tasks that are object-dependent. In general, you must:

1. Assign a meaningful identifier to the object—or, more exactly, to its Control Frame. You need do nothing with the identifier (the Name property) of the OCX object itself.
2. Write the initialize-controls() procedure if you want to do special initialization.
3. Select the events you wish to handle for this object and create procedures to do that.
4. Size and position the object image within the workspace.

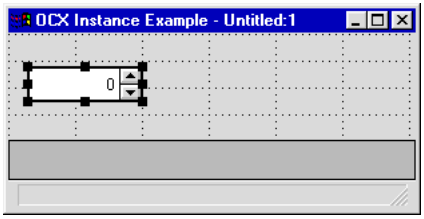
The following walkthrough presumes you are placing an instance of the Spin Control that AppBuilder supplies in the Objects Palette. Follow these steps:

1 ♦ Create a Window workspace and set the Message Area property, so that the workspace will be able to display messages conveniently.

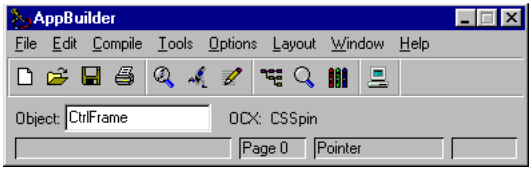
2 ♦ Click on the Spin Control tool icon in the Objects Palette:

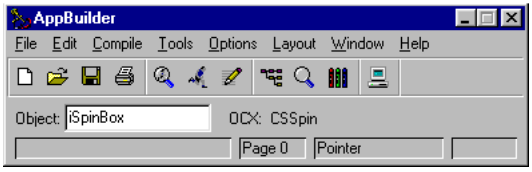


3 ♦ Move your mouse cursor over a bare spot in your workspace and click to place the instance. The default size will probably be too large—temporarily resize it now:

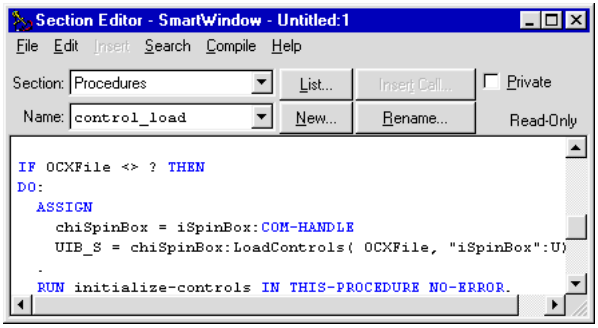



4 ♦ Note that AppBuilder's main window displays two Object identifiers. Replace the editable identifier (CtrlFrame) with one that is more meaningful in the context of your application. For this example, use iSpinBox:




5 ♦ Press **F2** to run the workspace momentarily, which forces AppBuilder to create the control_load() procedure. (This is not a step you need normally take; it is included here only for the sake of the example.) Stop the run as soon as you like. Choose Window→ Code Section Editor. When the editor window opens, select Procedures as the Section, and control_load as the procedure:



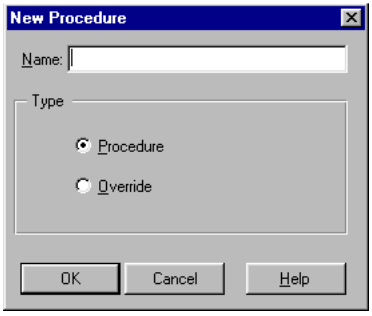
Note these lines of code:

```
ASSIGN
  chiSpinBox = iSpinBox:COM-HANDLE
  UIB_S = chiSpinBox:LoadControls( OCXFile, "iSpinBox":U )
.
RUN initialize-controls IN THIS-PROCEDURE NO-ERROR.
```

AppBuilder creates the COM–HANDLE identifier by prepending the digraph "ch" to whatever identifier you assigned to the Control Frame object. In this case, the Control Frame identifier is iSpinBox, so the handle is chiSpinBox. AppBuilder initializes the handle by assigning the Control Frame's COM–HANDLE property.

AppBuilder then uses that COM–HANDLE to call LoadControls(), and finally inserts a generic call to the optional initialize–controls() routine.

6 ♦ Choose the New button. The New Procedure dialog box opens:

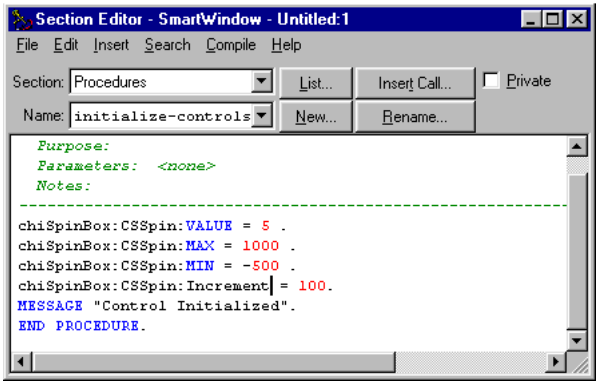



7 ♦ Type in **initialize–controls** as the procedure name. (Be careful not to use an underscore, if you are used to writing C.) Set Type to Procedure, if necessary, and choose OK. Section Editor generates a new stub routine.

8 ♦ Insert these lines of code:

```
ASSIGN
  chiSpinBox:CSSpin:Value = 5
  chiSpinBox:CSSpin:Max = 1000
  chiSpinBox:CSSpin:Min = -500
  chiSpinBox:CSSpin:Increment = 100.
MESSAGE "Control Initialized" .
```

Note how the code refers to the properties. The reference is not via the COM–HANDLE variable directly because that refers to the COM object layer in the adaptive interface, not to the ActiveX object itself. You must use an indirect reference, combining the COM–HANDLE , OCX, and property identifiers:

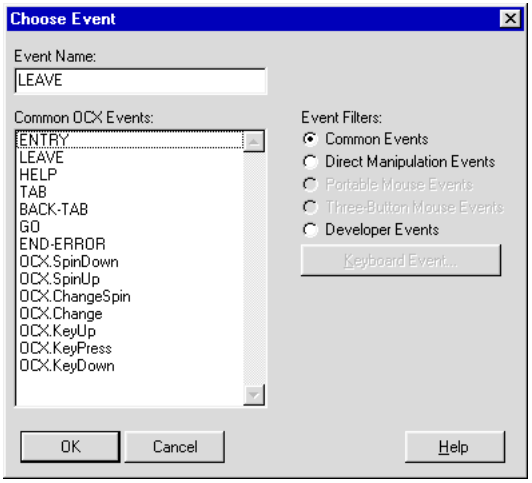



**NOTE:** The color-coding and all-caps appearance of certain properties, as shown here, is an artifact of the Section Editor settings. Those attributes are not significant in any way, and you can change them to suit yourself.

9 ♦ Change Section to Definitions and define a variable to represent the current spin value:

```
DEFINE VARIABLE iCurrSpinValue AS INTEGER INITIAL 0.
/*
** Note the initialization to 0.  This is compile-time
** initialization, done for safety's sake.
** It is overridden at run time by the initialization to 5.
*/
```

10 ♦ Change Section to Triggers, and choose New. The dialog box will open:

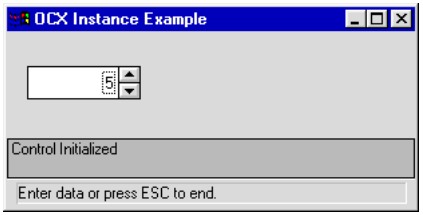



11 ♦ Select OCX.SpinUp and choose OK. Section Editor generates a new stub event handler. Insert these lines of code to complete it:

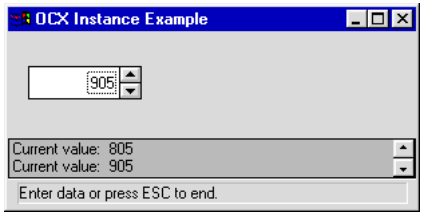
```
iCurrSpinValue = chiSpinBox:CSSpin:VALUE.
MESSAGE "Current value: " iCurrSpinValue.
```

12 ♦ Choose New again, and create a trigger for the OCX.SpinDown event. Use the same code.

Run the workspace. You should see the Spin Control initialized to 5, and your "Control Initialized" message:



When you operate the arrowhead buttons in the control, you should see the values change in increments of 100, to the limits –495 and 905:




### 8.2.3 Generic ActiveX Event Handlers

You might sometimes find that you have several ActiveX objects, perhaps of different kinds, that can all register the same kind of event. If it would meet your needs, you can write a single event handler that will respond to that event regardless of which object instance experiences it. If you do write such a generic handler, it need not apply to all objects. You can override it by writing handlers that you bind to specific instances. You can have only one generic handler for a particular event type, but you can have as many instance-specific handlers as you like.

Inside the event handler, use the generic identifier COM–SELF to refer to the object instance that experienced the event. For example, to refer to the experiencing instance's Value property in a generic handler, you would use the expression **COM–SELF:Value**. Note that you do not need to use the usual *chInstanceHandle:OCXName:*Value format.

To create a generic event handler, follow these steps:

1 ♦ Choose Window→ Code Section Editor to open that window. Set the Section to **Procedures**, not Triggers, and choose the New button.

2 ♦ Type in **ANYWHERE.***Eventname* as the name of the procedure (where *Eventname* is the actual name of the event, as for example Tick, Click, Choose, etc.).

3 ♦ Insert the code for handling the event.

4 ♦ If you want to bind a special handler to a particular instance, overriding the generic handler, set Section to Triggers, and choose New. Select the same event and define the special handler for it.

5 ♦ If you want to remove a special handler, making the generic handler the new event-service routine, bring the special handler up in the edit window and choose Edit→ Delete Trigger.

### 8.2.4 ActiveX (OCX) Properties Data

AppBuilder saves in a single file the properties data for all OCX objects that are local to a particular external procedure. (The file is not human readable.) By default, AppBuilder saves that file in the same directory where it saves the source files for the procedure, and using the same basic name with the extension .wrx. You can choose to use a different directory and/or filename, if you wish. To do so, follow these steps:

1 ♦ Choose Tools→ Procedure Settings.

2 ♦ From the OCX Binary drop-down list, select Other.

3 ♦ Type in the new filename, including the path if you want AppBuilder to store the file in some place other than in the default working directory.

## 8.3 Static Images

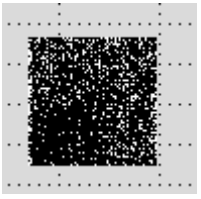
Static Image objects have no behavioral repertoire of their own, though you can use them as labels on Pushbuttons and in similar contexts. Consider cultural differences when choosing images, especially when choosing images to use as icons and labels. What is perfectly clear to one person might be mystifying to another, or even offensive. Be especially wary of using visual puns: they almost never translate well.

To create and configure a Static Image element, follow these steps:

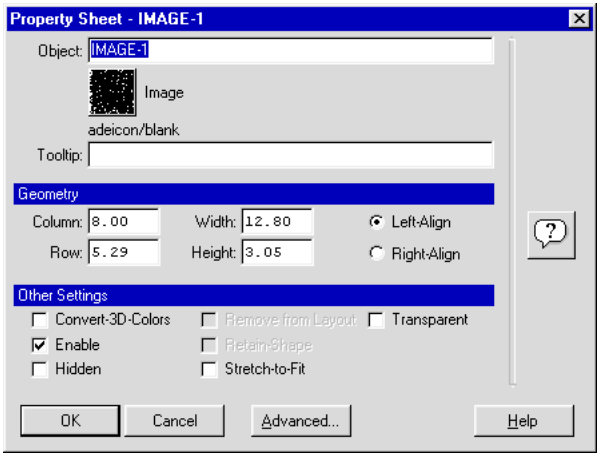
1 ♦ Click on the Image tool icon in the Objects Palette:



2 ♦ Move your mouse cursor to a bare spot in your workspace, and click to place the new Image object. Initially, the Image is undefined and looks like a smudged square. Note that it might not look exactly as shown here. Until an image is loaded, you see only the random contents of that area of memory:



3 ♦ Select the object and choose Tools→ Property Sheet. The dialog box opens:



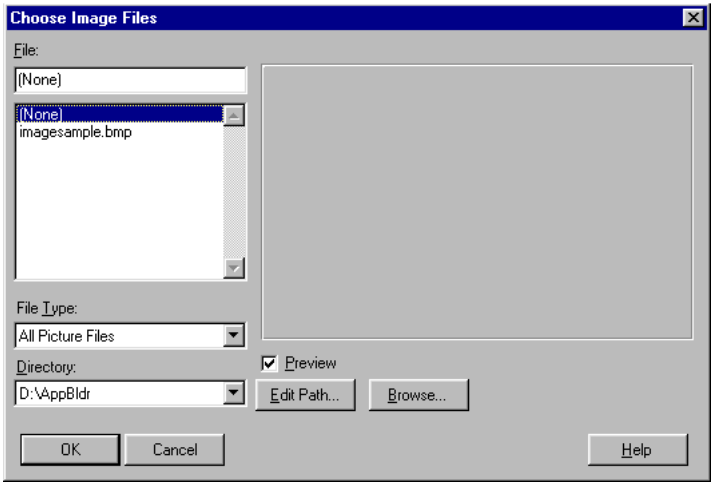

### Minimal Configuration

At a minimum, you should do these things:

- Change the Object identifier to one that more clearly conveys the purpose of this image in your application.
- Identify the actual image file.

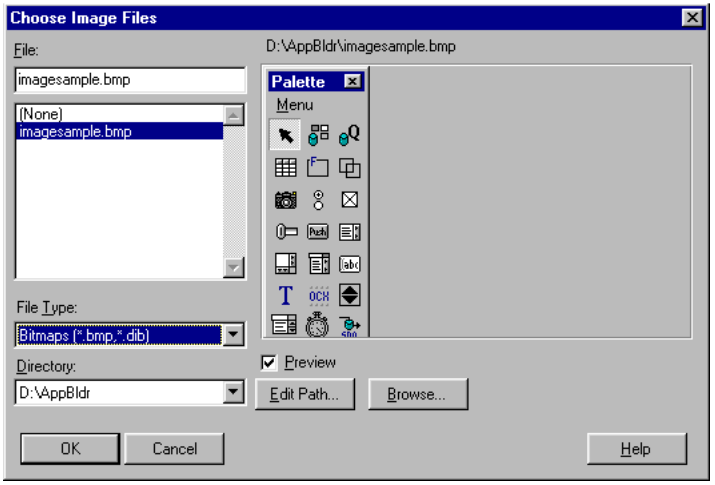
To identify the image file, follow these steps:

1 ♦ Click on the smudge image. The Choose Image Files dialog box opens:



2 ♦ Select the image type desired. AppBuilder supports many different image formats.

3 ♦ If the image you want to use is not listed, choose the Browse button to move to the correct directory. Select the filename. You will see a preview of the image, if you have the Preview box checked. For the purpose of this example, the image is that of the Objects Palette:



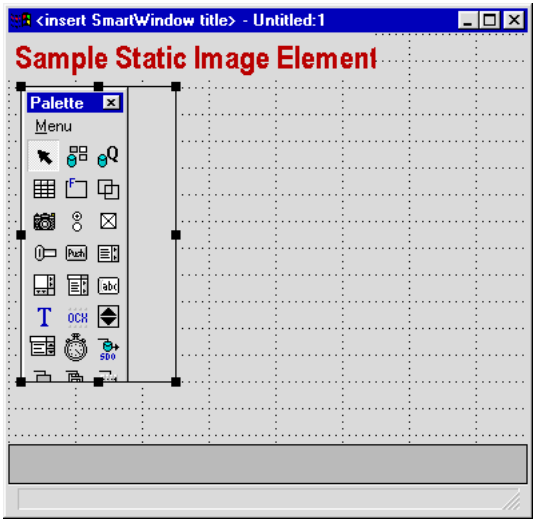

Note that Preview shows as much of the image as possible, at the size it will appear when the application runs; Progress does not rescale images here. If the image is the wrong size for your purposes, you must use some tool such as a graphics editor to rescale it before use, or must write code to rescale it under program control.

4 ♦ When you have the correct image file identified, choose OK to return to the property sheet. Some portion of the image—not necessarily the whole thing—appears in place of the smudge, as a token that you have successfully identified the image. The fully qualified filename of the image also appears, for your convenience:




5 ♦ Set any other properties you consider appropriate, and choose OK to dismiss the property sheet.

6 ♦ Drag the handles of the Image object to size and position it. Note that sizing incorrectly does not deform the Image, it clips it:




### Help

You can include a Tooltip string, if you wish. Under MS-Windows, the Image object will display the Tooltip string whenever the user allows the mouse cursor to hover over the image.

### Geometry

If you prefer to set the origin and size of the image explicitly rather than interactively, do so here.

### Miscellaneous Properties (Other Settings)

There are a few other properties you can change, if the default values do not meet your needs:

- **Convert 3D Colors** — Normally cleared. Setting this box causes AppBuilder to substitute platform-specific system colors in place of the actual colors in the Image. This often produces an unwelcome effect.
- **Enable** — Normally set. Clearing this box causes the Image object to decline the input focus when offered.
- **Hidden** — Normally cleared. Setting this box prevents the object from responding to implicit requests to show itself. You would have to include an explicit request.

- **Remove from Layout** — Cleared. Cannot be set except when defining an alternate layout.
- **Retain Shape** — Cleared. Can be set only if the option Stretch to Fit is set. Setting this box causes any stretch operation to maintain the original aspect ratio of the image. Does not apply to images that take the .ico file extension.
- **Stretch to Fit** — Normally cleared. Setting this box causes the image to be stretched to fit the available space. If Retain Shape is also set, the stretched image will maintain the original aspect ratio, even at the expense of fitting the space. Does not apply to images that take the .ico file extension.
- **Transparent** — Normally cleared. Setting this box causes the image to clear all its background pixels to a transparent state, thus revealing whatever is underneath them. The image considers any pixel a background pixel if it is the same color as the pixel in its lower-left corner. Does not apply to images that take the .ico file extension.

### Advanced Properties

The Static Image object has the normal minimum Advanced Properties. See the "Advanced Properties" section in Appendix A, "Frequently Used Dialogs," for more information.

## 8.4 Static Text

Static Text elements have no behavioral repertoire. They just sit there. Nevertheless, they can play an important role in your application by calling attention to important widgets, and by contributing to an overall corporate image, where appropriate.

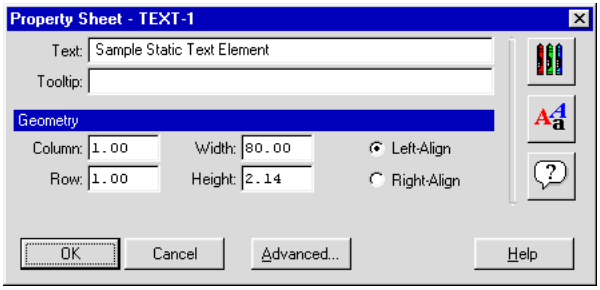
To create and configure a Static Text element, follow these steps:

1 ♦ Click on the Text tool icon in the Objects Palette:




2 ♦ Move your mouse cursor to a bare spot in your workspace, and click to place the new Text object.

3 ♦ Select the object and choose Tools→ Property Sheet. The dialog box opens:




4 ♦ Type in the text string the object will display.

5 ♦ Choose a special typestyle and colors, if appropriate. See the "Choosing Colors" and "Choosing a Typestyle" sections inAppendix A, "Frequently Used Dialogs," for more information.

6 ♦ Add translation information, if necessary. See the "String Attributes" section in Appendix A, "Frequently Used Dialogs."

7 ♦ Choose OK to dismiss the property sheet. Select the object and drag its handles to position and size it appropriately.

## 8.5 Timer Objects (ActiveX)

The majority of GUI-based applications follow an asynchronous, event-driven model similar to that for real-time or process-control applications. The majority of their time is spent waiting for input, in this case from the user rather than some hardware device.

But there are a number of cases for which it can be convenient to perform some activity at a regular interval, regardless of what the user might be doing. Providing a real-time clock display is perhaps the simplest case. Checking or reporting on the state of some lengthy process is another.

AppBuilder supplies a simple OCX Timer object for your use. The Timer is interrupt-driven from the system clock, and generates a single event (Tick) at some regular interval you determine. You configure the Interval property to be a multiple of 1/1000 of a second, and write an event handler to react to the Tick event. The Timer will continue to generate this event with metronomic regularity until you reset or disable it.

**CAUTION:** It takes a certain amount of time to react to any event. If you set up a situation where you must respond to every Tick event, but you configure them to occur more frequently than you can handle them, your application might behave in a way that is both strange and hard to debug.

To create and configure a Timer instance, follow these steps:

1 ♦ Click on the OCX Timer tool icon in the Objects Palette:



2 ♦ Move your cursor to some bare spot in your workspace and click to place the object. The Timer object has a design-time representation, but is invisible at run time.

3 ♦ Change the Object identifier, shown in AppBuilder's main window, to one that more clearly identifies the role of this Timer in your application.

4 ♦ Choose Window→ OCX Property Editor to open the Timer's own property sheet.

5 ♦ Set the Interval property to the desired integer frequency, representing thousandths of a second. For example, to get a Tick event every second, set Interval to 1000. For 1/10th of a second, set Interval to 100, and for a Tick every 5 seconds, set Interval to 5000. Close the window.

6 ♦ Open the Section Editor. Set the Section to Triggers, if necessary, and choose the New button to create a new event handler. Select OCX.Tick as the event, and choose OK. AppBuilder will create a stub event handler.

7 ♦ Insert the code to perform the task for which you set the timer.

This is a toy handler for the Tick event:

```
/*
**  In the definitions section:
*/

DEFINE VARIABLE bTickTock AS LOGICAL INITIAL FALSE.

/*
**  In the OCX.Tick event handler:
*/

bTickTock = NOT( bTickTock ).  /* flip the value */
IF bTickTock THEN  /* decide whether to tick or tock */
  MESSAGE "Tick".
ELSE
  MESSAGE "Tock".
```

When run, this toy event handler will alternately, and endlessly, write “Tick” or “Tock” to the message area of the Window, presuming you have configured your Window object to have a message area. The endless, synchronous looping action comes not from any loop code in the handler, but rather from the fact that the Timer itself is bound to the endlessly-operating system clock.

# A

# Frequently Used Dialogs

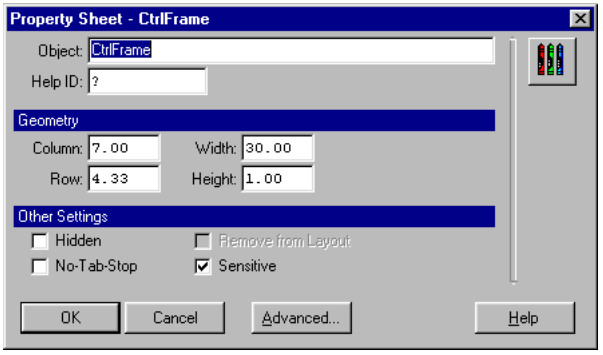
You will encounter a number of dialog boxes over and over again as you use AppBuilder. While the context might change in which you find them, they themselves generally have a somewhat fixed format. This appendix serves as a reference for their use. The dialog boxes discussed here are:

- ActiveX (OCX) Control Frame Properties
- ADM Pages Manager
- Advanced Procedure Settings
- Advanced Properties
- Colors Selector and Editor
- Custom Lists Naming
- Database Field Selector
- Method Library Includes List
- Procedure Settings
- Properties Window
- SmartLinks Editor
- SmartInfo

- SmartObject Properties
- String Attributes
- Traversal Path Editor
- Temp-Tables Maintenance
- Text Style Selector and Editor

## A.1 ActiveX Control Frame Properties

When you select an ActiveX (OCX) object and choose Tools → Property Sheet, the dialog box that opens does not display the properties for the ActiveX object itself. It displays the properties for the invisible Control Frame object that serves to adapt the ActiveX object to the Progress environment:

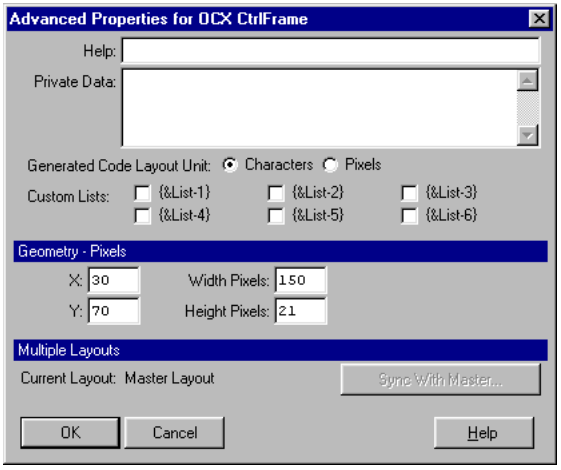



Since Control Frames are adapters, they have relatively few properties, and not all of those have a perceptible effect. You can set those that exist, if you choose:

- **Object** — You should change this identifier to better reflect the role of the ActiveX object in your application.
- **Help ID** — If you plan to provide context-sensitive help for this object, enter a unique integer here as the identifier.
- **Geometry** — If you prefer not to position and size the object interactively, you can set those values explicitly here instead. This dialog box uses character units; choose the Advanced button to set the same values using pixel units.
- **Hidden** — Normally cleared. Setting this box prevents the ActiveX object from responding to implicit requests to display itself.
- **No Tab Stop** — Normally cleared. Setting this box prevents the object from participating in the enclosing Frame's traversal group. Unless an object is part of a traversal group, it can neither gain nor lose the input focus as the result of the user pressing the **TAB** key.
- **Remove from Layout** — Cleared. Cannot be set except when defining an alternate layout.
- **Sensitive** — Normally set. Clearing this box causes the object to decline the input focus when offered.

### Advanced Properties

Click on the Advanced button to open the Advanced Properties dialog box for this Control Frame:




The Control Frame does not have the properties Manual Highlight, Movable, Resizable, or Selectable. Nor does it offer you the ability to set alignment.

## A.2 ADM Pages Manager



The button shown here is available in the Procedure Settings dialog box for objects that are members of the class SmartContainer.

ADM paging allows you to multiplex your display space in useful ways. But once you have a complex, paged design in place, managing all the

SmartObjects can be quite a difficult task. AppBuilder provides the Pages dialog box to help with that.

Using the Pages dialog box, you can:

- Review the list of SmartObjects assigned to a page.
- Set the Design and Startup pages.
- Move SmartObjects from one page to another.
- Delete SmartObjects from a page.

- Delete whole pages.
- Exchange the SmartObject contents of any two pages.

**NOTE:** The Pages dialog box only provides help with SmartObjects. Other objects, whether basic 4GL or ActiveX (OCX), are not shown by this dialog box.

To open the Pages dialog box, choose Tools→ Procedure Settings, or the Procedure Settings button on the Toolbar, then choose the Pages button. Figure A–1 shows the dialog box.

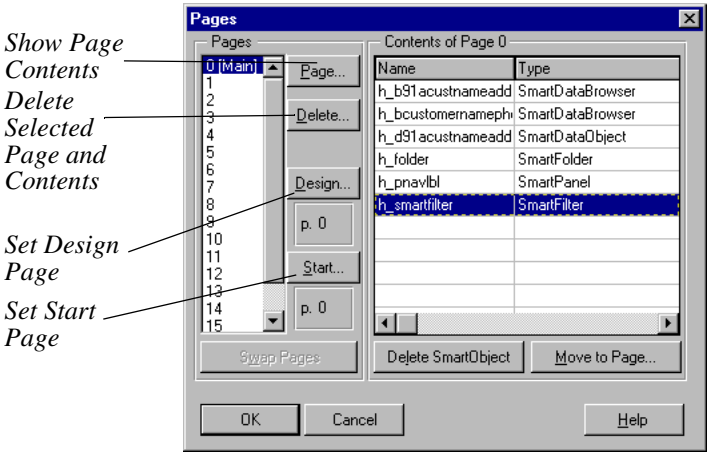



**Figure A–1: Pages Dialog Box**

### Reviewing the List of SmartObjects Assigned to a Page

To see the list of SmartObjects assigned to any page, click on that page number in the Pages list at the left side, or choose the Page button and enter the desired page number in the small dialog box that opens in response.

**NOTE:** Choosing to set the page number through the Page button and dialog box only determines which page's contents are displayed in the right-hand list. It does **not** set the Design Page as it would do in other circumstances.

### Setting the Design or Startup Page

The paging system identifies two special pages:

- **Design Page** — Whenever you place a SmartObject, it always goes onto the current Design Page.
- **Startup Page** — Whenever you run an application that uses ADM paging, the Startup Page is always the initial page displayed.

See "Special Pages" in Chapter 3, "Organizer Objects," for a more complete discussion of the relationship between these pages and the default Page 0 (zero).

To alter where you place future SmartObjects, or which SmartObjects appear at startup, choose the Design... or Start... button as appropriate, and enter the desired number in the dialog box that appears. Note that the dialog boxes are identical apart from the text in their title bars. Be sure you are setting the appropriate value.

### Moving a SmartObject Between Two Pages

To move a SmartObject from one page to another, highlight the object in the listing for its current page, choose the Move to Page... button, enter the destination page number in the Move to Page dialog box, and choose OK. Note that you cannot move the object by cutting and pasting the list entry.

### Exchanging the Contents of Two Pages

To completely exchange the contents of two pages, one of which might be empty, in a single operation, highlight them both (shift-click to highlight the second) and then choose the Swap Pages button. If one of the pages is empty, the swap operation will be equivalent to a move-all operation.

### Deleting a SmartObject from a Page

To delete a SmartObject from a page, highlight the object in the listing and choose the Delete SmartObject button.

### Deleting a Page and All Its SmartObjects

To delete a page and all SmartObjects on it, select the page number in the left-hand list, then choose the Delete button. A question box prompts you for confirmation. On completion, the page number will no longer appear in the list. You can easily recreate the page by assigning some SmartObject to it.

## A.3 Advanced Procedure Settings

AppBuilder uses the same Advanced Procedure Settings dialog box for all procedure level objects. Some of the fields might not apply to a particular object type, either because the qualities are predefined and fixed, or because they are entirely irrelevant.

You invoke this dialog box by choosing the Advanced button in the Procedure Settings dialog box.

**NOTE:** If you close the Advanced Procedure Settings dialog box by clicking OK, you will always be shown a message box reminding you to include all needed method libraries. This is true even if you have only inspected the settings. If you have made no changes you want to save, you can avoid this message by clicking Cancel instead of OK.

Figure A–2 shows the dialog box with the options that are enabled for SmartDialog objects.

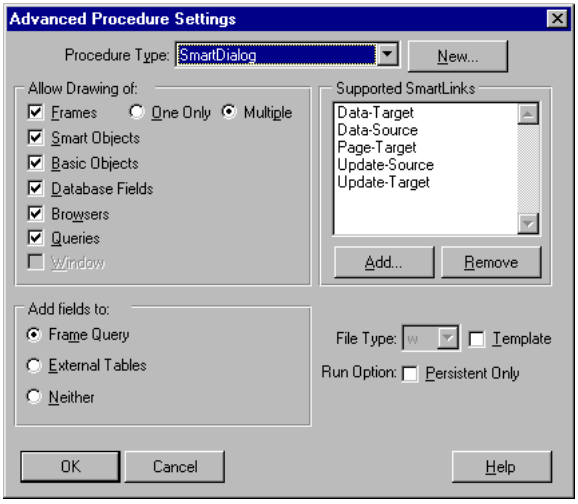



**Figure A–2: Advanced Procedure Settings Dialog Box**

Clockwise from the top, the options are:

- **Procedure Type** — AppBuilder sets this by default to match the object type. In other words, when you open this dialog box for a SmartWindow, you will see "SmartWindow" listed here. If you wish, you can select a procedure type different to the object type, for example a SmartDataBrowser procedure for a SmartWindow type. For more information, see the *Progress ADM 2 Guide*.

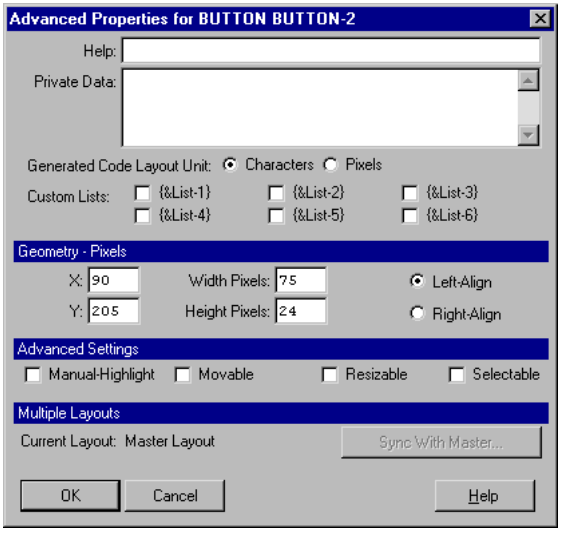
- **Supported SmartLinks** — Lists the SmartLink types an object knows how to handle as originator, recipient, or both. As shown in Figure A–2, a SmartDialog can handle either end of a Data or Update link, and knows how to be on the receiving end of a Page link.

You can add additional link types to this list, if you are prepared to write the code to support them. For further information, see the *Progress ADM 2 Guide*.

- **File Type** —Determines the filename suffix when the file is saved. For SmartObject sources, this is always .w.
- **Template** — Normally cleared. Checking this box causes certain information in the file to be automatically stripped out before saving, making the file more generic. If left unchecked, the file is saved as a master file rather than a template.
- **Add Fields to** — Determines where database fields will be added: to the Frame Query, the procedure's external table, if any, or to neither. Generally you should not alter the default setting, which is Frame Query.
- **Run Option: Persistent Only** — Normally set. When set, any attempt to run the object nonpersistently fails with an error message. This is true of all SmartObjects except the SmartDialog.
- **Allow Drawing of** — Determines what sort of objects can be made children of this object. If you clear, for example, the Basic Objects box, AppBuilder will refuse to allow you to place basic 4GL objects onto this workspace. You will be able to choose a 4GL object tool from the Objects Palette, but not release such objects onto the workspace.

## A.4 Advanced Properties

All basic 4GL objects have an Advanced Properties dialog box under the Advanced button in their property sheet. Most advanced properties are the same for all widgets. The Advanced Properties dialog box for a Pushbutton is shown here:




Change any of the default settings that do not meet your needs:

- **Help** — The text you enter here appears in the status area of the enclosing Window whenever this object has focus. If you enter no special text, a default string appears instead. If the enclosing Window has no status area, no text appears.

- **Private Data** — AppBuilder writes out the contents of this area as the value for the PRIVATE-DATA attribute. You can write code to use this data in any way that meets your needs.

- **Generated Code Layout Unit** — Choose whether AppBuilder uses character (the default) or pixel units when it generates the source statements to create and manage this object.

- **Custom Lists** — Custom lists are a way to refer to many objects with a single identifier. The six lists shown always belong to the nearest supporting Frame, not to this object. You can add this object to any or all of the lists, as you choose. If you move this object so that it is supported by a different Frame, you will have to create new associations if you want them.

- **Geometry** — If you prefer to set the XY origin and size using explicit pixel units, you can do that here. You can also choose whether to align to the left or right, but this has no effect other than to change the number used for the X origin.

- **Manual Highlight** — Normally cleared. Setting this box allows you to write code to define a custom highlight effect. Your custom effect is applied whenever this object has input focus.

- **Movable** — Normally cleared. Setting this box makes it impossible for the user to give this object input focus by clicking with the mouse. Instead, the user can use the mouse to drag the object body to a different position within the bounds of the enclosing Frame. Note that if the object has a separate label, as in the case of a Fill–in, the label does not move. The result is that the object and its label can be visually separated from one another.

- **Resizable** — Normally cleared. Setting this box makes it impossible for the user to give this object input focus by clicking with the mouse. Instead, if the selectable box is also set, clicking with the mouse causes handles to appear so that the user can resize the object.

- **Sync with Master** — If this is an alternate layout, choosing this button resets the characteristics of this object in this layout to agree with its characteristics in the main layout. For example, if the width were 12 characters in the alternate layout but 14 in the main layout, choosing Sync would cause the object to be 14 characters wide in both layouts.

## A.5 Colors Selector and Editor



The button shown here is available in the Property Sheet, where relevant.

Almost every object with a visible run-time representation has a foreground and background color that you can set. Whether your settings will override any system defaults is platform-dependent. For example, you can set the title color for a Frame object, but such settings have no effect under MS-Windows.

It is good usability practice to be sparing with color. Too much color, or too many colors, can create a recreational or even childlike impression on the user. This might not be the most desirable outcome, if your application is one for office use. But you need not avoid color changes altogether. Used carefully, special color changes can help focus the user's attention in appropriate ways. This improves the usability of your product and, often, its marketability. Careful use of color can also be a defining element in a corporate identity program, allowing you to create semicustom software packages with little extra effort.

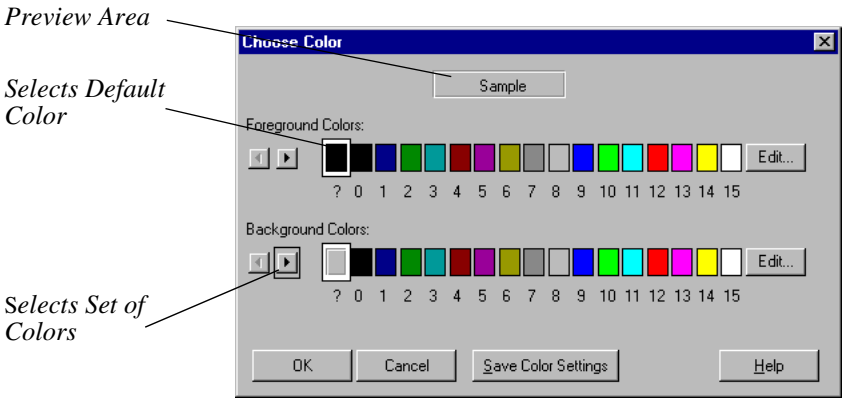
Progress widgets that display data use the foreground and background colors for that purpose. They display their data in the foreground color against the background color. In the case of Data Browser widgets, you can also choose a color for the internal lines separating rows and columns.

Progress widgets such as the Rectangle use the foreground color for their border, and the background color for the fill, if filled.

## A.5.1 Choosing Colors

To set the foreground and background data colors for an object, follow these steps:

1 ♦ Select the object and choose Tools→ Color, or open the property sheet and choose the Colors button (the crayons). The Color Selector dialog box opens:




Sixteen colors remains the baseline standard under MS-Windows, and those colors occupy slots 0 through 15 in the Progress color table. The boxes labeled with the standard Progress symbol for the unknown value (undefined state)—the question mark (?)—represent the default colors for the object. The default colors are those assigned by the object's parent, but are most often the colors shown in the example: black (slot 0) foreground and light gray (slot 8) background.

2 ♦ Click on a new color for the desired context. If you wish to use a color not in the default set, see the "Defining New Colors" section.

3 ♦ If the object whose colors you are changing must be readable (text, instrumentation, graphs, etc.), verify that you have chosen a good combination by examining the text in the Sample rectangle.

Text is always less readable on a monitor than on paper, so it is good practice to change your design choices rather than compromise readability. While good contrast is the most important factor over which you have some control, other factors also come into play, including font size and style. There are also other factors in play over which you as the programmer have no control—the characteristics of the eventual user's monitor being the most important. For this reason, you should choose the most highly readable combinations whenever possible.

### A.5.2 Defining New Colors

Although the defacto industry standard today is 256 colors (8 bits/pixel) and rapidly moving higher, the default MS-Windows color table has only 16 slots (4 bits/pixel). AppBuilder's color table has a total of 256 slots, numbered 0 through 255. Only slots 0 through 15 have colors defined on them by default, and those colors match the default MS-Windows set, as shown in Table A–1.

**Table A–1: Progress Color Table** *(1 of 2)*

| Slot | Progress Color Name | Red-Green-Blue (RGB) Values | | | | | |
|---|---|---|---|---|---|---|---|
| | | Decimal | | | Hex | | |
| 0 | Black | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | Dark Blue | 0 | 0 | 128 | 0 | 0 | 80 |
| 2 | Dark Green | 0 | 128 | 0 | 0 | 80 | 0 |
| 3 | Blue Green | 0 | 128 | 128 | 0 | 80 | 80 |
| 4 | Red | 128 | 0 | 0 | 80 | 0 | 0 |
| 5 | Purple | 128 | 0 | 128 | 80 | 0 | 80 |
| 6 | Olive | 128 | 128 | 0 | 80 | 80 | 00 |
| 7 | Gray | 128 | 128 | 128 | 80 | 80 | 80 |
| 8 | Light Gray | 192 | 192 | 192 | C0 | C0 | C0 |
| 9 | Blue | 0 | 0 | 255 | 00 | 00 | FF |
| 10 | Green | 0 | 255 | 0 | 00 | FF | 00 |

**Table A–1: Progress Color Table** *(2 of 2)*

| Slot | Progress Color Name | Red-Green-Blue (RGB) Values | | | | | |
|---|---|---|---|---|---|---|---|
| | | Decimal | | | Hex | | |
| 11 | Turquoise | 0 | 255 | 255 | 00 | FF | FF |
| 12 | Red | 255 | 0 | 0 | FF | 00 | 00 |
| 13 | Pink | 255 | 0 | 255 | FF | 00 | FF |
| 14 | Yellow | 255 | 255 | 0 | FF | FF | 00 |
| 15 | White | 255 | 255 | 255 | FF | FF | FF |

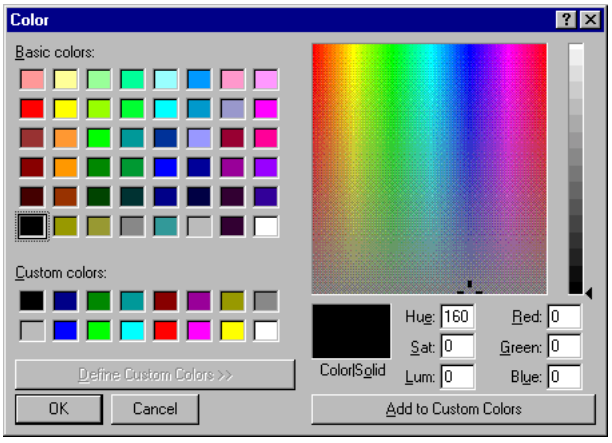
You can increase the colors available to your application—for example, to create a special corporate color scheme—by defining colors on slots 16 through 255 (16 through 127 for character mode), although not all objects can accept colors from slots above 15. You can also change the colors defined for slots 0 through 15 (for example, substituting a green for a red), though for compatibility reasons this is discouraged.

Note that changes made to the color table are permanent the moment they are made. Clicking Cancel does not discard the changes. To restore earlier colors, you must recreate them.

To add a new color to the table, follow these steps:

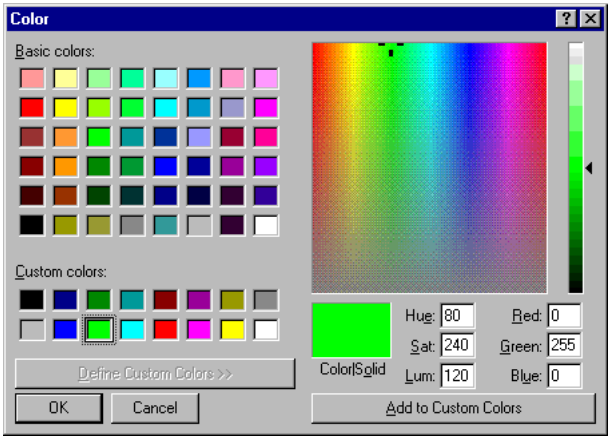
1 ♦ Click the arrowhead button to bring the next set of 16 color slots into view. They are all defined as black, initially.

2 ♦ Double-click the first undefined slot (slot 16, unless you have already defined additional colors for this application, or for another application during this Progress session). It does not matter whether you click the foreground or the background slot—they are really the same slot, and when you define a color, it appears in both places. The standard Windows color-editor dialog box opens:




3 ♦ If none of the existing colors meets your needs, you can modify any or all of the 16 colors in the Custom colors bank. These are not the same 16 colors that appear in the default slots. Click on the color you wish to modify.

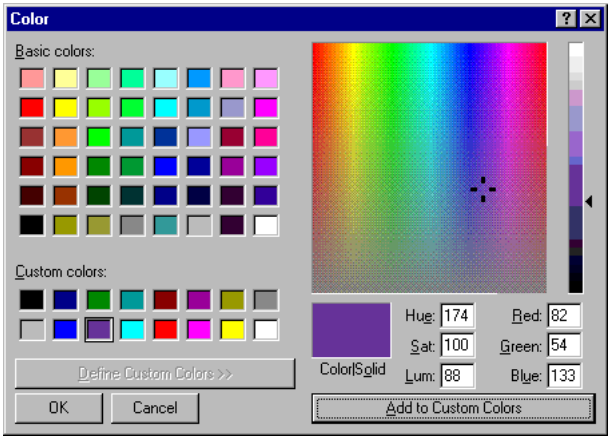
The color in that cell will immediately appear in the large Color/Solid swatch, its HSL and RGB values (HSL and RGB are just different color-notation schemes) appears in the readouts, and the cursors showing where the color is in the color space will reposition themselves appropriately, as shown here:



4 ♦ To create the new color, you can:

- Move the cursors to new positions in the rainbow color space. The larger space represents the qualities of Hue (red, green, etc.) and Saturation/Intensity (bright, rich color versus muted, grayish color). The long, narrow space to the right represents Luminance or Value (in paint or printer's ink, the amount of white or black (but not both) in the color; on a monitor, the amount of energy exciting the phosphor).
- Set the HSL or RGB values explicitly. This is a particularly useful method if you are trying to work to a known standard, or reproduce colors for which you know those values.

5 ♦ When the color is correct, click Add to Custom Colors to preserve it:




6 ♦ If you wish to define additional custom colors, select another Custom Color cell and repeat Step 4 and Step 5. When you are finished, select a color cell and click OK. The color editor dialog box closes, and the selected color is assigned to the color table.

## A.6 Custom Lists Naming




The button shown here is available in the Procedure Settings dialog box.

Custom lists would be called macros in some other languages, such as C. They provide a convenient way to refer to many objects with a single identifier. Each instance of a procedure or procedure-based object, such as a SmartWindow, has its own set of six custom lists. When the preprocessor encounters a reference to a custom list in source code, it replaces that reference with all the identifiers associated with that particular list.

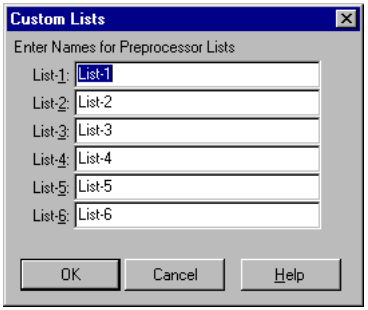



**Figure A–3: Custom Lists Naming Dialog Box**

The dialog box shown in Figure A–3 allows you to give more meaningful names to the lists that belong to a particular procedure. For example, if you create an area within a SmartWindow that has widgets dealing with Order Entry, and you assign them to the first list, you might give that first list the name OrderEntry or Order_Entry (embedded spaces are not allowed in the names you give) so as to more easily keep track of what you are doing.

To change the name of a list, merely type in the new name.

Your change will be reflected in the list of lists in the Procedure Settings dialog box, and in the properties dialog boxes for all the basic widgets subordinate to the procedure. To continue the earlier example, if you place a Pushbutton widget in the SmartWindow, and open its Advanced Properties dialog box, you will now see Order_Entry as the name of the first Custom List.

## A.7 Database Field Selector

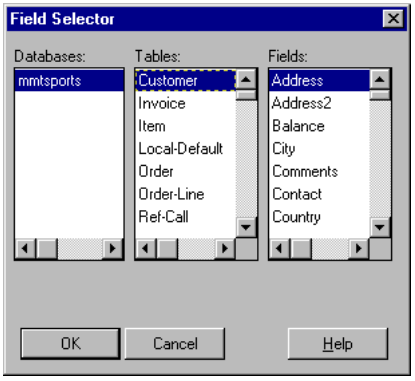


The button shown here is available in the Property Sheet dialog box, where relevant.

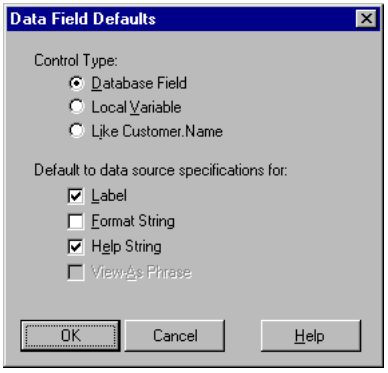
The Database Field Selector dialog box allows you to associate a database field with a data-capturing widget such as a Fill–in, Toggle Box, etc. By choosing appropriate widgets for such associations and adding the necessary additional supporting code, you can create layouts that are more subtly informative and helpful than those where you only use, for example, a Data Browser with its tabular display.

To make an association, follow these steps:

1 ♦ Select the widget and choose Tools → Property Sheet. When the dialog box opens, choose the Database Field button (the browser with a field coming out). The Field Selector dialog box opens:




2 ♦ Select, as appropriate, a database, a table, and a field to be associated with the widget. Choose OK to dismiss the dialog box. The Data Field Defaults dialog box immediately opens in its place:




3 ♦ Choose the control type:

- **Database Field** — Default. Causes the widget to be associated with the database field you chose in Step 2.
- **Local Variable** — Causes the widget to be associated with a local variable.

- **Like** [chosen table.field] — The chosen table.field shown is the one you selected in Step 2. Choosing the option causes the widget to inherit any triggers defined for the database field. If there are no triggers to inherit, you can define those you need. But if the object inherits even one trigger, you cannot define additional ones.

4 ♦ If you choose Database Field or Like, you can also choose whether to inherit some display characteristics from the data source:

- **Label** — Creates a label for the widget based on the field name.
- **Format String** — Ensures that the format of the widget is the same as that of the field.
- **Help String** — Uses the help string associated with the database field.

5 ♦ Choose OK to dismiss the dialog box. Your widget is now associated with the storage unit (database field or local variable) you chose.

## A.8 Method Library Includes List



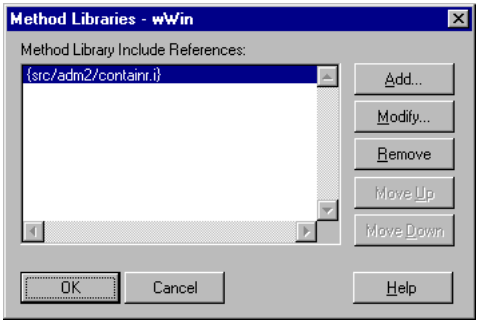
The button shown here is available in the Procedure Settings dialog box.

The Method Libraries dialog box lists all the header files relevant to the current external procedure. If you modify the behavior of a

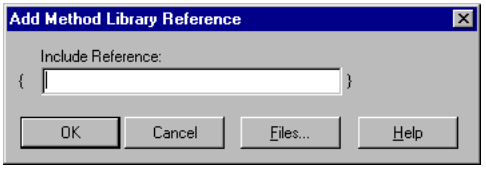
SmartObject, you must add references to any additional header files required by your changes.

To change the list of include files, follow these steps:

1 ♦ Choose Tools → Procedure Settings. When that dialog box opens, click the Method Libraries button (bookcase icon) in the upper-right corner. The Method Libraries dialog box opens:



2 ♦ Click Add. An editing dialog box opens:




3 ♦ Enter the filename to be added to the list, or choose the Files button to browse for the file. Note that the curly bracket delimiters are supplied automatically when you add a reference. You might need to use the Move Up/Move Down buttons to adjust the order in which files are included.

To modify an entry, select it and click Modify. A nearly identical editing dialog box opens, with the entry ready for editing.

To delete an entry, select it and click Delete.

## A.9 Procedure Settings

Broadly speaking, Procedure Settings deal with the nonvisual aspects of objects that are based on external procedures. In the case of a SmartWindow, for example, they are aggregated into seven groups, each represented by a dialog box discussed in this appendix:

- Main (this section)
- Advanced Procedure Settings
- Method Libraries
- Custom Lists
- Temp-Tables
- SmartLinks
- Pages

Other procedure based objects, including other SmartObjects, will have a different number of groups, depending on their nature. For example, a SmartDataObject will not have a Pages button because it is not a SmartContainer and thus does not support a paging context of its own.

To open the main Procedure Settings dialog box for a procedure-based object, you must first open the master object itself, if there is one. Choose File→ Open and select the object or, if the object is a SmartObject instance, click the menu button and choose Edit Master.

When the master object is open and selected, choose Tools→ Procedure Settings, or the Procedure Settings button from the toolbar. A dialog box similar to that in Figure A–4 opens.

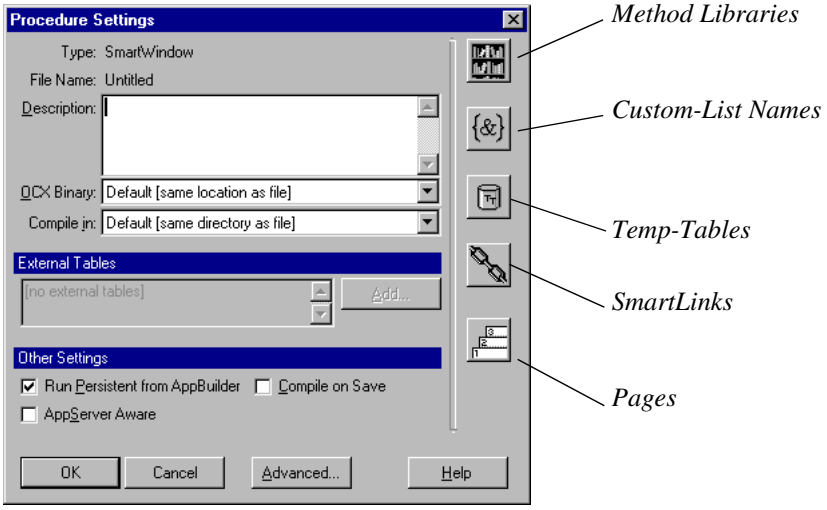



**Figure A–4: SmartWindow Main Procedure Settings Dialog Box**

The main Procedure Settings dialog box is used for all procedure level objects. Some of the fields will therefore not apply to a particular object type, either because the qualities are predefined and fixed, or because they are entirely irrelevant. The settings shown here are those for a SmartWindow:

- **Type** — Reflects the type of subprogram being used for this object. Although you can reset this value in the Advanced dialog box, see the *Progress ADM 2 Guide* for more information.
- **File Name** — Reflects the name of the .w file on disk. This is undefined until you save the object. You can change this value at any time with the usual MS-Windows tools—the Explorer file system interface, for example.
- **Description** — A general-purpose, multi-line field for comments about this object.
- **OCX Binary** — If you include any ActiveX objects in your SmartContainer's population, their properties are saved in a file, by default the .w file itself. If you prefer, you can specify a different file here.
- **Compile in** — By default, the .r object file that results from compilation will be stored in the same directory as the .w source. If you prefer, you can specify a different directory here.

- **Run Persistent from AppBuilder** — Determines whether the object procedure will preserve their data and interface contexts when the RUN statement that invoked them completes execution. Persistent contexts remain allocated until explicitly released, or until the end of the Progress session.

  In general, you should check this box for all SmartObjects other than SmartDialog.

- **Compile on Save** — Determines whether AppBuilder produces an object file when it saves the current source file, rather than recompiling whenever an instance is needed at run time. Checking this box is generally better, but because AppBuilder automatically compiles whenever changes are made, the advantages are not clear-cut.

- **AppServer Aware** — Determines whether AppBuilder includes the header files required for connecting to an AppServer service, making requests of the service, and later disconnects from the service. In an ADM-compliant application, AppServer awareness is a property of the SmartDataObject, and generally not needed by other objects.

## A.10 Properties Window

To open this dialog box, choose Window → Properties Window. It displays certain properties for the currently selected object. The displayed properties are different for different object types. If more than one object is currently selected, it displays only those property types they have in common (generally no more than their dimensions.) Figure A–5 shows the Properties Window for a control frame, such as that automatically supplied by AppBuilder when you place an OCX object.

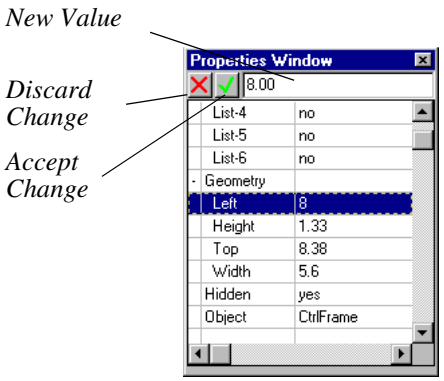



**Figure A–5: Properties Window**

To use this dialog box to make changes to one or more selected objects, follow these steps:

1 ♦ Select the object or objects.

2 ♦ Choose Window → Properties Window. The dialog box opens.

3 ♦ Select the line item you wish to change. If necessary, first click the + sign in the leftmost column to expand and see the values for a category.

4 ♦ The current value appears in the fill–in. Depending on the data type, the fill–in might change to a combo box, or display a button at the right side with the "..." convention that indicates a dialog box will open. Enter the new value for the field accordingly.

5 ♦ Click on the check-mark button to accept the change you made, or on the x-out button to discard it.

6 ♦ When finished, click on the Dismiss button at the right side of the title bar to dismiss the dialog box.

## A.11 SmartLinks Editor



The button shown here is available in the Procedure Settings dialog box.SmartLinks are the well-defined ADM communication pathways between SmartObjects. Although AppBuilder provides Advisor dialog boxes to semiautomatically create links between SmartObjects when you add them to a SmartContainer, there will be times when you will find it desirable or necessary to edit links by hand. The SmartLinks editor shown in Figure A–6 helps you with this task.

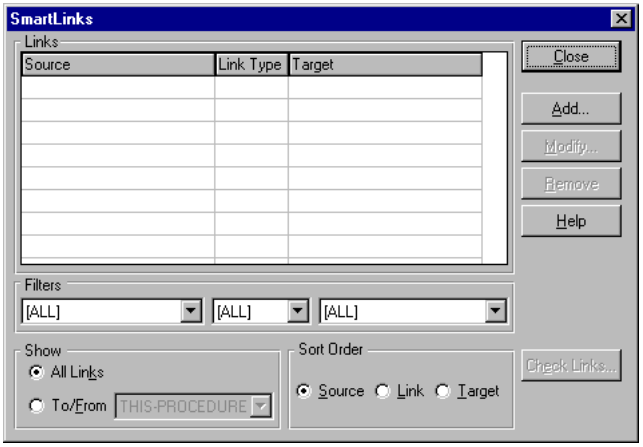
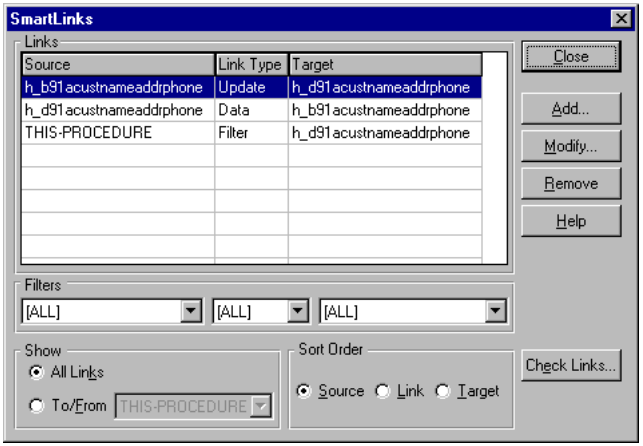



**Figure A–6: SmartLinks Editor Dialog Box**

You can open this editor in either of these ways:

- Click on the menu button of a SmartObject and choose the SmartLinks menu item.
- Choose Tools→ Procedure Settings and then click on the SmartLinks button (the chain-links icon).

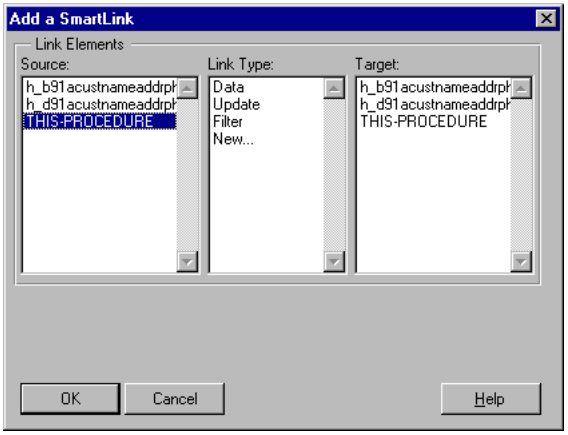
Regardless of where you start when you open the editor dialog box, **it always displays links from the standpoint of the current SmartContainer**. Any references to THIS–PROCEDURE are always references to that current SmartContainer, not to any other SmartObject that might be selected:



The SmartLinks editor dialog box provides extensive filtering capabilities. By default, the editor displays all current links. If the list is confusingly large, you can subset it by filtering on a particular source, link type, or destination, or any combination. You can also choose to sort the list by source, link type, or destination.

To add a new SmartLink, follow these steps:

1 ♦ Choose the Add button. The Add a SmartLink dialog box opens:




2 ♦ All SmartObjects local to the current SmartContainer, including the SmartContainer itself (THIS–PROCEDURE), are listed as possible sources and targets. Select one of them to be the source of the new SmartLink.

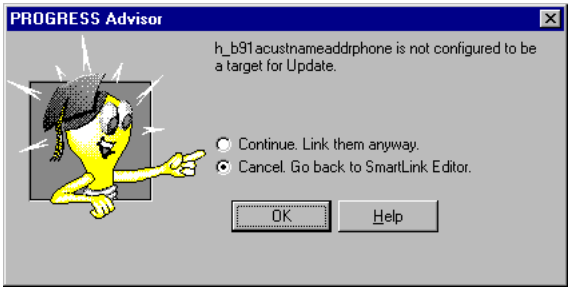
If you are not sure what types of SmartLink are supported by the selected object, choose the Info on Source button. The SmartInfo dialog box opens. See the "SmartInfo"section in this appendix for more information.

3 ♦ Selecting the source of the new SmartLink displays in the LinkType field the SmartLink types for which the selected object can act as source. Choose the type of SmartLink to add.

If you choose New, a dialog box opens and requires you to type in the identifier for the new type of SmartLink. You must be prepared to write code to manage such application-specific link types yourself. See the *Progress ADM 2 Guide* for information about creating and managing new types of SmartLinks.

4 ♦ Choose the target for the new link.

If the target you choose cannot deal with links of the type you selected, an Advisor appears asking that you confirm your choice or elect not to create the link:




5 ♦ Repeat Step 2, Step 3, and Step 4 until you have added all the necessary links. Choose OK to dismiss the Add dialog. Your new links appear in the SmartLinks list.

You can modify an existing link by selecting it and choosing the Modify button. The process is otherwise the same. When you dismiss the Modify dialog box, the link you selected will reflect the changes you made.

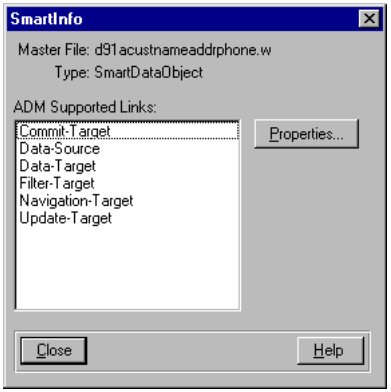
To delete a SmartLink, select it and choose the Remove button.

## A.12 SmartInfo

AppBuilder provides information about a number of SmartObject properties in the two SmartInfo dialog boxes. You can open the main SmartInfo dialog box in either of these ways:

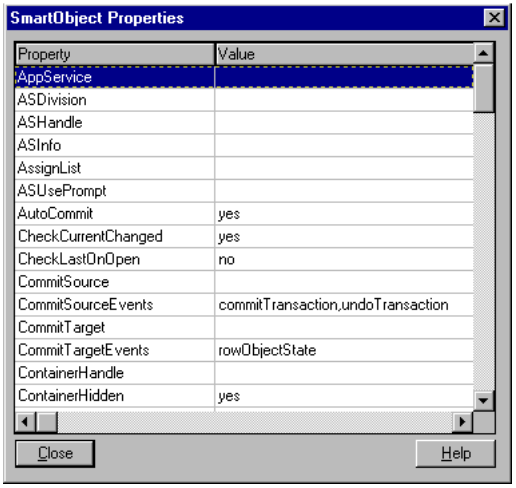
- Select the SmartObject, choose Tools→ Property Sheet, and then click the SmartInfo button (the **i** icon).
- Click the menu button for the SmartObject, and choose the SmartInfo menu item.

The main SmartInfo dialog box opens:




This dialog box provides the filename of the object's master, its class (in the example, SmartDataObject), and a list of supported link types, including directionality.

A number of other properties for the selected SmartObject can be inspected by choosing the Properties button. This read-only dialog box opens to present the properties:



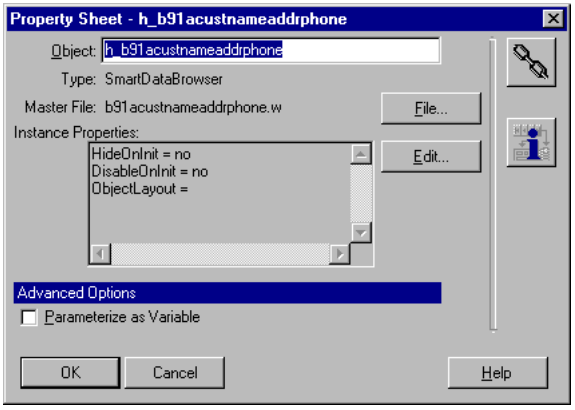

## A.13 SmartObject Properties

A SmartObject has few editable properties of its own, although the basic objects that make up its run-time representation, if any, have all the properties associated with their kind. There is no Advanced Properties dialog box for a SmartObject as such, though there is for any widget the object uses for its run-time representation.

You can open the Property Sheet for a SmartObject other than a SmartWindow, SmartDialog, or SmartFrame by any of these methods:

- Clicking its menu button and selecting the Properties menu item
- Right-clicking on the object and selecting the Properties menu item
- Selecting the object and choosing Tools→ Property Sheet

The Property Sheet dialog box then opens:




This Property Sheet dialog box combines information about both the master object and the particular selected instance. The fields are:

- **Instance handle identifier** — You should change this to more clearly reflect the role of the object in your application.
- **Object type identifier** — This is always a read-only field; it merely identifies the class to which this object belongs.
- **Filename of the master** — Choosing the File pushbutton opens a dialog box that allows you to associate a different master with this instance. Be cautious about using this ability.
- **List of instance properties** — The nature of the properties varies with the object class. See the relevant chapter of this manual for a description of the instance properties pertaining to a particular SmartObject class. Choose the Edit button to change the editable properties.
- **Parameterize as Variable check box** — Normally cleared. Setting this box causes AppBuilder to treat this object as a placeholder for its class rather than as an instance of a specific master object.

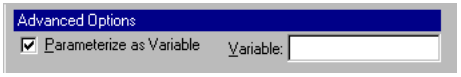
By default, every instance of a SmartObject represents a particular master object with a defined set of behaviors. For example, a given SmartDataBrowser instance is constrained to display a certain set of fields, generally from a particular database, and only those fields. Under some circumstances, you might find it advantageous to be able to treat a SmartObject instance as a design-time placeholder, instead. That would allow you to defer the choice of master until run time, when you could choose, load, and run a situation-specific instance under program control. See the "Creating a SmartObject Placeholder" section for information about how to do this.

### A.13.1 Creating a SmartObject Placeholder

To use an instance of a SmartObject as a design-time placeholder, follow these steps:

1 ♦ Place a SmartObject instance.

2 ♦ Open the object's Property Sheet. Set the Parameterize as Variable check box. A fill–in field opens to the right:




Type in the identifier for the variable you will use to control which master gets loaded at run time.

3 ♦ Select the SmartContainer that supports the new placeholder. Open the Section editor to the Definitions area, and define the variable. Add additional code in the appropriate section to resolve the reference.

## A.14 String Attributes



The button shown here is available in the Property Sheet for widgets for which it is relevant.

This dialog box takes different forms, depending on the object. For example, Figure A–7 shows the String Attributes dialog box for a Static Text object, and Figure A–8 the same dialog box for a Combo Box object.

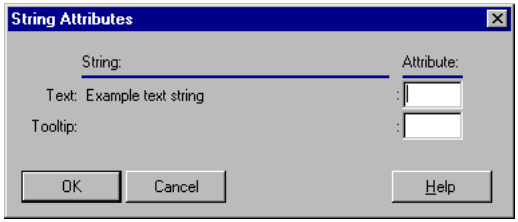



**Figure A–7: String Attributes Dialog Box for Static Text**

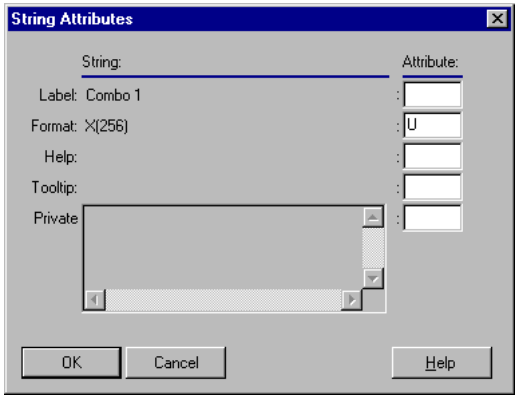



**Figure A–8: String Attributes Dialog Box for a Combo Box**

The string attributes you include here govern display and translation characteristics for the strings. Note that you also set attributes for the associated help and tooltip strings here.

As long as they do not contradict one another, you can include several attributes. The available attributes are:

- **L R C T** — Orientation: Left, Right, Centered, or Trimmed. Use only 1 of these. The default varies with the role of the string.
- **U** — Untranslatable. Add this marker to prevent the Progress translation utility from processing the string. By default, the utility processes all strings.
- **9999** — The INTEGER size of the space to be allocated in the literals table for this string. The default size is the length of the string itself, and the maximum size is 5120 characters. If the string is to be translated, and you believe the translation will produce a longer string, you should allow for that increased space here.

To specify one or more values, simply type them into the appropriate field.

## A.15 Traversal-Path Editor



The buttons shown here are available in the Property Sheet dialog box for Frame and Dialog Box objects.

Under MS-Windows, repeatedly pressing the **TAB** key causes the input focus to shift from one widget to another within a Frame. The default order in which the widgets are visited is left-to-right, top-to-bottom. You can change this default order using the Tab Editor. The Tab Editor is available through the property sheet for any Frame or Dialog widget, including the widgets managed by SmartFrames and SmartDialogs.

Choose the Tab Editor button (back-and-forth arrows) to open the Editor dialog box, shown in Figure A–9.

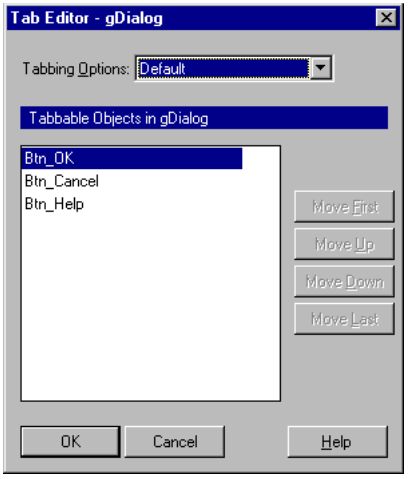



**Figure A–9: Tab Editor Dialog Box**

The three areas of interest are:

- The combo box that displays the current Tabbing Option
- The list of widgets that will be traversed
- The buttons that control how the items in the list are ordered

Initially, the Tabbing Option is set to Default. In addition to the Default ordering, there are four predefined orderings, plus Custom. One of the predefined orderings will be the same as the Default order. In English-speaking countries among others, this is the Left-to-Right By Rows order.

To impose one of the predefined traversal orderings, select it from the list of options. The order of the widgets in the objects list changes accordingly.

To define a special traversal order, follow these steps:

1 ♦ Select Custom as the Tabbing Option. This enables the four Move buttons.

2 ♦ Use the Move buttons to reorder the widgets in the objects list. The items will be visited in the order in which they appear in this list.

3 ♦ Choose OK.

## A.16 Temp-Tables Maintenance



The button shown here is available in the Procedure Settings dialog box. The editor is also called automatically when you choose to define a temp-table in a SmartDataObject.

Temp-tables are temporary (scratch) tables. You might choose to create and use a temp-table to avoid the overhead of manipulating a large original table. By selecting out only those records that are of interest, and creating from that subset a temp-table, you can make subsequent processing faster or more convenient by some large factor.

Temp-tables have these characteristics:

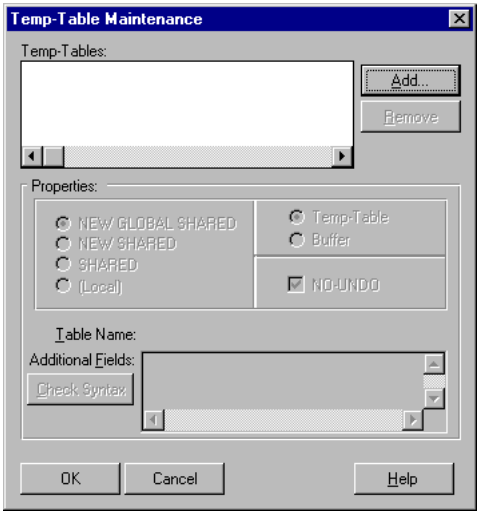
- **Single-user** — Unlike regular database tables, you cannot make the same temp-table available to more than one user. They are session-bound.
- **Temporary** — Temp-tables never outlast the session in which they are created, and might have a much shorter lifespan than that. You can choose to make the table:
  - **Local** – (Default). Creates a new local table. A local table is available only to the defining procedure, and is destroyed when that procedure terminates.
  - **Shared** – Specifies that the table is not to be created, but is being shared by the calling procedure that did create it, within the current session.
  - **New Shared** – Creates a new table and allows it to be shared by called procedures in the same session, if they declare the table Shared.
  - **New Global Shared** – Creates a new table and allows it to be shared by all other procedures in the session that also declare the table New Global Shared. Note that although all procedures must use the New syntax, the table is allocated only once.

Temp-tables can be declared NO–UNDO, which improves performance by eliminating the overhead of journaling. In AppBuilder, temp-tables are created NO–UNDO by default.

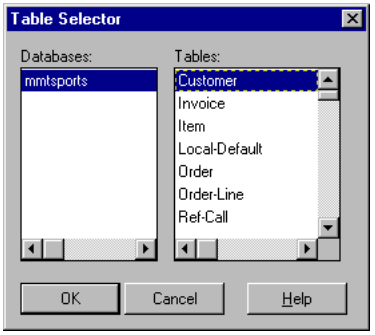
Temporary table storage can be allocated in memory or on disk. There are some differences in the characteristics of a table depending on where it resides. See the *Progress Programming Handbook* for a discussion of these differences.

To allocate a temp-table, follow these steps:

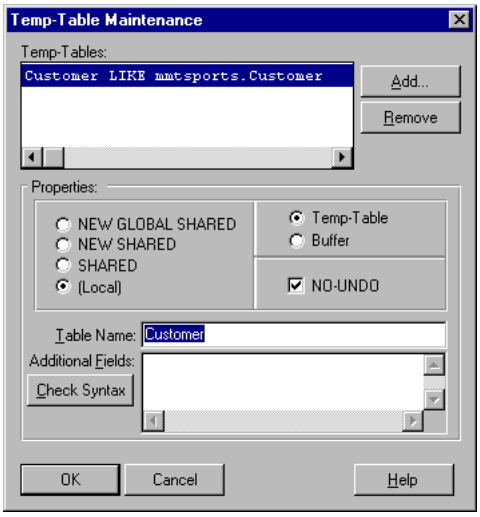
1 ♦ Choose Tools → Procedure Settings. When that dialog box opens, click the Temp-Tables button (the drum icon). The Temp-Table Maintenance dialog box opens:




2 ♦ Click Add to create a temp-table. The Table Selector dialog box opens, showing the currently connected databases:



3 ♦ Choose a database, if necessary, and the table you wish to copy from. Click OK. The Selector dialog box closes and your selection appears in the Temp-Table Maintenance dialog box:




Note that AppBuilder always defines the temp-table as being LIKE the reference table.

4 ♦ Modify the default properties, if they do not suit your purposes, and click OK. The default properties are:

- Allocation on disk (temp-table) rather than memory (buffer)
- Local scope rather than Shared or Global Shared
- NO–UNDO
- Name same as reference-table name. You might wish to change this to more clearly identify the reason you are using a temporary copy of the table.
- No additional fields. You can define additional fields, but they are not equivalent in all ways to the actual fields in the database records. You cannot refer to additional fields in a query, for example.

Note that this editor does not create code to populate the table. You must do that by hand, if the object in your application expects to find records in the table.

## A.17 Text Style Selector and Editor



The button shown here is available in the property sheet of most widgets. It also appears in the Column Editor dialog box for Data Browser widgets, where you can set the font for column labels separately.

AppBuilder provides a fonts table similar in principle to its color table. Rather than specify each change of typestyle–font, weight and slant, point size, and attributes such as underlining–on the fly, you define them in advance, with each definition occupying a slot in the fonts table. You then apply all characteristics at once by specifying a slot number.

As with the use of color, be sparing with font changes. Although at one time it was very common to use many sizes and styles of type in a single composition, today too many font changes can create an unfavorable impression on the user. Used carefully, special changes in the type can help focus the user's attention in a good way. Because of readability issues, fonts need to be selected with even more care than colors if they are to give the desired result.

Many companies use one or more particular fonts as part of their corporate identity program. If you are able to use those fonts too, without compromising readability, you might be able to gain a competitive advantage when marketing your software to such a company.

Typestyle selections apply to all text displayed in connection with a given widget. For example, you cannot choose one style for the column labels on a Browse widget and another style for the data it displays. Whatever style you select will be applied to both.

### A.17.1 Choosing a Typestyle

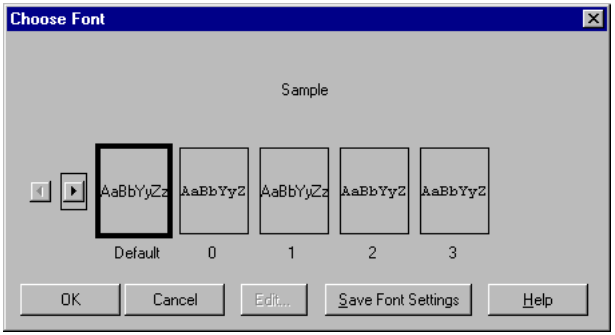
You can assign typestyles on a per-widget basis. Each widget that displays text, whether as labels, data, or both, can have its own typestyle. Both labels and data will be displayed in the selected style; you cannot select one style for the labels and another for the data in AppBuilder. You can write code to do that if it is important to your design.

Generally speaking, you should choose the same style for most if not all of the widgets in your application. Make each extra typestyle count.

To assign a typestyle to a widget, follow these steps:

1 ♦ Open the property sheet for the widget. If the widget is being displayed and managed by a SmartObject, you must click the object's menu button and choose Edit Master to get access to that widget.

2 ♦ When the Property Sheet dialog box opens, click the typestyle button (the A*A*a icon). The Choose Font dialog box opens:



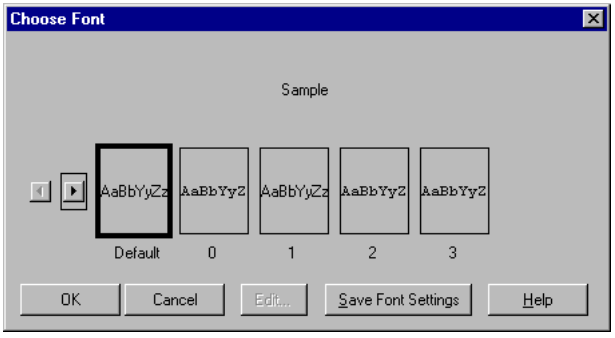

3 ♦ If necessary, click on the arrowhead buttons until the font sample you want to use is in view. Click it, and the thick outline appears, indicating selection.

4 ♦ Click OK to dismiss the Choose dialog box. Click OK to dismiss the property sheet. You should see your choice of typestyle reflected in the widget.

## A.17.2 Defining a Typestyle

The steps in defining a new typestyle or editing an existing one are the same. To define a new typestyle, follow these steps:

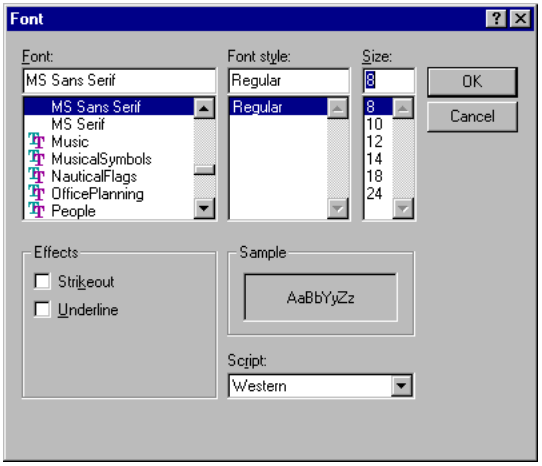
1 ♦ Open the property sheet for a widget. Click on the typestyle button (the A*A*a icon). The Choose Font dialog box opens:




2 ♦ Unless you also want to change the font being used for the current widget, make a note of which slot is current so that you can restore it before you exit this font-editing session.

3 ♦ Slots 0 through 7 have predefined styles that are not editable. You can assign styles to slots 8 through 255. Click the right arrowhead button until slot 8 appears.

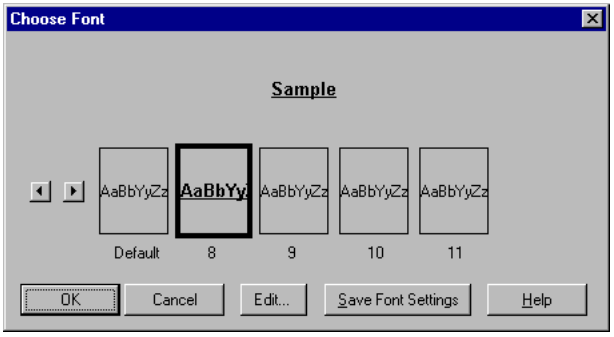
4 ♦ Double-click slot 8. The Font dialog box opens:




5 ♦ Choose a face, a weight and slant, a point size, and attributes (underline, etc.). Unless you are creating an application for a market that requires a special encoding (Greek, Cyrillic, etc.), be sure the Script setting is Western.

**CAUTION:** Remember that the fonts being presented to you by this dialog box are those you have installed on your development system. The fonts you choose here will not necessarily be available to the users of your application unless you provide them as part of your software distribution package. Unless you are prepared to do that, you should choose only those fonts that you know are supplied with the operating system.

6 ♦ Click OK to dismiss the Font dialog box. Your choices are reflected in the sample text in the slot. In this example, it shows the result of choosing 10pt Arial Bold, with underlining:



7 ♦ Unless you want your definition discarded at the end of the current AppBuilder session, click Save Font Settings to make your definition permanent.

8 ♦ Click on the slot you noted in Step 2. If you want to change the typestyle for the current widget, click on that slot instead.

9 ♦ Click on OK to dismiss the Choose Font dialog box. Click on OK to dismiss the Property Sheet dialog box.

# B

# Multiple Layouts

This appendix describes how to use multiple layouts to build your applications. In addition, it explains how to use AppBuilder to run and test applications for a character-based user interface.

This appendix discusses the following topics:

- Creating multi-mode AppBuilder applications by using multiple layouts
- Testing AppBuilder character-mode applications
- Using multiple layouts with SmartObjects

## B.1 Why Use Multiple Layouts?

The following list identifies just a few of many different usages you might consider to employ multiple layouts:

- If you plan to run an application in different run-time environments such as Windows and a character-based terminal, you might want to adjust the application's interface for each environment. If you are using SmartObjects, you might want to build a single master file for a SmartDataViewer or SmartContainer that provides a choice of visualizations. One solution is either to maintain a separate version of the application for each environment or to create a separate SmartObject instance for each visualization. This approach requires that you maintain and deploy multiple sets of source code.

- Another solution is to use the AppBuilder's capability to support multiple layouts in a single procedure file. The advantage of this approach is that you maintain and deploy one set of source files for an application. When you run a procedure file that contains multiple layouts, the procedure file adjusts its interface to suit the current run-time environment. You can also switch between layouts at run time.

- Multiple layouts can help you internationalize an application. For example, you might have an application that helps you to distribute products globally. Since different countries have different address formats and information requirements, your application can employ mulitple layouts to effectively address this addressing issue.

For more information on multiple layouts and international considerations, see the *Progress Internationalization Guide*.

## B.2 What Is a Layout?

A *layout* can be thought of as a collection of objects and associated property settings. These property settings determine how the objects appear when you run the application. Each layout has a name and an associated run-time expression. Run-time expressions evaluate to TRUE or FALSE, and determine which layout is presented during a specific session. Typically the same set of run-time expressions will appear for all procedure files in an application. For SmartObjects, layouts are typically associated with values of the ObjectLayout instance property.

When you run a procedure file, it evaluates the run-time expressions (if any) for the available layouts and, if a layout's expression evaluates to TRUE, uses that layout. For a SmartObject, this run-time expression determines the setting of the DefaultLayout property. If the SmartObject has no ObjectLayout property setting, it uses the DefaultLayout property setting to determine the layout. When the procedure file determines that it will use a particular layout,

the procedure file executes for that layout a CASE statement that contains all of the property settings assigned for the layout.

The AppBuilder generates a CASE statement that contains an entry for each layout in the procedure file. The CASE statement is contained within the internal procedure named {&LAYOUT–VARIABLE}s. This procedure name resolves to the name of the procedure file's window, with "–Layouts" appended. For example, if the window name is WINDOW–1, the name of this procedure is WINDOW–1–Layouts. For more information about how layouts are implemented within a procedure file, see the "How Multiple Layouts Work: A Look at the Code" section.

At design time, a layout is visually manifested in the design window. When you switch between layouts in the design window, the AppBuilder alters the appearance of the design window to match the characteristics of the new layout.

## B.3 Two Types of Layouts

A procedure file can contain two types of layouts:

- **Master layout** — The main layout of the application. A procedure file has one master layout.
- **Alternate layout** — An additional layout for a specific run-time environment. A procedure file can have multiple alternate layouts.

The following sections describe the characteristics of these types of layouts.

### B.3.1 Master Layout

The master layout is what the AppBuilder regards as the main layout of the procedure file. When you first create a procedure file, you see the master layout in the design window. If you do not create any alternate layouts, you see the master layout when you run the procedure file. You also see the master layout if the run-time expressions for all of the alternate layouts evaluate to FALSE.

It is helpful to think of the master layout as the compile-time layout, or the layout as it appears to the Progress Compiler. Unlike other layouts, the master layout does not necessarily require additional property assignments. Alternate layouts, in contrast, are implemented solely through the use of run-time property assignments.

The master layout consists of the following elements:

- All of the static widget definitions generated by the AppBuilder in the procedure file
- All of the property assignments, if any, in the Runtime Attributes and AppBuilder Settings code section
- The procedure file's window (a dynamic widget) and any ActiveX control frames
- A CASE statement entry for "Master Layout"

The master layout **never** has a run-time expression associated with it.

If desirable, you can run the "Master Layout" entry or any other CASE statement entry from within the procedure file. This allows you to alternate programmatically between layouts at run time. For more information, see the "Switching Between Layouts at Run Time" section.

## B.3.2 Alternate Layout

An alternate layout is a layout that you create for a specific run-time instance. For each alternate layout that you create, you supply a name, an optional run-time expression, and the emulation style (graphical or character). At run time, the procedure file tests any run-time expression, and if the expression evaluates to TRUE, the procedure file uses that layout, executing the appropriate CASE statement entry. If the procedure file is a SmartObject that has an ObjectLayout property setting, the SmartObject uses the specified layout, executing the appropriate CASE statement entry. The procedure file thus determines the layout before enabling or displaying any widgets. For more information about creating an alternate layout, see the "Creating Alternate Layouts" section.

### Standard Layouts

Standard layouts are alternate layouts with predefined run-time expressions. The AppBuilder predefines these expressions for three primary Progress run-time environments: character, MS-Windows, and MS-WIN95. Table B–1 lists the standard layouts.

**Table B–1: Standard Layouts and Run-time Expressions**

| Layout Name | Run-time Expression |
|---|---|
| Standard Character | SESSION:DISPLAY–TYPE = 'TTY':U |
| Standard MS-Windows | SESSION:WINDOW–SYSTEM = 'MS–WINDOWS':U |
| Standard Windows 95 | SESSION:WINDOW–SYSTEM = 'MS–WIN95':U |

If you decide to use a standard layout, you cannot change the layout's name or run-time expression.

### Layouts at Design Time

When you create or select an alternate layout, the AppBuilder changes the design window to assume the alternate layout's characteristics. The AppBuilder also reflects the characteristics of the layout in the property sheets of its widgets. Once widgets are in an alternate layout, you can then manipulate them within the layout or change their properties. Within an alternate layout, you can modify some but not all of the widgets' properties. For more information, see the "Alternate Layout Limitations" section.

In general, the changes that you make within an alternate layout visually affect that layout only, but there are some exceptions. For more information, see the "Modifying Layouts–Layout Inheritance" section.

For example, suppose that you start with the master layout, as shown in Figure B–1.

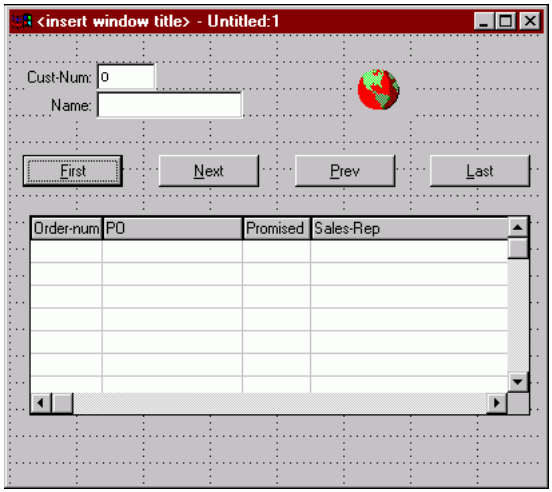



**Figure B–1: Master Layout**

Now, suppose that you choose the Standard Character layout option in the Alternate Layout dialog box. The AppBuilder now changes the design window to look like the one in Figure B–2.

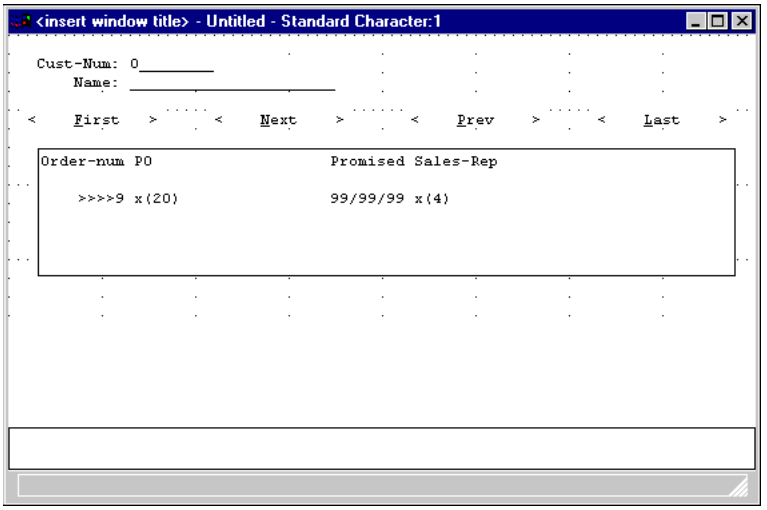



**Figure B–2: Character Alternate Layout**

Notice that image disappears and the buttons appear as text. Also, the grid changes to reflect the number of lines in a character terminal. For more information, see the "Simulating Character Applications" section.

All property sheets for the widgets in this layout reflect the characteristics of the new layout. In this example, the property sheet of the window reflects a change in size: character windows are not resizable and have a default size of 80 x 21 character units. Also, since the image does not appear in this layout, you cannot access its property sheet from within the layout. However, you can use the List Objects dialog box to access the image's property sheet. For more information about the List Objects dialog box, see the "Tool Bar" section in Chapter 2, "AppBuilder Interface."

In general, when you initially create an alternate layout, the AppBuilder tries to keep all objects and properties unchanged from the prior layout.

### Alternate Layout Scope

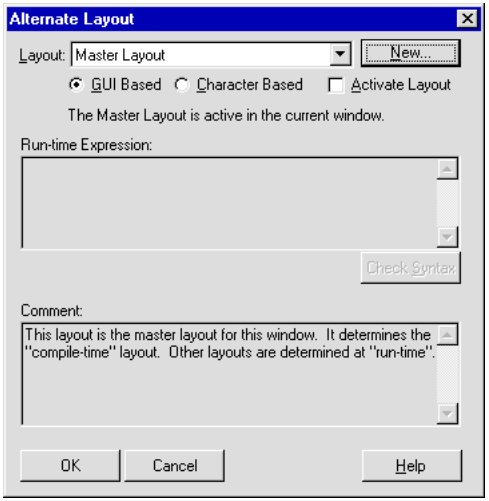
When you create a layout, it is available for all procedure files that you edit in the AppBuilder session. You do not have to define the layout individually for every procedure file that uses it.

The AppBuilder maintains a cumulative list of nonstandard layouts for each AppBuilder session. This list expands as you edit procedure files with previously defined layouts. Restarting the AppBuilder resets the list to include only the standard layouts and those alternate layouts used by the procedure files that you open.

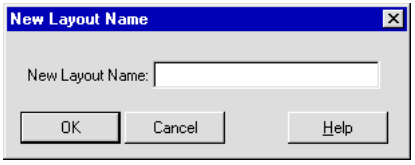
## B.4 Creating Alternate Layouts

Follow these steps to create an alternate layout:

1 ♦ Choose Layout→ Alternate Layout from the AppBuilder main menu. The Alternate Layout dialog box appears:




2 ♦ Choose the New button. The New Layout Name dialog box appears:




3 ♦ Type the name of the layout, then choose OK. The AppBuilder returns to the Alternate Layout dialog box.

4 ♦ Activate the GUI-based radio button to create a graphical look for the design window. Activate the Character-based radio button to simulate a character design window.

5 ♦ Type the run-time expression to test in the Expression panel, then choose OK.

Any valid Progress expression that resolves to a LOGICAL value is acceptable. However, if you use a character string within the expression, place the string in single quotes (' ') not double quotes (" "). If you need to use embedded quotation marks, "escape" the quotation marks with the tilde (~) character. This convention helps the AppBuilder to generate code that avoids compiler ambiguities.

When you enter an expression, you can optionally enter a comment that describes the alternate layout. The AppBuilder places this comment in the Runtime Properties and AppBuilder Settings section of the procedure file, where other developers can see it. Enter the comment by typing it in the Comment fill-in field in the Custom Layout dialog box.

You have finished creating the alternate layout. The AppBuilder redisplays your design window, with the name of the alternate layout in the title bar.

**NOTE:** The Alternate Layout dialog box allows you to add, delete, or modify only one alternate layout at a time.

## B.5 Selecting a Predefined Alternate Layout

Follow these steps to select a layout that has already been defined:

1 ♦ Choose Layouts→ Alternate Layout from the AppBuilder main menu. The Alternate Layout dialog box appears.

2 ♦ Select the standard alternate layout that you want from the Layout combo box.

3 ♦ Choose OK.

If you select a standard layout, you cannot change its expression or name. However, you can modify the expression and name of a nonstandard alternate layout.

## B.6 Modifying Layouts–Layout Inheritance

In general, if you make a change to the master layout, such as adding a button, the AppBuilder automatically adds the same button to all of the other layouts in the application. Similarly, if you change the size (or other property) of a widget in the master layout, that change is also reflected in the other layouts in the application. This process of automatically passing on the master layout's characteristics to alternate layouts is called *layout inheritance*.

Within each alternate layout, you can explicitly override inheritance on a property-by-property basis for each widget. You can do this by manipulating the widgets or by changing their properties. For example, if you change the color of a button in an alternate layout, it *disinherits* that property from the master. No matter how you change the color of the button in the master, the alternate layout is unaffected, and vice versa.

**NOTE:** Not all properties can be disinherited. Some properties can only have one value for all layouts. Such properties are insensitive in the widget's property sheet and cannot be changed.

You can also override layout inheritance on a widget-by-widget basis by inserting a widget into an alternate layout or by removing a widget from the alternate layout.

You can re-establish layout inheritance for a widget on a property-by-property basis. To do this, access the widget's advanced property sheet and choose the Sync with Master button. For more information, see the "Re-establishing Layout Inheritance" section in this appendix.

## B.6.1 Layout Inheritance Behavior at Run Time

At run time, inheritance always occurs from the master layout to all other layouts in the procedure file. However, inheritance is conditional in that a character layout does not inherit all of the widgets or properties in a graphical layout, and inheritance can be explicitly overridden.

### Character Layout Inheritance

If Character Progress does not support a particular widget, a character layout cannot inherit the widget even if it appears in the master layout. Similarly, if Character Progress does not support a particular property for a widget, the character layout cannot inherit the property. For example, the height of Fill-ins, Combo Boxes, Toggle Boxes, Pushbuttons, and Text widgets is always 1 in a character interface.

When you create a character alternate layout, the AppBuilder changes the appearance of the design window and the widgets within the design window to fit within the limitations of a character environment. Thus, buttons can only be one line high, and the character window has a default width of 80 character units and a height of 21 character units. For more information, see the "Simulating Character Applications" section.

### Overriding Layout Inheritance

If you change a property of a widget in an alternate layout, that layout no longer inherits the property from the master layout unless you explicitly re-establish layout inheritance. Also, if you insert a widget into an alternate layout, or delete a widget from an alternate layout, you disinherit the widget from the master layout.

There are two special cases:

- If you change either coordinate of a widget's position, the other coordinate is disinherited as well. You must synchronize both coordinates with the master layout to re-establish layout inheritance.
- If you change either dimension of a widget, the other dimension is disinherited as well. You must synchronize both dimensions to re-establish layout inheritance.

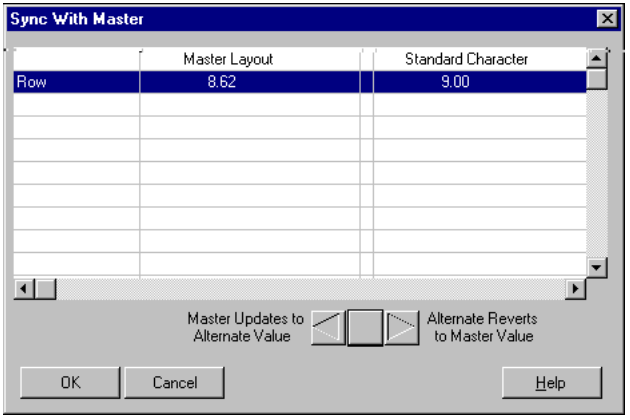
## B.6.2 Layout Inheritance Behavior at Design Time

When you create a new layout at design time, it inherits the characteristics of the current layout, even if the current layout is not the master layout. The new layout also disinherits each property that the current layout disinherits from the master. However, at run time, layout inheritance occurs between the new alternate layout and the master layout.

## B.6.3 Re-establishing Layout Inheritance

Follow these steps to re-establish layout inheritance:

1 ♦ While in an alternate layout, open the advanced Property Sheet for the widget whose layout inheritance is to be re-established, then choose the Sync With Master button. The Sync With Master dialog box appears:




This dialog box displays, for the current widget, all of the property discrepancies between the master layout and the alternate layout. You can re-establish layout inheritance in two ways:

- Have the master adopt the properties of the alternate layout. This changes the widget's definition in the master layout.
- Have the alternate layout adopt the properties of the master layout. This changes the properties of the alternate layout to match those of the master layout.

2 ♦ Select the property to synchronize.

3 ♦ Choose the button that indicates how to synchronize the property:

- If you choose the Alternate Reverts to Master Value button, a greater than symbol (>) appears to the right of the Master Layout Column. This indicates that the alternate layout adopts the property setting from the master layout.
- If you choose the Master Updates to Alternate Value button, a less than symbol (<) appears to the right of the Master Layout Column. This indicates that the master layout adopts the property setting from the alternate layout.
- To remove any synchronization choice, choose the center button between the Alternate Reverts to Master Value button and the Master Updates to Alternate Value button.

If neither symbol appears, the properties will not be synchronized.

An alternate technique to indicate how you want to synchronize a property is to repeatedly double-click on the property in the selection list. As you double-click on the property, the AppBuilder rotates among the greater than (>) symbol, the less than (<) symbol, and no symbol.

4 ♦ When you finish specifying how to synchronize the properties, choose OK.

The AppBuilder synchronizes the properties according to the button you choose. The AppBuilder also re-establishes inheritance for the property, with two exceptions:

- If you change either coordinate of a widget's position, the other coordinate is disinherited as well. You must synchronize both coordinates with the master layout to re-establish layout inheritance.
- If you change either dimension of a widget, the other dimension is disinherited as well. You must synchronize both dimensions with the master layout to re-establish layout inheritance.

## B.7 How Multiple Layouts Work: A Look at the Code

Within a procedure file, the AppBuilder supports an alternate layout with the following code:

- Run-time Properties and AppBuilder Settings code section
- Layout Cases ({&LAYOUT–VARIABLE}s) internal procedure
- Enable_UI internal procedure

### B.7.1 Run-time Properties and AppBuilder Settings

For each alternate layout, the AppBuilder places an IF statement in the Run-time Properties and AppBuilder Settings code section. This IF statement checks to see whether the layout's run-time expression evaluates to TRUE. If so, this statement runs the layout case's internal procedure, passing to it the name of the layout as an input parameter.

For example, the multi-layout run-time adjustments code appears as follows:

```
/* _MULTI-LAYOUT-RUN-TIME-ADJUSTMENTS */

/* LAYOUT-NAME: "Standard Character"
   LAYOUT-TYPE: CHARACTER
   EXPRESSION:  SESSION:DISPLAY-TYPE = 'TTY':U
   COMMENT:     This layout is the standard layout specification for
                a customized Character based terminal.  It is usually
                selected to modify a window that has a GUI based
                master layout.
                                                                      */
IF SESSION:DISPLAY-TYPE = 'TTY':U  THEN
  RUN C-Win-layouts (INPUT 'Standard Character':U) NO-ERROR.

/* LAYOUT-NAME: "Standard MS Windows"
   LAYOUT-TYPE: GUI
   EXPRESSION:  SESSION:WINDOW-SYSTEM = 'MS-WINDOWS':U
   COMMENT:     This layout is the standard layout specification for
                a customized MS Windows window.  It is usually
                selected to modify a window that needs to have a
                standard "MS Windows" look.
                                                                      */
ELSE IF SESSION:WINDOW-SYSTEM = 'MS-WINDOWS':U  THEN
  RUN C-Win-layouts (INPUT 'Standard MS Windows':U) NO-ERROR.

/* LAYOUT-NAME: "Standard Windows 95"
   LAYOUT-TYPE: GUI
   EXPRESSION:  SESSION:WINDOW-SYSTEM = 'MS-WIN95':U
   COMMENT:     This layout is the standard layout specification for
                a customized Windows 95 window.  It is usually
                selected to modify a window that needs to have a
                standard "Windows 95" look.
                                                                      */
ELSE IF SESSION:WINDOW-SYSTEM = 'MS-WIN95':U THEN
  RUN C-Win-layouts (INPUT 'Standard Windows 95':U) NO-ERROR.

/* END-OF-LAYOUT-DEFINITIONS */

IF SESSION:DISPLAY-TYPE = "GUI":U AND VALID-HANDLE(C-Win)
THEN C-Win:HIDDEN = no.
```

If the expression SESSION:WINDOW–SYSTEM = 'MS–WINDOWS':U evaluates to TRUE, this IF statement passes the input parameter 'Standard MS Windows' to the layout cases internal procedure. For SmartObjects, the DefaultLayout property is set. This is used to run the appropriate layout case later, when the SmartObject is initialized.

Thus, for SmartObjects, the multi-layout run-time adjustments code appears as follows:

```
/* _MULTI-LAYOUT-RUN-TIME-ADJUSTMENTS */

/* LAYOUT-NAME: "Standard Character"
   LAYOUT-TYPE: CHARACTER
   EXPRESSION:  SESSION:DISPLAY-TYPE = 'TTY':U
   COMMENT:     This layout is the standard layout specification for
                a customized Character based terminal.  It is usually
                selected to modify a window that has a GUI based
                master layout.
                                                                      */
IF SESSION:DISPLAY-TYPE = 'TTY':U  THEN
  DYNAMIC-FUNCTION('setDefaultLayout':U, 'Standard Character':U) NO-ERROR.

/* LAYOUT-NAME: "Standard MS Windows"
   LAYOUT-TYPE: GUI
   EXPRESSION:  SESSION:WINDOW-SYSTEM = 'MS-WINDOWS':U
   COMMENT:     This layout is the standard layout specification for
                a customized MS Windows window.  It is usually
                selected to modify a window that needs to have a
                standard "MS Windows" look.
                                                                      */
ELSE IF SESSION:WINDOW-SYSTEM = 'MS-WINDOWS':U  THEN
  DYNAMIC-FUNCTION('setDefaultLayout':U, 'Standard MS Windows':U) NO-ERROR.

/* LAYOUT-NAME: "Standard Windows 95"
   LAYOUT-TYPE: GUI
   EXPRESSION:  SESSION:WINDOW-SYSTEM = 'MS-WIN95':U
   COMMENT:     This layout is the standard layout specification for
                a customized Windows 95 window.  It is usually
                selected to modify a window that needs to have a
                standard "Windows 95" look.
                                                                      */
ELSE IF SESSION:WINDOW-SYSTEM = 'MS-WIN95':U THEN
  DYNAMIC-FUNCTION('setDefaultLayout':U, 'Standard Windows 95':U) NO-ERROR.

DYNAMIC-FUNCTION('setLayoutOptions':U, "Master Layout,Standard
Character,Standard MS Windows,Standard Windows 95":U).
```

### B.7.2 Layout Cases Internal Procedure

This internal procedure contains a CASE statement that has an entry for each layout in the procedure file. The CASE statement entry contains the run-time assignments for the layout. The Preprocessor reference {&LAYOUT–VARIABLE} resolves to the name of the window in the procedure file prefixed to "–layout". Following is a sample layout case's internal procedure. Note that the variable name {&LAYOUT–VARIABLE} is usually the same as {&WINDOW–NAME}–layout. The layout cases internal procedure is named {&LAYOUT–VARIABLE}s, which in this sample resolves to C–Win–layouts:

*(1 of 2)*

```
PROCEDURE C-Win-layouts:
  DEFINE INPUT PARAMETER layout AS CHARACTER                      NO-UNDO.
  DEFINE VARIABLE lbl-hndl AS WIDGET-HANDLE                       NO-UNDO.
  DEFINE VARIABLE widg-pos AS DECIMAL                             NO-UNDO.

  /* Copy the name of the active layout into a variable accessible to   */
  /* the rest of this file.                                             */
  C-Win-layout = layout.

  CASE layout:
    WHEN "Master Layout" THEN DO:
      ASSIGN
         &IF '{&WINDOW-SYSTEM}' NE 'TTY':U &THEN
         C-Win:HIDDEN                                    = yes &ENDIF
         C-Win:HEIGHT                                    = 16.

      ASSIGN
         Btn_Done:HIDDEN IN FRAME DEFAULT-FRAME          = yes
         Btn_Done:HEIGHT IN FRAME DEFAULT-FRAME          = 1.14
         Btn_Done:ROW IN FRAME DEFAULT-FRAME             = 3.62
         Btn_Done:HIDDEN IN FRAME DEFAULT-FRAME          = no.

      ASSIGN
         Btn_Help:HIDDEN IN FRAME DEFAULT-FRAME          = yes
         Btn_Help:HEIGHT IN FRAME DEFAULT-FRAME          = 1.14
         Btn_Help:ROW IN FRAME DEFAULT-FRAME             = 3.38
         Btn_Help:HIDDEN IN FRAME DEFAULT-FRAME          = no
         Btn_Help:HIDDEN IN FRAME DEFAULT-FRAME          = no.
```

*(2 of 2)*

```
      ASSIGN
         C-Win:VIRTUAL-HEIGHT                              = 16.00
         &IF '{&WINDOW-SYSTEM}' NE 'TTY':U &THEN
         C-Win:HIDDEN                                      = no &ENDIF.

    END.  /* Master Layout Layout Case */

    WHEN "Standard Character":U THEN DO:
      ASSIGN
         Btn_Done:HIDDEN IN FRAME DEFAULT-FRAME            = yes
         Btn_Done:HEIGHT IN FRAME DEFAULT-FRAME            = 1
         Btn_Done:ROW IN FRAME DEFAULT-FRAME               = 4
         Btn_Done:HIDDEN IN FRAME DEFAULT-FRAME            = no NO-ERROR.

      ASSIGN
         Btn_Help:HIDDEN IN FRAME DEFAULT-FRAME            = yes
         Btn_Help:HEIGHT IN FRAME DEFAULT-FRAME            = 1
         Btn_Help:ROW IN FRAME DEFAULT-FRAME               = 3 NO-ERROR.

    END.  /* Standard Character Layout Case */

  END CASE.
END PROCEDURE.  /* C-Win-layouts */
```

This sample contains CASE statement entries for two layouts, the master layout and the standard character layout. The Btn_Help button appears in the master layout but not in the standard character layout. The master layout also sets size properties to decimal character units because a graphical environment supports fractions of a character. However, in a character environment, they are set to integer character units.

### B.7.3 Enable_UI Internal Procedure

The Enable_UI internal procedure controls whether widgets are enabled for specific layouts. This procedure is used only for non-SmartObject procedure files. It uses the WHEN option of the ENABLE statement to determine whether a widget in a SmartObject is enabled.

Here is a sample Enable_UI procedure:

```
PROCEDURE enable_UI :
/*------------------------------------------------------------------------
  Purpose:     ENABLE the User Interface
  Parameters:  <none>
  Notes:       Here we display/view/enable the widgets in the
               user-interface.  In addition, OPEN all queries
               associated with each FRAME and BROWSE.
               These statements here are based on the "Other
               Settings" section of the widget Property Sheets.
------------------------------------------------------------------------*/

  {&OPEN-QUERY-DEFAULT-FRAME}
  GET FIRST DEFAULT-FRAME.
  IF AVAILABLE Customer THEN
    DISPLAY Customer.Cust-Num Customer.Name
      WITH FRAME DEFAULT-FRAME IN WINDOW C-Win.
  ENABLE IMAGE-1 Btn_First Btn_Next Btn_Prev Btn_Last BROWSE-1
         Btn_Photo WHEN NOT (SESSION:DISPLAY-TYPE = 'TTY':U )
      WITH FRAME DEFAULT-FRAME IN WINDOW C-Win.
  {&OPEN-BROWSERS-IN-QUERY-DEFAULT-FRAME}
  VIEW C-Win.
END PROCEDURE.
```

In this example, the Btn_Photo is enabled only if the standard character layout is not used at run time.

## B.8 Alternate Layout Limitations

Alternate layouts are somewhat less flexible than the master layout. This section describes the limitations of alternate layouts.

### B.8.1 Modifiable Properties

Within an alternate layout, you can change some, but not all of the properties that you can change in the master layout.Table B–2 lists the properties that you can change.

**Table B–2: Modifiable Properties in Alternate Layouts** *(1 of 2)*

| | | |
|---|---|---|
| BGCOLOR | GRAPHIC–EDGE | TITLE–FGCOLOR |
| COLUMN | HEIGHT | VIRTUAL–HEIGHT |
| EDGE–PIXELS | ROW | VIRTUAL–WIDTH |

**Table B–2: Modifiable Properties in Alternate Layouts** *(2 of 2)*

| | | |
|---|---|---|
| FGCOLOR | SEPARATORS | VISIBLE |
| FILLED | THREE–D | WIDTH |
| FONT | TITLE–BGCOLOR | |

**NOTE:** To exclude a widget from a particular alternate layout, set the Remove from Layout toggle box in the widget's property sheet.

### B.8.2 Text Widgets in Alternate Layouts

You can insert Text widgets only in the master layout. If the master layout contains a Text widget, you cannot cut or change the properties of the widget in an alternate layout. However, you can simulate a Text object in different layouts by using a Fill-in object with no label and activating the VIEW–AS–TEXT toggle box in the Fill–in's property sheet. The content of the field will appear to be a Static Text object.

### B.8.3 Combo Boxes in Alternate Layouts

In an alternate layout, you can only modify the height of a Simple Combo Box. You cannot modify the height of a Drop–down or Drop–down List.

### B.8.4 SIZE–TO–FIT

The SIZE–TO–FIT setting works only for the initial call to the master layout. In fact, when you change layouts, the width/height of the frame that you see is the design size, not the initial SIZE–TO–FIT size; that is, there is a margin on the right and bottom, even if you choose to set the layout to the master layout.

## B.9 Switching Between Layouts at Run Time

The multiple layout functionality that the AppBuilder provides is meant primarily to establish a layout once: when you run a procedure file. However, you can also switch among layouts at run time, if you write the necessary code to manage it.

**NOTE:** For SmartObjects, all of the code for switching among layouts at run time is managed by the ADM setInstanceProperties() function and applyLayout() procedure. For more information, see the "Applying SmartObject Layouts at Run Time" section.

One way to switch among layouts is to have a trigger execute the CASE statement entries for the different layouts. Within the trigger, follow these steps:

1 ♦ Execute the "Master Layout" CASE entry in the layout cases internal procedure with the following statement. The {&LAYOUT–VARIABLE} Preprocessor name is set:

```
RUN {&LAYOUT-VARIABLE}s (INPUT "Master Layout").
```

This statement changes all of the run-time properties to match those of the master layout. This step is necessary because all of the other layouts change their settings relative to the master layout. Thus, before you can switch to a new layout, you should switch back to the master layout.

2 ♦ Execute the CASE entry for the layout you want. This sets all of the layout's properties that you specify:

```
RUN {&LAYOUT-VARIABLE}s (INPUT "My Layout").
```

3 ♦ If necessary, write code that correctly enables the widgets in the new layout.

If widgets are enabled differently from one layout to another, you cannot execute the Enable_UI procedure to do your enabling. The Enable_UI procedure enables widgets based on WHEN phrases built using the layouts' associated run-time expressions. The explicit purpose of these run-time expressions is to identify a unique run-time environment. Since you want to switch layouts only within the same run-time environment, you must write your own code to handle the enabling.

## B.10 Simulating Character Applications

To help you develop applications for character interfaces, the AppBuilder simulates character environments in its design windows. Within a character simulator window, the widgets appear as they would in a character environment, to the extent possible. For example, buttons appear as they do in character environments, but some other widgets, such as radio sets, have a graphical appearance. To see exactly how the widgets appear at run time, you must run the application in the character environment.

**NOTE:** You can run character applications directly from the AppBuilder. For instructions, see the "Running Character Applications" section.

There are some operations that you can perform on widgets in a graphical window that you cannot perform on widgets in the character simulator window. Make sure to consider the following when designing a character interface:

- You can size buttons horizontally but not vertically. The height of a button in a character layout is one character.
- A browser must be at least six characters high.
- A (horizontal) radio set must be at least three characters high.

When you create a new design window or dialog box, the New dialog box appears. Select TTY Window or TTY Dialog to put the master layout in character mode. In general, if you plan to have both character and graphical layouts, make the master layout graphical.

## B.11 Running Character Applications

AppBuilder allows you to run character-mode procedures and view them in actual character-mode format. To do this, follow these steps:

1 ♦ Start the broker process for all necessary application databases. For example:

```
PROSERVE sports
```

2 ♦ Start the Progress session and connect to the application database in multi-user mode.

3 ♦ When you are ready to run your character procedure, choose Layout→ Alternate Layout from the AppBuilder Main Window.

4 ♦ On the Alternate Layout box, select Standard Character from the Layout combo box.

5 ♦ Choose Compile→ Run from the Main window.

AppBuilder disables itself and opens the AppBuilder Character Run Window, which starts a Progress Character client session. The Character client first connects to all the databases to which the AppBuilder is connected, then runs your character application.

6 ♦ To return to design mode in AppBuilder, exit the layout in the manner you designed for that layout; for example, by pressing the **ESC** key in the Character Run Window.

The AppBuilder Character Run Window clears the running layout and displays a status message of "Procedure complete. Waiting for next run." The AppBuilder enables itself so that you can make further changes to the character layout. Note that the Character client session is still running and remains connected to the application databases, so that you can run the procedure again to test your changes.

7 ♦ To close the Character Run Window, choose Compile→ Close Character Run Window in the AppBuilder Main window.

The Character client session exits and the Character Run Window closes.

**NOTE:** If the Character Run Window does not close, you might have to delete the file that the AppBuilder uses to communicate with the Character Run Window. This file has the form `pnnnnnm.tty` and is located in the Progress temporary file directory (by default, your current directory). When you delete this file, the character session ends and the Character Run window closes.

### B.11.1 Character Client Connection Issues

If the application databases with which you are testing a character layout require connection parameters in addition to the database names, specify the database names and additional connection parameters in the default Progress `startup.pf` file.

If you want to specify different connection parameters for the AppBuilder client and the Character Run client, create a parameter file for the Character client, then set the Progress WinChar PROSTARTUP registry setting to the new parameter file. For example, if the AppBuilder client session has different code page settings than the Character client, you can specify appropriate code page startup parameters in the character parameter file.

If you connect to additional databases during the AppBuilder session, you must specify only the database name and multi-user mode. The Character Run Window client will not recognize any additional connection options you specify, and running the character layout might fail to connect to the database.

### B.11.2 Double-byte Characters

The Character Run window does not support the display of double-byte characters. For more information about double-byte characters, see the *Progress Internationalization Guide*.

## B.12 Multiple Layouts for SmartObjects

Multiple layouts for SmartObjects are designed to provide alternate visualizations of a SmartObject, as might be required by use in different SmartContainers or by other functional criteria of the application, such as access privileges. Thus, the setup and initialization of a SmartObject layout is not primarily a function of the run-time environment, but of the SmartContainer logic that instantiates the SmartObject.

For SmartObjects, multiple layouts are supported by these ADM properties:

- **LayoutOptions** — Contains a comma-separated list of the layouts that you set up for a SmartObject in the AppBuilder at design time.
- **LayoutVariable** — Contains the layout that you choose (other than the default) for an instance of the SmartObject when you add it to a SmartContainer. (If you choose the default, the SmartObject uses the value of the DefaultLayout property.) This is also the variable that you set to change a SmartObject's layout at run time.
- **DefaultLayout** — Contains the layout that the SmartObject uses if you do not set an ObjectLayout property value for an instance of the SmartObject. DefaultLayout is set based on the run-time settings established in the Alternate Layout dialog box. If DefaultLayout has no value, the SmartObject uses the master layout.
- **ObjectLayout** — Contains the layout that the SmartObject uses.

Unlike the run-time expressions used for basic procedure files, the setting of these properties does not immediately affect a SmartObject's layout. Rather, the action of these properties depends on initialization of the SmartObject by its SmartContainer. At run time, the initializeObject() event procedure runs the layout cases internal procedure {&LAYOUT–VARIABLE}s by running applyLayout. The applyLayout() event procedure uses the value of the ObjectLayout or DefaultLayout property to determine initial visualization for the SmartObject. At any point after initialization, you can choose a different layout for the SmartObject by setting and applying a different value to the ObjectLayout property (from LayoutOptions).

### B.12.1 Creating SmartObject Alternate Layouts

You can create an alternate layout for a SmartObject or SmartContainer much like you do for a basic procedure file (see the "Creating Alternate Layouts"section). However, you should not enter a run-time expression to determine the active layout. Figure B–3 shows the Alternate Layout dialog box that appears for a SmartObject.

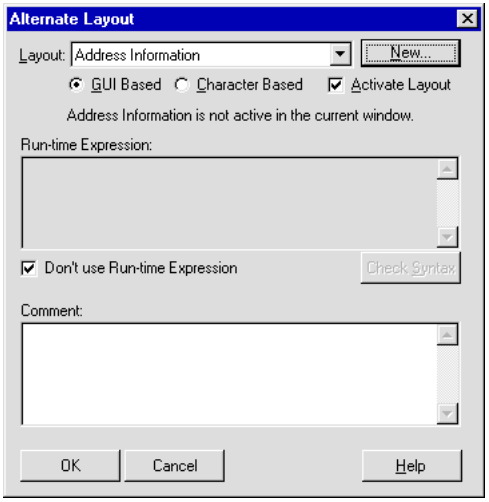



**Figure B–3: Alternate Layout Dialog Box for SmartObjects**

This example shows the setting for an alternate layout, Address Information, created for the SmartObject. Note the Don't use Run-time Expression option. This appears only for SmartObjects and is turned on by default. To activate a run-time expression for this layout, turn off the option. The default setting when you turn it off is FALSE.

## B.12.2 Modifying SmartObject Layouts

You can modify and synchronize SmartObject layouts in the design window similar to basic procedure file layouts. (For more information, see the"Modifying Layouts–Layout Inheritance" section.) However, SmartObject instances in a SmartContainer layout have additional features.

To illustrate working with SmartObject layouts, the rest of this section references a SmartDialog that contains a SmartDataViewer that has three layouts: a master and two alternates. The master layout, shown in Figure B–4, contains many of the fields of the sports database Customer table, some of which overlay each other.

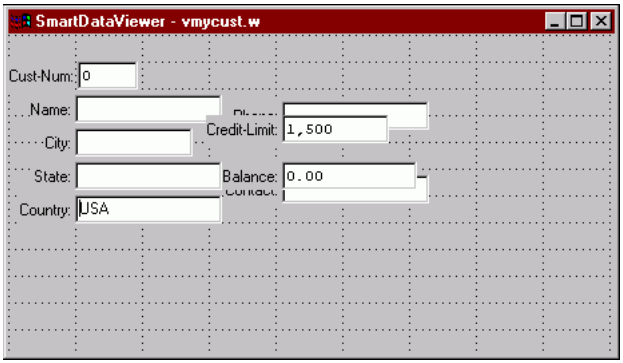



**Figure B–4: SmartDataViewer Master Layout**

Each of the alternate layouts, shown in Figure B–5 and Figure B–6, displays or hides different sets of the fields in the master layout.

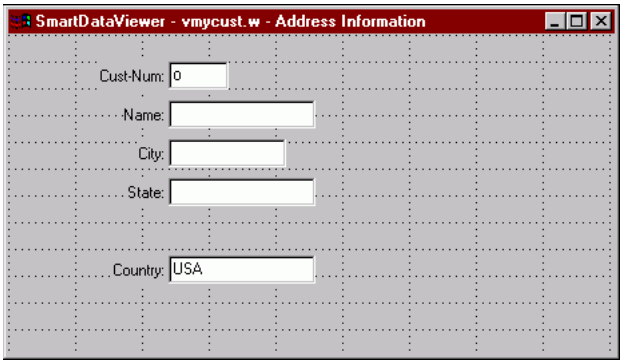



**Figure B–5: SmartDataViewer Address Alternate Layout**

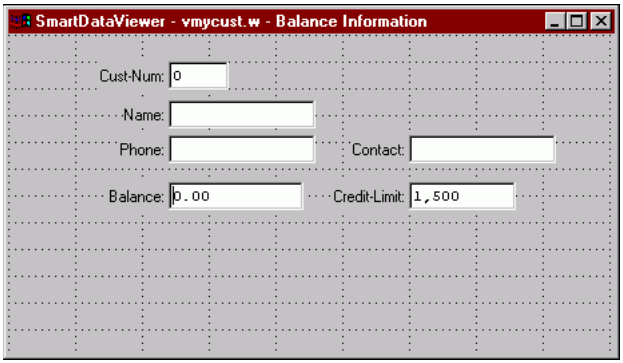



**Figure B–6: SmartDataViewer Balance Alternate Layout**

In general, SmartObjects instances inherit the active layout of the parent SmartContainer in the AppBuilder. While the SmartContainer is in an alternate layout, you cannot set the ObjectLayout (or any other instance property) of the SmartObjects that it contains. To set the active layout of a SmartObject instance, the parent SmartContainer must be in the master layout. Also, once set, the layout for a SmartObject instance overrides any subsequent layout setting inherited from the parent SmartContainer. If you want the SmartContainer to change the layout of its SmartObjects dynamically, you can create an override createObjects() event procedure that sets the ObjectLayout properties of the SmartObjects after they are created.

### Setting Layout Options for SmartObject Instances

Several SmartObjects provided with Progress include instance properties dialog boxes that allow you to set run-time options for a SmartObject instance. For more information, see the "SmartObject Properties" section in Appendix A, "Frequently Used Dialogs." You can set instance properties only for a SmartObject displayed in the master layout of the parent SmartContainer, in the example, a SmartDialog.

You can access the Properties dialog box for an instance in two ways. One is to click on the instance menu button to display the pop-up menu shown in Figure B–7.

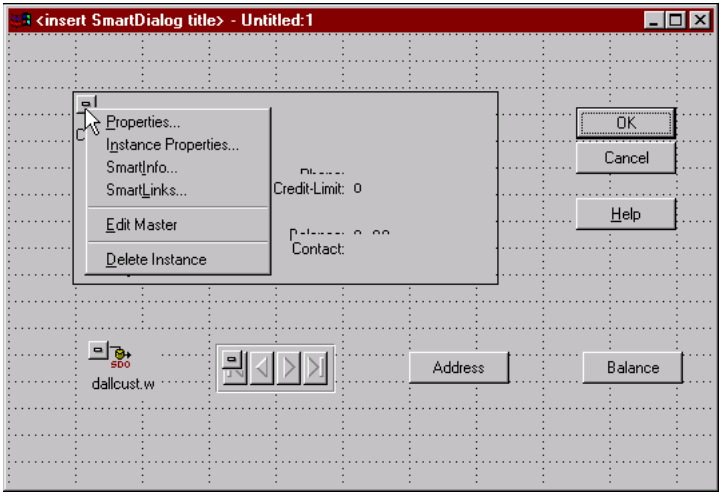



**Figure B–7: SmartDataViewer Menu Button's Pop-up Menu**

This allows you to go directly to the Instance Properties dialog box by clicking on Instance Properties. Alternatively, you can click on Properties (or double-click on the instance itself) to open the Instance property sheet, then click on the Edit button to open the Instance Properties dialog box.

For the same SmartObject instance with the parent SmartContainer displayed in an alternate layout, the menu button's pop-up menu changes to prevent you from modifying any instance properties, as shown in Figure B–8.

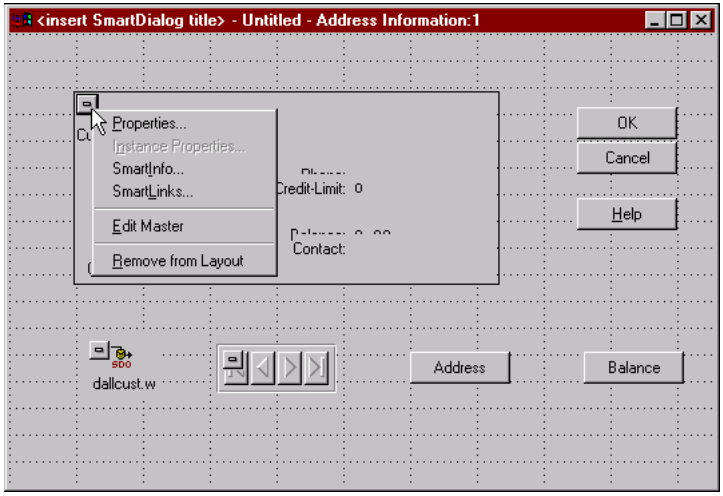



**Figure B–8: Menu Button's Pop-up Menu in Alternate Layout**

In this case, the Instance Properties option is insensitive and the Delete Instance option is replaced with the Remove from Layout option. Both Remove from Layout and Delete Instance hide the SmartObject from view. However, Remove from Layout only hides the SmartObject, while Delete Instance actually deletes it from the SmartContainer. Remove from Layout is used for alternate layouts rather than Delete Instance because the SmartObject instance is still available from the master layout of the SmartContainer.

Also, the Instance property sheet changes in a SmartContainer alternate layout, as shown in Figure B–9.

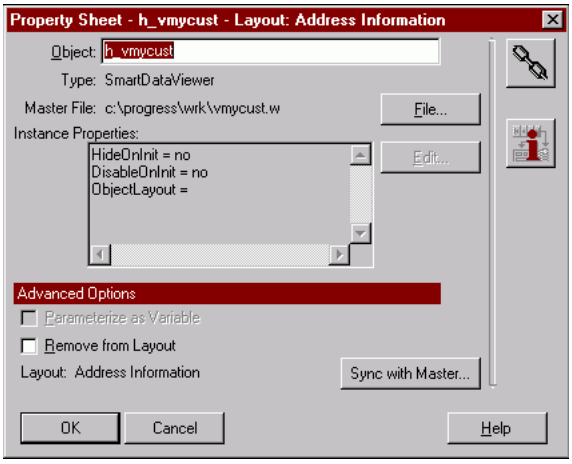



**Figure B–9: Instance Property Sheet in Alternate Layout**

This is the property sheet for the SmartDataViewer instance in an alternate layout. Note that the Edit button is insensitive, preventing editing of the instance properties. The Remove from Layout option, the Layout inherited from the SmartContainer, and the Sync with Master button are also added to the property sheet. The Parameterize as Variable option, normally available in the master layout, is also insensitive.

This example illustrates how you might have an alternate layout standard for the windows and dialog boxes of an entire application, while maintaining the ability to apply instance-specific layouts (Layout = Balance Information) for SmartObjects contained within these windows and dialog boxes. Thus, you might have two or more application visualizations in which the SmartObject instances retain the same functional layouts.

Two supported SmartObjects, SmartDataViewers and SmartDataBrowsers, support ObjectLayout as an instance property. You can set this property in the Instance Properties dialog box using the Layout combo box, as shown in Figure B–10.

**NOTE:** If the SmartObject has no alternate layouts, the Layout combo box is insensitive

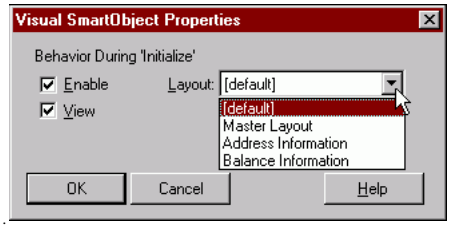



**Figure B–10: SmartDataViewer Properties with Layout Options**

You get this particular Instance Properties dialog box for the SmartDataViewer instance shown in Figure B–7, displayed in the SmartContainer master layout.

In general, the Layout combo box lists the LayoutOptions property values available for the SmartObject instance. These are the alternate layout names that you define for the SmartObject when you create or edit the SmartObject in the AppBuilder using the Alternate Layout dialog box (Figure B–3). If the SmartObject has a visualization for an alternate layout (such as Balance Information), you can make it visible for the instance by selecting the Layout value. Once you select and confirm the layout (by clicking on OK), the visualization for the selected layout appears in the SmartContainer design window for that SmartObject instance (Figure B–8). Note the difference in the SmartDataViewer instance layout between Figure B–7and Figure B–8.

Regardless of which layout you set for the SmartContainer, the SmartObject instance always displays the selected layout, as indicated by the ObjectLayout instance property setting shown in Figure B–9. This is true at run time as well as at design time, unless you explicitly create a run-time override of the instance layout. For more information on run-time layout overrides, see the "Applying SmartObject Layouts at Run Time" section.

If you leave the instance layout setting as [default], the instance in the design window has no layout property setting and displays the SmartObject master layout or the current layout of the SmartContainer, if different. This setting also allows you to specify the initial layout for the instance by the run-time setting of the DefaultLayout property.

## B.12.3 Applying SmartObject Layouts at Run Time

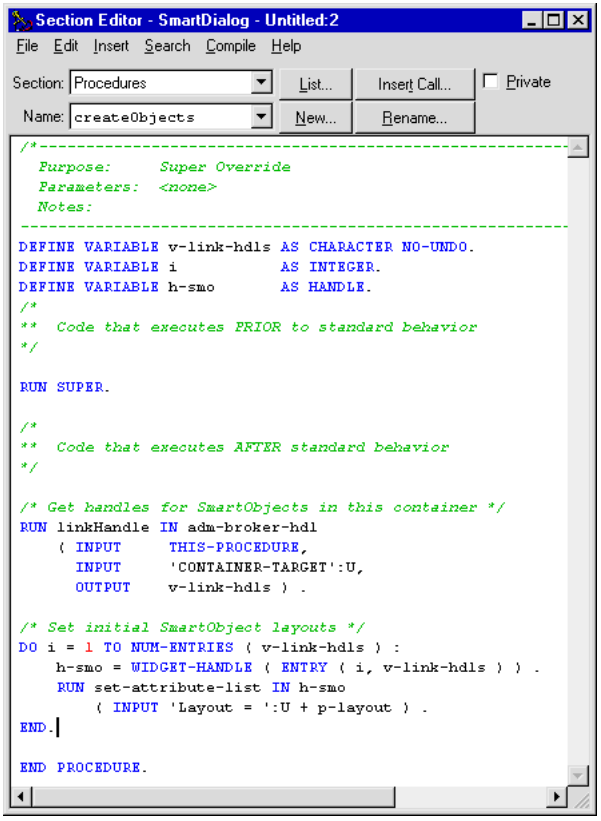
You can reliably affect the layout of a SmartObject at run time in two ways:

- Set the SmartObject ObjectLayout property before SmartObject initialization and display, usually in a local version of the createObjects() event procedure.
- Set the SmartObject ObjectLayout property and apply it any time after SmartObject initialization.

Thus, you can set the initial SmartObject layout and change it as often as you like at run time.

### Setting the Initial Layout

You can dynamically set the initial layout for your SmartObjects by creating a local createObjects() event procedure in the SmartContainer that parents the SmartObjects:




The code after the RUN SUPER statement is what you might add to the local createObjects() procedure. This code finds all of the SmartObjects in the SmartContainer (SmartObjects that have Container–Target links to this container). Next, it returns each link handle (the procedure handle for each SmartObject) and sets the ObjectLayout property for the associated SmartObject.

**NOTE:** Setting a property for a SmartObject that does not recognize the specified property has no effect. In this case, only SmartDataViewers and SmartDataBrowsers respond to the setting of ObjectLayout.

The p–layout identifier in this example represents a character-string input parameter to the SmartContainer that holds the initial layout value. This assumes that the SmartContainer is called from some other procedure, perhaps with the actual p–layout value determined at run time.

Note that the initial setting of ObjectLayout occurs after the ADM createObjects() procedure executes. Thus, the SmartContainer changes all the SmartObject instance settings of ObjectLayout after the SmartContainer creates its SmartObjects but before it initializes them (using initializeObject()).

### Changing Layouts

You can change the layout of a SmartObject with multiple layouts at any time after initialization. For example, you might do this in a user interface trigger that explicitly changes the layout when the user clicks on a button, or in response to an expression value that determines what layout a particular user is permitted to see (for example, a security key). You might have a series of expressions in a CASE statement that sets a varying layout according to a variety of criteria.

In all of these cases you execute the following code, where *so–handle* is your SmartObject procedure handle and *chosen–layout* is the name that you have assigned to the alternate layout to be applied:

```
DYNAMIC-FUNCTION('setObjectLayout':U IN so-handle, chosen-layout).
RUN applyLayout IN so-handle.
```

Thus, for the Balance button shown in Figure B–8:, you can create a trigger like this:

```
ON CHOOSE OF bBalance IN FRAME D-Dialog /* Balance */
DO:
    DYNAMIC-FUNCTION('setObjectLayout':U IN h_v_cust,
      "Balance Information":U).
    RUN applyLayout IN h_v_cust.
END.
```

Executing this code immediately redisplays the layout for the SmartObject specified by the h_v_cust handle. This code can execute anywhere inside or outside of the SmartObject as long as it occurs after SmartObject initialization and while the SmartObject is still instantiated.

# C

# Customizing AppBuilder

This appendix documents two techniques for customizing AppBuilder. It discusses the following topics:

- Custom object files and Extended Features (XFTRs)
- Creating custom object files
- Creating Extended Features (XFTRs)
- AppBuilder's API

Note that you can also use ADE hooks to change the behavior of AppBuilder; however, this appendix does not discuss this technique. For information, see the *Progress Basic Development Tools* manual (character only; information for Windows is in online help).

**NOTE:** Customizing AppBuilder, especially using XFTRs, is an advanced feature. You do not need to understand the material in this appendix to use AppBuilder effectively.

## C.1 Custom Object Files and Extended Features

Two techniques for customizing AppBuilder are:

- Creating custom object (.cst) files

  Custom object files determine which objects are available in AppBuilder. They determine the options on both the New dialog box (accessed by choosing File→ New from AppBuilder's main menu) and the Object Palette.

- Creating Extended Features (XFTRs)

  An *Extended Feature*, or *XFTR*, is a custom code block that you write and place in a procedure file. At critical moments in the processing of the procedure file, AppBuilder passes control to the XFTR, which can then run its own 4GL procedures. These procedures can interact with AppBuilder, calling the AppBuilder Application Program Interface (API). You use XFTRs to perform a variety of customized processing, such as creating and deleting objects, modifying code sections, and creating custom Preprocessor variables. Wizards and Cue Cards are examples of customized processing implemented with XFTRs.

The following sections describe these techniques.

## C.2 Creating Custom Object Files

A custom object file is a text file (normally with a .cst extension) that tells AppBuilder to work with your own custom widgets and SmartObjects. You can use a custom object file to control the appearance and behavior of the Object Palette and some aspects of the New dialog box.

Although you can create a custom object file from scratch, it is much simpler to copy and modify an existing custom object file. Progress supplies default custom object files that you can copy and modify according to your application needs.

Progress supplies different default custom object files depending on your AppBuilder licensing option. Table C–1 identifies all the custom files in the current release and the object definitions they contain. The files you have depend on your AppBuilder license.

**Table C–1: Custom Object Files and Their Contents**

| Filename | Contents |
|---|---|
| `progress.cst` | Basic Objects, SmartPanel, SmartFolder, SmartContainer, SmartWindow, SmartDialog, SmartFrame, Method Library, Structured Include, Structured Procedure |
| `shared.cst` | SmartDataObject |
| `smart.cst` | SmartDataBrowser and SmartDataViewer |
| `activex.cst` | CSSpin, CSComboBox, PSTimer |
| `web.cst` | CGI Wrapper, Frameset, HTML Mapping, Structured Include, Main, Structured Procedure, Report Template, Table, and Method Library |
| `smartv8.cst` | Version 8 SmartObject definitions |

**NOTE:** The ActiveX control definitions that were in the Progress Version 8 .cst file are now contained in the `activex.cst` file.

## C.2.1 General Coding Rules

Here are some general comments on coding custom object files:

- You can put comments (/* ... */) into a custom object file, but they must appear only in the following two locations:
  - At the top of a custom object file. The system ignores all lines up to the first line that begins with an asterisk (*) or a pound sign (#).
  - Inside TRIGGER blocks.

  If you place a comment anywhere else in a custom object file, you will lose data.
- Custom object files must end with a blank line, otherwise you will lose the last line of code.

### C.2.2 Entry Types

You can add four types of entries to a custom objects file:

- Add a button to the Object Palette
- Add a SmartObject to an Object Palette button pop-up menu
- Add a template to the New dialog box
- Add a custom widget to an Object Palette button pop-up menu

The following sections describe these entries.

### C.2.3 Adding a Button to the Objects Palette

Use this type of entry to define your own object type and place it on the Object Palette:

#### SYNTAX

```
#object-type &Default
UP-IMAGE-FILE bitmap-file [X,Y]
DOWN-IMAGE-FILE bitmap-file [X,Y]
LABEL label-name
{     USE master-file-name
   |  NEW template-file-name
        [ DIRECTORY-LIST list FILTER filter TITLE title ]
   |  DIRECTORY-LIST list FILTER filter TITLE title
}
[ DB-CONNECT ]
```

*#object-type* &Default

Specifies the type of object to add to the Object Palette.

UP-IMAGE-FILE *bitmap-file* [X,Y]

Specifies the name of the bitmap file for *object-type* when deselected. To fit properly on the Object Palette, the image must be 28 by 28 pixels, including all shadows and highlights.

`DOWN-IMAGE-FILE` *`bitmap-file`* `[X,Y]`

Specifies the name of the bitmap file for *object-type* when selected. To fit properly on the Object Palette, the image must be 28 by 28 pixels, including all shadows and highlights.

`LABEL` *`label-name`*

Specifies the text for the label for *object-type.* This label appears on the Object Palette and in the Object Palette menu.

`USE` *`master-file-name`*

Specifies the name of the SmartObject master file to instantiate when *smartobject-type* is selected from the Object Palette. If you specify USE, a Choose Object dialog box does not appear.

`NEW` *`template-file-name`*

Specifies the template file that AppBuilder opens when the user performs a New operation from Object Palette. You cannot use this option with the USE option.

`DIRECTORY-LIST` *`list`*

Specifies the directories that appear in the Choose dialog box for *object-type*.

`FILTER` *`filter`*

Specifies a comma-separated list of file search masks that appear in the Choose dialog box for the current directory.

`TITLE` *`title`*

Specifies the title for the Choose dialog box for *object-type*.

`DB-CONNECT`

Specifies whether AppBuilder prompts you to connect a database (if one is not connected) when you select the *object-type*. This is useful for data-aware objects such as SmartDataViewers, SmartDataObjects, and SmartDataBrowsers.

The sample entry in Figure C–1 adds a SmartObject type, specifically a SmartDataBrowser.

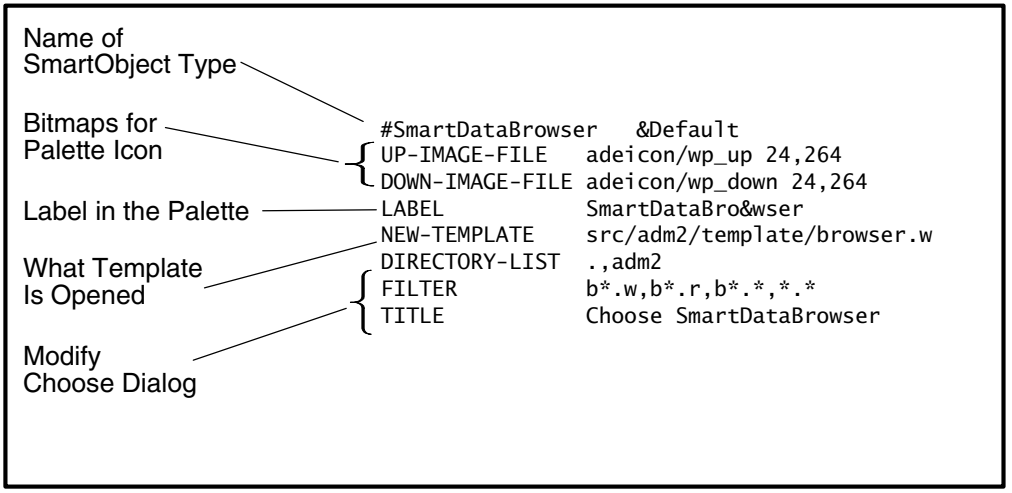



**Figure C–1: Sample Entry for SmartObject Type**

**NOTE:** You can also use the `adecomm/_chosobj.w` procedure in the `src/adecomm.pl` procedure library to modify the Choose Object dialog box. For more information, see the *Progress Basic Development Tools* manual (character only; information for Windows is in online help).

## C.2.4 Adding a SmartObject to a Palette Button Pop-up Menu

Use this type of entry to add an object to the Object Palette's pop-up menu for an object type. The new object you define also appears in the Object Palette's submenu for the object type. This type of entry also allows you to select an object without first using the Choose dialog box. Before you create this type of entry, make sure that you have created the Object type (see the "Adding a Button to the Objects Palette" section):

### SYNTAX

```
*smartobject-type    name
{   USE master-file
  | DIRECTORY-LIST list FILTER filter TITLE title
}
```

**smartobject-type*

Specifies the name of an existing object type.

*name*

Specifies the name of the object as it appears on the Object Palette's menus.

USE *master-file*

Specifies the name of the SmartObject master file to instantiate when *name* is selected from the Object Palette:

DIRECTORY-LIST *list*

Specifies the directories that appear in the Choose dialog box for *smartobject-type*.

FILTER *filter*

Specifies a comma-separated list of file search masks that appears in the Choose dialog box for the current directory.

TITLE *title*

Specifies the title for the Choose dialog box for *smartobject-type.*

The sample entries in Figure C–2 add SmartObject masters from `src/template/smart.cst`.

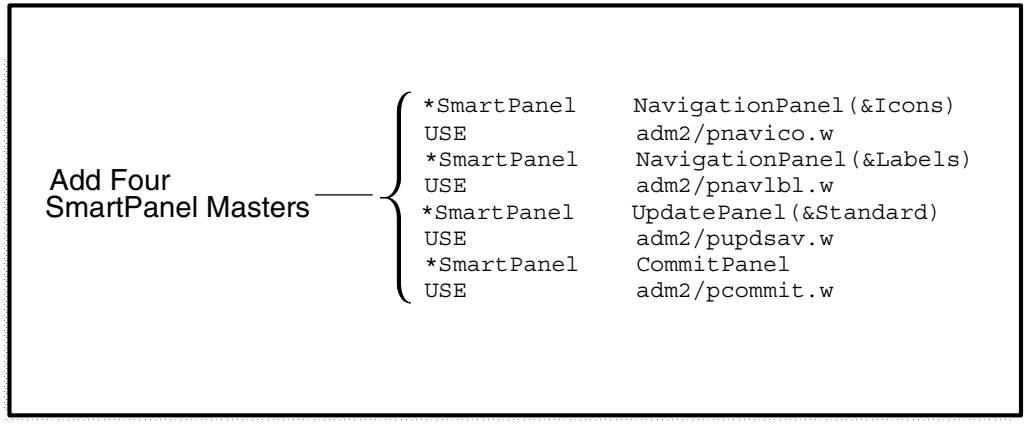



**Figure C–2: Sample Entries for Adding a SmartObject Master**

## C.2.5 Adding a New Template

This type of entry allows you to define your own template, whose name appears in the New dialog box:

### SYNTAX

```
{     *NEW-CONTAINER
   |  *NEW-SMARTOBJECT TYPE name
   |  *NEW-PROCEDURE
} template-name
NEW-TEMPLATE template-file
```

*NEW-CONTAINER

Specifies that the template appears in the New dialog box when the Containers toggle box is selected. A container is a either a window, a dialog box, or a SmartContainer.

*NEW-SMARTOBJECT

Specifies that the template appears in the New dialog box when the SmartObjects toggle box is selected.

TYPE *name*

Specifies the type of SmartObject. AppBuilder uses this value to determine whether to connect a database when the particular type of SmartObject is created.

*NEW-PROCEDURE

Specifies that the template appears in the New dialog box when the Procedures toggle box is selected. A procedure can be either a structured procedure file, a structured include file, or a method library.

*template-name*

Specifies the name of the template as it appears in the New dialog box.

NEW-TEMPLATE *template-file*

Specifies the pathname of the template file that AppBuilder opens when you select the template from the New dialog box.

The sample entries in Figure C–3 define New templates.

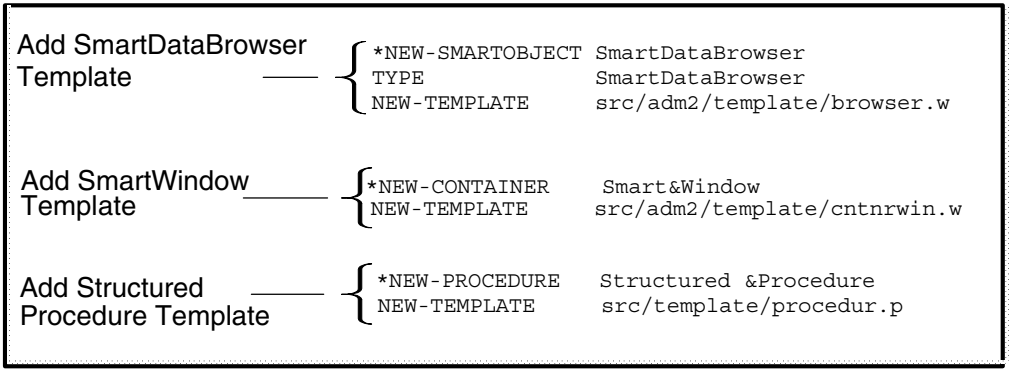



**Figure C–3: Sample Entries for New Templates**

## C.2.6 Adding a Widget to a Palette Button Pop-up Menu

You create and define custom widgets in a custom object file. Here is the syntax for adding custom widgets:

**SYNTAX**

```
*widget-type name
[ DESCRIPTION [ description-text-string ] ]
{ [ attribute value ] ...
  [ INHERIT name-of-entry ] ...
  [ trigger-block ] ...
} ...
RUN file-name
```

**widget-type*

Specifies the type of custom widget to create. It must be one of the following: BROWSE, BUTTON, COMBO-BOX, EDITOR, FILL-IN, FRAME, IMAGE, RADIO-SET, RECTANGLE, SELECTION-LIST, TEXT, and TOGGLE-BOX.

*name*

Specifies the display name of the custom widget as it appears on the Palette submenu (or pop-up menu on the Palette). After you choose a widget, AppBuilder displays *display-name*, followed by *widget-type*, in the status bar of AppBuilder main window.

`DESCRIPTION [ description-text-string ]`

Specifies the description of the widget. The description helps another developer to read and understand the intent of each custom widget.

*attribute*

Specifies an attribute to set for the widget. You can specify any attribute that you can also set from the widget's property sheet. See Table C–2, later in this appendix, for a list of the attributes that you can specify and the widgets to which they can apply.

*value*

Specifies the value for *attribute* (based on the data type of the attribute).

`INHERIT name-of-entry`

Specifies the display name of another widget in the custom widgets file. The current widget inherits all of the attribute settings specified for the other widget. If attribute settings conflict, the latter setting applies.

*trigger-block*

Specifies a trigger for the custom widget:

**SYNTAX**

```
TRIGGER event
  DO [ trigger-code ] END.
END TRIGGER
```

In this syntax, *event* specifies an event to pair with the custom widget and *trigger-code* specifies the code statements to execute for the widget/event pair.

`RUN file-name`

Specifies a procedure to run immediately before the object is created. The procedure file can take two parameters:

- **INPUT** *p_id* **AS INTEGER NO-UNDO** - The context ID of the object being created.
- **OUTPUT** *p_sct* **AS CHAR NO-UNDO** - A list of new custom object entries to process.

You can have multiple RUN statements for a single custom object file; they are run sequentially. Do not end a RUN statement line with a trailing period ( . ).

Table C–2 describes the attributes you can set in the custom objects file and the widgets to which they apply. In the Data Type column:

- C indicates CHARACTER
- P indicates PIXELS
- I indicates INTEGER
- L indicates LOGICAL

**Table C–2: Settable Attributes for Custom Widgets** *(1 of 7)*

| Attribute | Data Type | Allowed Widgets |
|---|---|---|
| AUTO-COMPLETION | L | COMBO BOX |
| AUTO-END-KEY | L | BUTTON |
| AUTO-GO | L | BUTTON |
| AUTO-INDENT | L | BUTTON, EDITOR, FILL-IN, SELECTION-LIST, TOGGLE-BOX |
| AUTO-RESIZE | L | BUTTON, EDITOR, FILL-IN, SELECTION-LIST, SLIDER, TOGGLE-BOX |
| AUTO-RETURN | L | FILL-IN |
| BGCOLOR | I | BROWSER, BUTTON, COMBO-BOX, EDITOR, FILL-IN, FRAME, IMAGE, RADIO-SET, RECTANGLE, SELECTION-LIST, SLIDER, TEXT, TOGGLE-BOX |
| BLANK | L | COMBO-BOX, FILL-IN |
| BOX-SELECTABLE | L | FRAME |
| CANCEL-BUTTON | L | BUTTON |

**Table C–2: Settable Attributes for Custom Widgets** *(2 of 7)*

| Attribute | Data Type | Allowed Widgets |
|---|---|---|
| COLUMN-SCROLLING | L | BROWSER |
| DATA-TYPE | C | COMBO-BOX, FILL-IN, RADIO-SET |
| DEBLANK | L | COMBO-BOX, FILL-IN |
| DEFAULT-BUTTON | L | BUTTON |
| DEFAULT-STYLE | L | BUTTON |
| DISPLAY | L | COMBO-BOX, EDITOR, FILL-IN, RADIO-SET, SELECTION-LIST, SLIDER, TOGGLE-BOX |
| DOWN | L | FRAME |
| DRAG-ENABLED | L | SELECTION-LIST |
| EDGE-PIXELS | I | RECTANGLE |
| ENABLE | L | BROWSER, BUTTON, COMBO-BOX, EDITOR, FILL-IN, IMAGE, RADIO-SET, RECTANGLE, SELECTION-LIST, SLIDER, TOGGLE-BOX |
| EXPAND | L | RADIO-SET |
| FGCOLOR | I | BROWSER, BUTTON, COMBO-BOX, EDITOR, FILL-IN, FRAME, IMAGE, RADIO-SET, RECTANGLE, SELECTION-LIST, SLIDER, TEXT, TOGGLE-BOX |
| FILLED | L | RECTANGLE |
| FONT | I | BROWSER, BUTTON, COMBO-BOX, EDITOR, FILL-IN, FRAME, RADIO-SET, SELECTION-LIST, SLIDER, TEXT, TOGGLE-BOX |
| FORMAT | C | COMBO-BOX, FILL-IN |

**Table C–2: Settable Attributes for Custom Widgets** *(3 of 7)*

| Attribute | Data Type | Allowed Widgets |
|---|---|---|
| GRAPHIC-EDGE | L | RECTANGLE |
| HEIGHT | I | BROWSER, BUTTON, COMBO-BOX, EDITOR, FILL-IN, FRAME, IMAGE, RADIO-SET, RECTANGLE, SELECTION-LIST, SLIDER, TEXT, TOGGLE-BOX |
| HEIGHT-PIXELS | I | BROWSER, BUTTON, COMBO-BOX, EDITOR, FILL-IN, FRAME, IMAGE, RADIO-SET, RECTANGLE, SELECTION-LIST, SLIDER, TEXT, TOGGLE-BOX |
| HELP | C | BUTTON, COMBO-BOX, EDITOR, FILL-IN, RADIO-SET, SELECTION-LIST, SLIDER, TOGGLE-BOX |
| HIDDEN | L | BROWSER, BUTTON, COMBO-BOX, EDITOR, FILL-IN, FRAME, IMAGE, RADIO-SET, RECTANGLE, SELECTION-LIST, SLIDER, TEXT, TOGGLE-BOX |
| HORIZONTAL | L | RADIO-SET, SLIDER |
| IMAGE-DOWN | C | BUTTON |
| IMAGE-FILE | C | BUTTON, IMAGE |
| IMAGE-INSENSITIVE | C | BUTTON |
| INITIAL-VALUE | C | COMBO-BOX, EDITOR, FILL-IN, RADIO-SET, SELECTION-LIST, SLIDER, TOGGLE-BOX |
| INNER-LINES | I | COMBO-BOX |
| KEEP-TAB-ORDER | L | FRAME |
| LABEL | C | BROWSER, BUTTON, COMBO-BOX, FILL-IN, FRAME, TOGGLE-BOX |

**Table C–2: Settable Attributes for Custom Widgets** *(4 of 7)*

| Attribute | Data Type | Allowed Widgets |
|---|---|---|
| LARGE | L | EDITOR |
| LAYOUT-UNIT | C or P | BROWSER, BUTTON, COMBO-BOX, EDITOR, FILL-IN, FRAME, IMAGE, RADIO-SET, RECTANGLE, SELECTION-LIST, SLIDER, TEXT, TOGGLE-BOX |
| LIST-ITEMS | C | COMBO-BOX, RADIO-SET, SELECTION-LIST |
| LOCK-COLUMNS | I | BROWSER |
| MANUAL-HIGHLIGHT | L | BUTTON, COMBO-BOX, EDITOR, FILL-IN, FRAME, IMAGE, RADIO-SET, RECTANGLE, SELECTION-LIST, SLIDER, TEXT, TOGGLE-BOX |
| MAX CHARS | I | EDITOR |
| MAX-DATA-GUESS | I | BROWSER |
| MAX-VALUE | I | SLIDER |
| MIN-VALUE | I | SLIDER |
| MOVABLE | L | BUTTON, COMBO-BOX, EDITOR, FILL-IN, FRAME, IMAGE, RADIO-SET, RECTANGLE, SELECTION-LIST, SLIDER, TEXT, TOGGLE-BOX |
| MULTIPLE | L | BROWSER, SELECTION-LIST |
| NAME | C | BROWSER, BUTTON, COMBO-BOX, EDITOR, FILL-IN, FRAME, IMAGE, RADIO-SET, RECTANGLE, SELECTION-LIST, SLIDER, TOGGLE-BOX |
| NATIVE | L | FILL-IN |
| NO-BOX | L | BROWSER, FRAME |

**Table C–2: Settable Attributes for Custom Widgets** *(5 of 7)*

| Attribute | Data Type | Allowed Widgets |
|---|---|---|
| NO-HIDE | L | FRAME |
| NO-LABEL | L | COMBO-BOX, FILL-IN |
| NO-LABELS | L | BROWSER, FRAME |
| NO-UNDERLINE | L | FRAME |
| NO-UNDO | L | COMBO-BOX, EDITOR, FILL-IN, RADIO-SET, SELECTION-LIST, SLIDER, TOGGLE-BOX |
| NO-VALIDATE | L | FRAME |
| OPEN-QUERY | L | BROWSER, FRAME |
| OVERLAY | L | FRAME |
| PAGE-BOTTOM | L | FRAME |
| PAGE-TOP | L | FRAME |
| PRIVATE-DATA | C | BROWSER, BUTTON, COMBO-BOX, EDITOR, FILL-IN, FRAME, IMAGE, RADIO-SET, RECTANGLE, SELECTION-LIST, SLIDER, TEXT, TOGGLE-BOX |
| RADIO-BUTTONS[1] | C | RADIO-SET |
| READ-ONLY | L | EDITOR |
| RESIZABLE | L | BUTTON, COMBO-BOX, EDITOR, FILL-IN, FRAME, IMAGE, RADIO-SET, RECTANGLE, SELECTION-LIST, SLIDER, TEXT, TOGGLE-BOX |
| RETAIN | I | FRAME |
| RETURN-INSERTED | L | EDITOR |
| SCROLLABLE | L | FRAME |

**Table C–2: Settable Attributes for Custom Widgets** *(6 of 7)*

| Attribute | Data Type | Allowed Widgets |
|---|---|---|
| SCROLLBAR-HORIZONTAL | L | EDITOR, SELECTION-LIST |
| SCROLLBAR-VERTICAL | L | EDITOR, SELECTION-LIST |
| SELECTABLE | L | BUTTON, COMBO-BOX, EDITOR, FILL-IN, IMAGE, RADIO-SET, RECTANGLE, SELECTION-LIST, SLIDER, TOGGLE-BOX |
| SENSITIVE | L | FRAME |
| SEPARATORS | L | BROWSER |
| SHARED | L | BROWSER, COMBO-BOX, EDITOR, FILL-IN, FRAME, RADIO-SET, SELECTION-LIST, SLIDER, TOGGLE-BOX |
| SIDE-LABELS | L | FRAME |
| SORT | L | COMBO-BOX, SELECTION-LIST |
| TITLE | L | BROWSER, FRAME |
| TITLE-BAR | L | BROWSER, FRAME |
| TITLE-BGCOLOR | I | FRAME |
| TITLE-FGCOLOR | I | FRAME |
| TOP-ONLY | L | FRAME |
| UNIQUE-MATCH | L | COMBO BOX |
| VIEW | L | BROWSER, BUTTON, COMBO-BOX, EDITOR, FILL-IN, FRAME, IMAGE, RADIO-SET, RECTANGLE, SELECTION-LIST, SLIDER, TEXT, TOGGLE-BOX |
| VIEW-AS-TEXT | L | FILL-IN |

**Table C–2: Settable Attributes for Custom Widgets** *(7 of 7)*

| Attribute | Data Type | Allowed Widgets |
|---|---|---|
| VIRTUAL-HEIGHT | I | FRAME |
| VIRTUAL-HEIGHT-PIXELS | I | FRAME |
| VIRTUAL-WIDTH | I | FRAME |
| VIRTUAL-WIDTH-PIXELS | I | FRAME |
| WIDTH | I | BROWSER, BUTTON, COMBO-BOX, EDITOR, FILL-IN, FRAME, IMAGE, RADIO-SET, RECTANGLE, SELECTION-LIST, SLIDER, TEXT, TOGGLE-BOX |
| WIDTH-PIXELS | I | BROWSER, BUTTON, COMBO-BOX, EDITOR, FILL-IN, FRAME, IMAGE, RADIO-SET, RECTANGLE, SELECTION-LIST, SLIDER, TEXT, TOGGLE-BOX |
| WORD-WRAP | L | EDITOR |

[1] The RADIO-BUTTONS attribute represents the labels and values for the individual buttons in a set, arranged in a comma-separated list.The following example identifies this format: label, value, label, value, label, value, and so forth. One label/value pair is defined for each button identified in a set.

### Sample Widget Entry

The following sample entry is in the src/template/progress.cst file:

```
*BUTTON         Ne&xt
DESCRIPTION     Simple Navigation Button
INHERIT         C&ustom Size/Color
LABEL           &Next
NAME            Btn_Next
TRIGGER CHOOSE
DO:
  &IF "{&PROCEDURE-TYPE}" EQ "SmartPanel" &THEN
    &IF "{&ADM-VERSION}" EQ "ADM1.1" &THEN
      RUN notify IN THIS-PROCEDURE ("get-next") NO-ERROR.
    &ELSE
      PUBLISH "fetchNext":U.
    &ENDIF
  &ELSEIF "{&TABLES-IN-QUERY-{&FRAME-NAME}}" NE "" &THEN
/*
**  This is a simple NEXT RECORD navigation button, useful for building
**  test screens quickly.  NOTE: if there are no tables in the query,
**  then this code will not compile' so use the preprocessor to skip it.
*/
    GET NEXT {&FRAME-NAME}.
    IF NOT AVAILABLE {&FIRST-TABLE-IN-QUERY-{&FRAME-NAME}}
      THEN GET LAST {&FRAME-NAME}.
    IF AVAILABLE {&FIRST-TABLE-IN-QUERY-{&FRAME-NAME}} THEN DO:
      DISPLAY {&FIELDS-IN-QUERY-{&FRAME-NAME}} WITH FRAME {&FRAME-NAME}.
      {&OPEN-BROWSERS-IN-QUERY-{&FRAME-NAME}}
    END.
  &ENDIF
END TRIGGER
```

If you analyze this sample entry one section at a time, you can see how to write your own custom widget entry. In this example:

- The first line specifies the widget type BUTTON and the display name for the widget:

  ```
  *BUTTON Ne&xt
  ```

  AppBuilder displays this name in the appropriate Object Palette menus. The ampersand (&) specifies that the "x" is the mnemonic on the Object Palette menu.

- The second line describes the button's functionality:

  ```
  DESCRIPTION Simple Navigation Button
  ```

AppBuilder does nothing with this description. The description is meant as a summary for someone reading the custom object file.

- The third line names another custom widget (Custom Size/Color) in the `progress.cst` file by using the widget's display name:

```
INHERIT C&ustom Size/Color
```

The current widget (Next) inherits all of the attribute settings for the Custom Size/Color button. Where you place the INHERIT keyword affects how attributes are set. Later definitions override earlier definitions.

- The fourth line specifies the label of the button:

```
LABEL &Next
```

The ampersand (&) specifies that "N" is the mnemonic for the button.

- The fifth line specifies the widget name that AppBuilder uses internally to refer to the button:

```
NAME Btn_Next
```

## C.3 Creating Extended Features (XFTRs)

An XFTR is a code block that you write to customize the behavior of AppBuilder. There are two uses for XFTRs:

- Adding custom run-time code
- Adding text blocks that will direct your own routines to interact with the user

An XFTR can be self-modifying; that is, change itself based on user action. Figure C–4 shows a sample XFTR.

```
&ANALYZE-SUSPEND _UIB-CODE-BLOCK _XFTR "SmartDataBrowserWizard" bTableWin
_INLINE
/* Actions: adm2/support/_wizard.w ? ? ? adm2/support/_wizdel.p */
/* SmartDataBrowser Wizard
Welcome to the SmartDataBrowser Wizard! During the next few steps, the wizard
will lead you through creating a SmartDataBrowser object. First you will
choose a DataSource to supply data to the SmartDataBrowser, then you can
specify the columns that will be displayed in the SmartDataBrowser. Press Next
to proceed.
adm2/support/_wizntro.w,adm2/support/_wizds.w,adm2/support/_wizdfld.w,adm2/s
upport/_wizend.w
*/
/* _UIB-CODE-BLOCK-END */
&ANALYZE-RESUME
```

**Figure C–4: XFTR in browser.w Template**

This code does not, by itself, perform any customized processing. However, it points to other 4GL procedures ( `adm2/support/_wizard.w` and `adm2/support/_wizdel.p`) that do perform customized processing. AppBuilder calls these procedures when significant AppBuilder events occur; for example, when a design window is realized or code is generated. Consequently, these procedures are called *`XFTR` event handlers*. An XFTR can reference up to five such procedures for AppBuilder events (see Table C–3).

As this example illustrates, an XFTR has five sections:

- An opening &ANALYZE-SUSPEND directive (required)
- A comment that maps event types to event handlers (required)
- A comment that contains code passed to the event handlers (optional)
- A comment that closes the code block (required)
- A closing &ANALYZE-RESUME directive (required)

The following sections explain how to set up these sections.

### C.3.1 The &ANALYZE-SUSPEND Directive for XFTRs

An XFTR code block begins with an &ANALYZE-SUSPEND directive. This directive is required. Here is its syntax for XFTR:

**SYNTAX**

```
&ANALYZE-SUSPEND UIB-CODE-BLOCK XFTR name object INLINE
```

The *name* value is a quoted string that specifies the name of the XFTR. The *object* value is an unquoted string with no embedded spaces that specifies a window name; if the procedure file has no window, specify this value as "Procedure."

### C.3.2 The Event-mapping Comment

This comment maps various AppBuilder events to XFTR event handlers. It is parsed by AppBuilder and must be the first comment after the &ANALYZE-SUSPEND directive. Figure C–5 provides an example, and illustrates the components of this comment type.

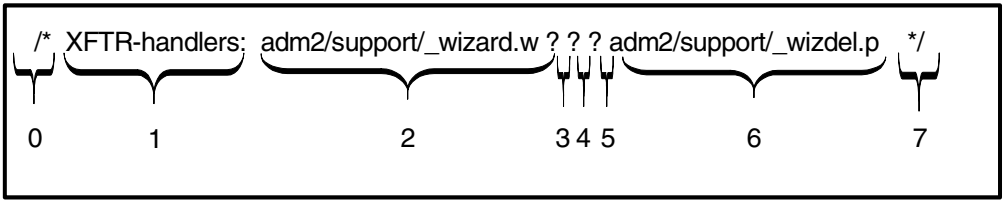



**Figure C–5: Event Handler Components**

Components 0 and 7 are standard comment delimiters. Component 1, which must end in a colon, is simply a title for the comment. Components 2 through 6 are space-delimited procedure names for the XFTR event handlers; each maps to a particular AppBuilder event, as follows:

- Position 2 maps to the Realize event.
- Position 3 maps to the Edit event.
- Position 4 maps to the Destroy event.
- Position 5 maps to the Read event.
- Position 6 maps to the Write event.

For example, in Figure C–5, if an AppBuilder Realize event occurs, AppBuilder calls the `adm2/support/_wizard.w` event. Note that a question mark indicates that no event handler is specified for the corresponding event.

Table C–3 summarizes AppBuilder events that you can map to XFTR event handlers.

**Table C–3: AppBuilder Events**

| AppBuilder Event | Description | User Actions |
|---|---|---|
| Realize | Occurs when a design window is realized. | Opening a procedure file, creating a new procedure file, changing layouts, or changing size of a window. |
| Edit | Occurs when AppBuilder presents an XFTR code block for editing. | Selecting an XFTR code section from the Section Editor's List Sections dialog box. |
| Destroy | Occurs when a procedure file is closed, or when AppBuilder closes a design window. | Closing a design window. |
| Read | Occurs when AppBuilder has finished opening a procedure file. | Opening a procedure file. |
| Write | Occurs when AppBuilder generates code and saves it to a procedure file. Specifically, occurs when the XFTR is encountered at code generation. | Saving or running a design window. |

### C.3.3 The Compilable Code Comment

This second comment, which is optional, has no specific format. The code section enclosed between the comment delimiters must be a block of Progress-compilable code (which can include comments). AppBuilder passes the contents of this code section to the XFTR event handlers that you reference in the first comment; the code executes when the procedure file is run.

### C.3.4 The Ending Comment

The third comment simply specifies the end of the code block. Include it exactly as shown:

```
/* _UIB-CODE-BLOCK-END */
```

### C.3.5 The &ANALYZE-RESUME Directive for XFTRs

An XFTR code block ends with an &ANALYZE-RESUME directive. Include it exactly as shown:

```
&ANALYZE-RESUME
```

### C.3.6 How AppBuilder Runs XFTR Event Handlers

When a AppBuilder event occurs for a specific procedure file and the procedure file contains an XFTR code block with an XFTR event handler specified in the corresponding position, AppBuilder runs the XFTR event handler and passes it two parameters:

- The context ID (integer) of the XFTR code block (INPUT parameter). A *context ID* is a unique integer ID for the code block. It can be used by other procedures in AppBuilder's API.
- The contents (character) of the optional code section (the section in between the first and last comments) in the XFTR code block (INPUT-OUTPUT parameter).

Make sure that your XFTR event handler can accept these parameters.

## C.4 AppBuilder's API

Once AppBuilder runs an XFTR event handler, the event handler can access AppBuilder's API. This allows the XFTR event handlers to interact with AppBuilder to provide customized processing that is coordinated with AppBuilder's standard processing.

### C.4.1 Context IDs

AppBuilder's API allows XFTR event handlers to gather information about all of the widgets, procedure files, SmartObject instances, and user-editable code sections in AppBuilder. Each of these different objects is stored in a separate record in AppBuilder's internal temporary tables. The context ID provides a unique reference for each type of object.

Although AppBuilder's API allows you to reference some objects by name or by handle, you should use the context ID to reference objects. Context IDs have the following advantages:

- They are guaranteed to be unique for each AppBuilder session.
- They generally provide faster access to the available objects.

### C.4.2 Naming Convention

All of the parameter names used in AppBuilder's API begin with the letter p (for parameter), which is followed by a second letter that specifies the data type, as indicated in Table C–4:

**Table C–4: Naming Conventions**

| Character | Data Type |
|---|---|
| c | Character |
| d | Decimal |
| h | Handle |
| i | Integer |
| l | Logical |

### C.4.3 AppBuilder's API Summary

Table C–5 summarizes the various procedures that comprise AppBuilder's API. These procedures are stored in the `adeuib.pl` procedure library. Each of these procedures is described in detail later in this appendix.

**Table C–5: API Procedures** *(1 of 2)*

| API Procedure | Purpose |
|---|---|
| `adeuib/_ablink.p` | Add and remove SmartLinks at design time in AppBuilder. |
| `adeuib/_accsect.p` | Create, delete, read, and write code sections in AppBuilder. |
| `adeuib/_dbtbnam.p` | Use AppBuilder to correctly generate table names based on the setting of the "Qualify Database Fields with Database Names" user preference. |
| `adeuib/_uibinfo.p` | Get information about the widgets, SmartObject instances, and the procedure itself, in a design window. |
| `adeuib/_uib_crt.p` | Create objects in AppBuilder. |
| `adeuib/_uib_del.p` | Delete objects in AppBuilder. |
| `adeuib/_uib_dlg.p` | Call a subset of the AppBuilder's dialog boxes. |

**Table C–5: API Procedures** *(2 of 2)*

| API Procedure | Purpose |
|---|---|
| adeuib/_winsave.p | Mark a procedure file as modified. |
| adeuib/_namespc.p | Register names of variables or procedures that cannot be used by AppBuilder users. |

## C.4.4 Add/Remove SmartLinks (adeuib/_ablink.p)

Use the adeuib/_ablink.p procedure to add/remove a SmartLink joining two SmartObjects:

### Calling Sequence

```
RUN adeuib/_ablink.p.p
  ( INPUT  pc_operation ,
    INPUT  pi_sourcecontext ,
    INPUT  pc_linkname ,
    INPUT  pi_targetcontext ,
  ) .
```

### Input Parameters

The adeuib/_accsect.p procedure takes the following input parameters:

*pc_operation*

A quoted character string that specifies the operation to be performed:

- **"ADD"**- Adds the link identified by the *pc_linkname* parameter.
- **"REMOVE"**- Removes the link identified by the *pc_linkname* parameter.

*pc_sourcecontext*

The context ID for the object at the originating (source) end of the link.

If the operation is "REMOVE", you can use the question mark (?) character to refer to THIS-PROCEDURE.

If the operation is "REMOVE" and the target context argument represents a single destination object, you can use:

- The context ID for a single source object
- A comma-separated list of context IDs, representing multiple source objects
- The star (*) character, representing all source objects

*pc_linkname*

A quoted character string identifying the type of link to be ADDed or REMOVEd. For example "FILTER"

*pc_targetcontext*

The context ID for the object at the destination (target) end of the link.

If the operation is "REMOVE", you can use the question mark (?) character to refer to THIS-PROCEDURE.

If the operation is "REMOVE" and the source context argument represents a single object, you can use:

- The context ID for a single destination object
- A comma-separated list of context IDs, representing multiple destination objects
- The star (*) character, representing all destination objects

**NOTE:** Either the source context or the target context must represent a single object. You cannot have multiple objects at both the source and target ends. If the operation is "ADD", both source and target must be single objects.

### Example

The following code purges all Navigation links originating at the caller:

```
/*
** Get own context ID.  Example presumes handle of
** current procedure previously stored in phSelf
*/

RUN adeuib/_uibinfo.p
    ( INPUT ?,
      INPUT "HANDLE " + STRING( phSelf ),
      INPUT "CONTEXT":U,
      OUTPUT cContextID
    ).

/*
** Remove all Nav links
*/

RUN adeuib/_ablink.p
    ( INPUT "REMOVE":U,
      INPUT cContextID,
      INPUT "NAVIGATION":U,
      INPUT "*":U
    ).
```

## C.4.5 Access Section (adeuib/_accsect.p)

Use the `adeuib/_accsect.p` procedure to read and write code sections in AppBuilder:

### Calling Sequence

```
RUN adeuib/_accsect.p
  ( INPUT         pc_mode ,
    INPUT         pi_context ,
    INPUT         pc_section ,
    INPUT-OUTPUT  pi_srecid ,
    INPUT-OUTPUT  pc_code
  ) .
```

### Input Parameters

The `adeuib/_accsect.p` procedure takes the following input parameters:

*pc_mode*

A quoted character string that specifies the mode of operation. Valid values are:

- **"GET"**— Reads the contents of the section or returns unknown (?).
- **"SET"**— Writes the contents of the section.
- **"DELETE"**— Deletes the current section.

  You cannot actually delete either the MAIN-CODE-BLOCK or the DEFINITIONS sections (see pc_section below). The DELETE value empties the section but does not also remove the stub.

*pi_context*

The context ID of the object to access. If unknown (?), AppBuilder assumes the current window or procedure.

*pc_section*

The name of the code section to access. Valid values are:

- DEFINITIONS
- MAIN-CODE-BLOCK
- TRIGGER:*event-name*
  (For example, TRIGGER:CHOOSE)
- PROCEDURE:*name*:*handler*
  (For example, PROCEDURE:mytest:maintain.p. See the "Examples" section below for a description of how you can use this.)
- FUNCTION:*name*
- XFTR:*xftr-name*

  When you have a TRIGGER, then *pi_context* refers to a widget. In all other cases, *pi_context* points to the procedure.

  If *pi_srecid* is unknown (?), then *pi_context* and *pc_section* are used to identify an existing section. If *pi_srecid* is known, then *pc_section* is ignored.

### Input-Output Parameters

The adeuib/_accsect.p procedure uses the following INPUT-OUTPUT parameters:

*pi_srecid*

The context ID of the current section. If its value is other than unknown, pc_section is ignored.

*pc_code*

The contents of the trigger or other code section.

### Return Values

This procedure returns:

"Error"

Indicates that pi_context is not specified and there is no current procedure or window.

### Examples

The following code resets the MAIN-CODE-BLOCK to an include file:

```
/* Get the current value */
 pi_srecid=?.
 RUN adeuib/_accsect.p ("GET", ?, "MAIN-CODE-BLOCK",
                        INPUT-OUTPUT srecid
                        INPUT-OUTPUT code).
 IF RETURN-VALUE NE "Error" THEN DO:
   IF code ne "{include.i}" THEN DO:
      code = "{include.i}".
      RUN adeuib/_accsect.p ("SET", ?, ?,
                             INPUT-OUTPUT srecid,
                             INPUT-OUTPUT code).
   END.
 END.
```

The following example creates a AppBuilder-maintained procedure: a procedure that the Code Section Editor displays as a READ-ONLY procedure. The following code fragment calls `adeuib/_accsect.p`:

```
DEF VAR section_id as INTEGER INITIAL ?.
DEF VAR code AS CHAR .

RUN adeuib/_accsect.p(INPUT "SET":U,
                       INPUT ?,
                      INPUT "PROCEDURE:mytest:maintain.p",
                       INPUT-OUTPUT section_id,
                       INPUT-OUTPUT code).
```

In this example:

- The first INPUT parameter, "SET":U, specifies SET mode; SET mode tells `adeuib/_accsect.p` to write or create a code section.
- The second INPUT parameter, ?, specifies the context ID of the object to access. By default, `adeuib/_accsect.p` interprets the unknown value (?) to be the context ID of the current window or procedure.
- The third INPUT parameter, "PROCEDURE:mytest:maintain.p", specifies the following:
  - The type of section to create is an internal procedure.
  - The name of the internal procedure is mytest.
  - The procedure named `maintain.p` will handle generating the code for the internal procedure.
- The first INPUT-OUTPUT parameter specifies the context ID of the section; this value must be unknown (?) or pc_section is ignored.
- The final INPUT-OUTPUT parameter is a character string that holds the contents of the section.

This maintain.p procedure generates the code for the mytest internal procedure:

```
/*------------------------------------------------------------------------
File: maintain.p

Description:
    Sample procedure showing how a developer procedure can maintain
    a AppBuilder Code Section.
Input Parameters:
    p_id  - (INTEGER) context ID of the current procedure

Output Parameters:
    p_Code - (CHAR) code to return. (Including "END PROCEDURE." ).

------------------------------------------------------------------------*/
DEFINE INPUT  PARAMETER     p_id      AS INTEGER    NO-UNDO.
DEFINE OUTPUT PARAMETER     p_code    AS CHAR       NO-UNDO.

DEFINE VAR ch AS CHAR NO-UNDO.

/* Line for comments. */
&Scoped-define COMMENT-LINE
------------------------------------------------------------------------
/* Standard End-of-line character */
&Scoped-define EOL CHR(10)

/* Get some info about the current procedure. */
RUN adeuib/_uibinfo.p (p_id, ?, "FILE-NAME", OUTPUT ch).

p_code =
"/*{&COMMENT-LINE}" + {&EOL} +
"  Purpose:     Sample code for file " + ch + {&EOL} +
"  Parameters:  <none>" + {&EOL} +
"{&COMMENT-LINE}*/" + {&EOL} + {&EOL} +
"   MESSAGE 'Hello World!':U." + {&EOL} + {&EOL} +
"END PROCEDURE.".
```

The maintain.p procedure generates a mytest procedure that the Code Section Editor displays as read only. Here are the contents of the generated mytest procedure:

```
/*------------------------------------------------------------------------
  Purpose:     Sample code for file C:\WORK\V-CUST.W
  Parameters:  <none>
------------------------------------------------------------------------*/

   MESSAGE 'Hello World!':U.

END PROCEDURE.
```

AppBuilder-maintained Procedures provide an alternative to traditional XFTRs. They basically fill the role of an XFTR with only a single WRITE handler. The advantages of AppBuilder-maintained procedures over XFTRs are:

- You can add and delete them programmatically using `adeuib/_accsect.p`.
- They are visible to you, the developer, so that you can see the code being generated.

Their disadvantages are:

- They can be deleted by the user from the Code Section Editor menu (Edit/Delete Procedure).
- They cannot store data in comments used to rebuild the code section. That is, the contents of the AppBuilder-maintained procedure must be derived totally from external sources or calls to `adeuib/_uibinfo.p`. For example, an XFTR handler can read a section and have it affect what it generates; however, an AppBuilder-maintained procedure can only write code.

## C.4.6 AppBuilder Database Table Names (adeuib/_dbtbnam.p)

Use the `adeuib/_dbtbnam.p` procedure to have AppBuilder correctly generate table names based on setting the "Qualify Database Fields with Database Names" user preference:

### Calling Sequence

```
RUN adeuib/_dbtbnam.p
  ( INPUT p_id ,
    INPUT p_name_in
    OUTPUT p_name_out
  ) .
```

### Input Parameters

The `adeuib/_dbtbnam.p` procedure uses the following input parameters:

*pi_context*

The context ID of the current procedure (to obtain this ID, use `adeuib/_uibinfo.p`).

*pc_name_in*

A table name. This name might or might not be qualified with a database name.

### Output Parameter

The adeuib/_dbtbnam.p procedure uses the following output parameter:

*pc_name_out*

A table name that meets the settings of the "Qualify Database Fields with Database Names" user preference.

### Example

```
DEFINE VAR  db-tbl     AS CHAR       NO-UNDO.
DEFINE VAR  proc-id    AS INTEGER    NO-UNDO.
DEFINE VAR  ch         AS CHAR       NO-UNDO.

/*
**  Get the context of the current procedure.
*/
RUN adeuib/_uibinfo.p (INPUT ?, INPUT ?, INPUT "PROCEDURE":U, OUTPUT ch).
proc-id = INTEGER (ch).
/*
**  The user may have wanted to suppress the db name in AppBuilder.
**  Ask AppBuilder to set up the database/table name.
*/
RUN adeuib/_dbtbnam.p (proc-id, "Sports.Customer":U, OUTPUT db-tbl).
MESSAGE db-tbl.
```

## C.4.7 AppBuilder Information (adeuib/_uibinfo.p)

Use the adeuib/_uibinfo.p procedure to obtain information about the widgets and SmartObject instances in a design window:

### Calling Sequence

```
RUN adeuib/_uibinfo.p
  ( INPUT  pi_context ,
    INPUT  pc_name ,
    INPUT  pc_request ,
    OUTPUT pc_info
   ) .
```

### Input Parameters

The adeuib/_uibinfo.p procedure takes the following input parameters:

*pi_context*

The context ID of the widget, SmartObject instance, code section, or procedure to access. If unknown (?), AppBuilder assumes the current procedure. The value of *pi_context* can be the context ID for any of the following:

- An object. For example, you can obtain the context of a button, Btn_OK, with the following code:

```
RUN adeuib/_uibinfo.p (?,"Btn_OK","CONTEXT", OUTPUT c_info).
```

- A procedure. For example, you can obtain the context for the current procedure with the following code:

```
RUN  adeuib/_uibinfo.p (?, ?, "PROCEDURE", OUTPUT c_info).
```

- A SmartObject instance. For example, you can obtain the context of a SmartObject instance with the following code:

```
RUN  adeuib/_uibinfo.p (?, "HANDLE " + STRING(THIS-PROCEDURE).
```

- A code section. For example, you can obtain the context ID and contents of the MAIN-CODE-BLOCK section with the following code:

```
RUN adeuib/_accsect.p
  ("GET", ?, "MAIN-CODE-BLOCK",
    INPUT-OUTPUT i_context,
    INPUT-OUTPUT c_code).
```

*pc_name*

A quoted character string that specifies the name of an object. This string has the following syntax:

**SYNTAX**

```
{    object
       [ IN FRAME frame-name ]
       [ IN WINDOW window-name ]
   |   object
       [ IN FRAME frame-name ]
       [ IN PROCEDURE file-name ]
   |   HANDLE handle
}
```

In this syntax:

- *object* is the name of the widget or SmartObject instance.
- *frame-name* is the name of its parent frame.
- *window-name* is the name of its parent window.
- *file-name* is the name of the parent procedure file.
- *handle* is either the widget-handle of a widget or the procedure handle for a SmartObject instance.

If you omit the FRAME or WINDOW option, the current window and current frame are assumed. If an object is unique in a window, you can refer to it as object IN WINDOW *window-name*.

To refer to a window or frame, preface the object with the type. For example, to find frame f:

```
FRAME f IN WINDOW w
```

To find window w:

```
WINDOW w
```

To find a procedure with the name `p.w`:

```
PROCEDURE p.w
```

The following special cases also apply:

- **?** — Gets the current object (shown in the AppBuilder's main window)
- **FRAME ?** — Gets the current frame.
- **WINDOW ?** — Gets the current window.

- **PROCEDURE ?** — Gets the current procedure.

  If you use the HANDLE *handle* to reference an object, the handle can be either a WIDGET-HANDLE of the object or PROCEDURE-HANDLE for a SmartObject (converted to a STRING).

- **PALETTE** *object* — *object* is the name directly following # in the cst file.

*pc_request*

A quoted character string that specifies what to request. Valid requests are:

- **"NAME"** — Returns the name of the object.
- **"ATTRIBUTES"** — Returns a string with all of the attributes read from the cst file.
- **"PROCEDURE"** — Returns the STRING(pi_context) for the current procedure.
- **"FILE-NAME"** — Returns the name of the parent procedure file as last saved by AppBuilder; if not saved, returns "?".
- **"TEMPLATE"** — Returns YES or NO depending on whether the current object is a template.
- **"TYPE"** — Returns the type of object.
- **"HANDLE"**— Returns either the widget-handle of the object or the adm-object-handle of a SmartObject.
- **"PROCEDURE-HANDLE"**— Returns the procedure handle of a SmartObject.
- **"CONTEXT"**— Returns the context ID of the widget or SmartObject instance.
- **"CONTAINS** *contains-phrase*"— Returns all objects contained in the current context (of a type or with key toggle boxes set). By default, returns the context ID of the items that match the filter, but you can ask for the list of names. The asterisk (*) returns all objects.

The syntax for *contains-phrase* is as follows:

**SYNTAX**

```
{ * | object-list }
[ {      DISPLAY | ENABLE | MANUAL-HIGHLIGHT | MOVABLE
         | RESIZABLE | SELECTABLE | SHARED | VISIBLE
         | LIST-1 | LIST-2 | LIST-3 | LIST-4 | LIST-5 | LIST-6
    }
   { TRUE/YES | FALSE/NO }
]
[ RETURN { CONTEXT | NAME } ]
```

In this syntax, *object-list* is a comma-separated list (with no spaces) of object types.

- **"FRAMES"** — Returns all frames (in a window). This is shorthand for CONTAINS FRAME RETURN NAME.
- **"FIELDS"** — Returns dbfields in frame or dbfields in browse.
- **"EXTERNAL-TABLES"** — Returns the list of external tables for a procedure.
- **"TABLES"** — The tables used by the query of a FRAME, BROWSE, or QUERY object.
- **"4GL-QUERY"** — The 4GL query for the query of a FRAME, BROWSE, or QUERY object.
- **"WBX-FILE-NAME"** — The name of the .wrx in which to save run-time attributes for the OCXs in a procedure.
- **"COMPILE-INTO-DIR"** — The directory where the procedure file is compiled.
- **"&FRAME-NAME [RETURN NAME]"** — The context ID or name of the FRAME-NAME, or ?.
- **"&QUERY-NAME [RETURN NAME]"** — The context ID or name of QUERY-NAME, or ?.
- **"&BROWSE-NAME [RETURN NAME]"** — The context ID or name of BROWSE-NAME, or ?.

### Other Options

To return a comma-separated list of procedure names:

```
RUN adeuib/_uibinfo.p
    (INPUT ?,                              /* P_context (INTEGER)    */
     INPUT "SESSION",                      /* p_name    (CHARACTER)  */
     INPUT "PROCEDURES RETURN NAME",       /* P_request (CHARACTER)  */
     OUTPUT name-list).                      /* p_info     (CHARACTER)  */
```

The following returns the same list but uses context IDs instead of names:

```
RUN adeuib/_uibinfo.p
    (INPUT ?,                              /* P_context (INTEGER)    */
     INPUT "SESSION",                      /* p_name    (CHARACTER)  */
     INPUT "PROCEDURES RETURN CONTEXT",    /* P_request (CHARACTER)  */
     OUTPUT name-list).                      /* p_info      (CHARACTER)  */
```

## SECTIONS

You can ask a procedure for the list of internal procedures and functions defined in AppBuilder. The list can return the names or context IDs of the procedure and function sections, which you can use in `adeuib/_accsect.p` to get, modify, or delete the contents of code blocks:

## SYNTAX

```
SECTIONS PROCEDURE [ ,FUNCTION ]
  [ RETURN { CONTEXT | NAME } ]
```

If you do not specify either RETURN CONTEXT or RETURN NAME, it returns the list in context ID order.

To obtain a list of all internal procedures in a file by name, use the following:

```
SECTIONS PROCEDURE RETURN NAME
```

To obtain all of the procedures and functions defined in a file, use the following:

```
SECTIONS PROCEDURE,FUNCTION
```

This returns the list in context ID order.

### HTML-FILE-NAME

Returns the HTML filename associated with an HTML mapping object procedure.

### BROKERURL

Returns the broker URL set in Web Preferences:

```
RUN adeuib/_uibinfo.p(?,"SESSION":U,"BrokerURL":U,OUTPUT BrokerURL).
```

### REMOTE

Returns TRUE if the Remote File Management button in AppBuilder is set to 'Remote', otherwise returns FALSE:

```
RUN adeuib/_uibinfo.p(?,"SESSION":U,"REMOTE":U, OUTPUT RemoteFlag).
```

### WEBBROWSER

Returns the Web Browser set in Web Preferences.

### DATAOBJECT

Returns the SmartDataObject associated with the procedure:

```
RUN adeuib/_uibinfo.p(INT(hProc),"","DataObject":U, OUTPUT SDOName).
```

### DATAOBJECT-INCLUDE

Returns the name of the include file associated with the SmartDataObject of the procedure:

```
RUN adeuib/_uibinfo.p(INT(hProc),"","DataObject-Include":U,
  OUTPUT SDOiName).
```

### PALETTE-ITEM

For a given custom definition in a .cst file, returns the settings for ATTRIBUTES, LABEL, TEMPLATE, and FILES:

### SYNTAX

```
PALETTE-ITEM cst-obj-name
  [ ATTRIBUTES | LABEL | TEMPLATE | FILES ]
```

```
RUN adeuib/_uibinfo.p ( INPUT ?, INPUT "PALETTE-ITEM SmartDataObject":U,
  INPUT "ATTRIBUTES":U, OUTPUT Attributes).
```

#### Output Parameter

The adeuib/_uibinfo.p procedure uses the following output parameter:

*pc_info*

Output value, cast as a character string.

### C.4.8 AppBuilder Create (adeuib/_uib_crt.p)

Use the adeuib/_uib_crt.p procedure to create objects in AppBuilder. You can create under program control the objects found in the custom object files. The TYPES, ARGUMENTS, and CUSTOM TYPES are the same as those defined in the custom object files. This acts just as if you choose a button from the Object Palette and insert it into a window or frame:

#### Calling Sequence

```
RUN adeuib/_uib_crt.p
  ( INPUT pi_parent ,
    INPUT pc_type ,
    INPUT pc_custom ,
    INPUT pd_Row ,
    INPUT pd_Column ,
    INPUT pd_height ,
    INPUT pd_width ,
    OUTPUT pi_context
   ) .
```

#### Input Parameters

The adeuib/_uib_crt.p procedure takes the following input parameters:

*pi_parent*

The context ID of the parent of the object to create. If this is unknown (?), the parent is assumed to be the current frame. If there is no current frame, then the current window is assumed.

*pc_type*

The type of object to create (for example, "BUTTON" or "SmartObject").

*pc_custom*

The name of the custom object type (for example, “OK Button”). If this is unknown (?), the “&Default” object is created. This is of the form:

```
"case:value"
```

### Special Cases of pc_custom

Here are some special cases of the pc_custom parameter:

"Custom:*name*"

The name of the custom object defined in Custom files. For example, the following draws a Next button:

```
Custom:&Next
```

"SmartObject*:object-file*"

The name of a SmartObject to load. For example:

```
SmartObject: C:/Progra~1/Progress/gui/object/p-nav.w
```

"SPECIAL*:attributes-values*"

This is similar to creating a new custom object file entry temporarily. The attributes and values are parsed the same way as entries in the custom object file. Blank lines are ignored. You must separate lines with a carriage return (that is, CHR(10)). For example:

```
"SPECIAL: " +
   "BGCOLOR    7 " + CHR(10) +
   "FONT       2 " + CHR(10) +
   "NAME       test".
```

*pd_ROW*

The ROW to create the object.

*pd_COLUMN*

The COLUMN to create the object.

*pd_HEIGHT*

The HEIGHT of the object in characters; if unknown (?), the default height is used.

*pd_WIDTH*

The WIDTH of the object in characters; if unknown (?), the default width is used.

### Output Parameter

The `adeuib/_uib_crt.p` procedure uses the following output parameter:

*pi_context*

The context ID of the object created. If the creation fails, this is unknown (?). You can use this value as the object context ID for the companion programs, which are `adeuib/_accsect.p`, `adeuib/_uibinfo.p`, and `adeuib/_uib_del.p`.

### Return Values

The `adeuib/_uib_crt.p` procedure returns:

`"Error"`

If *pi_context* does not point to a valid object.

## C.4.9 AppBuilder Delete (adeuib/_uib_del.p)

Use the `adeuib/_uib_del.p` procedure to delete objects in AppBuilder:

### Calling Sequence

```
RUN adeuib/_uib_del.p
  ( INPUT pi_context ) .
```

### Input Parameter

The `adeuib/_uib_del.p` procedure takes the following input parameter:

*pi_context*

The context ID of the object to delete. There is no default value for this; you must pass a valid context ID. This value is the same value returned by `adeuib/_uibinfo.p`.

### Return Values

This procedure returns:

"Error"

Indicates that *pi_context* does not point to a valid object.

"Fail"

Indicates that the object was not deleted.

## C.4.10 AppBuilder Dialog (adeuib/_uib_dlg.p)

Use the adeuib/_uib_dlg.p procedure to invoke a subset of AppBuilder's dialog boxes:

### Calling Sequence

```
RUN adeuib/_uib_dlg.p
  ( INPUT pi_context ,
    INPUT pc_dname ,
    INPUT-OUTPUT pc_args
   ) .
```

### Input Parameters

The adeuib/_uib_dlg.p procedure takes the following input parameters:

*pi_context*

The context ID of the object to which the dialog box will apply.

*pc_dname*

A quoted character string that specifies the name of the AppBuilder dialog box to call. Valid values are:

- **"QUERY BUILDER"** — The context ID must be for a query, frame, or browse.
- **"COLUMN EDITOR"** — The context ID must be for a browse.
- **"EXTERNAL-TABLES"** — The context ID must be for a procedure.

### Input/Output Parameters

The adeuib/_uib_dlg.p procedure uses the following input/output parameters:

*pc_args*

A character string to send to the dialog box. For the Column Editor or the External Tables dialog box, you can send a *table-list*, which is a comma-separated list of tables. For the Query Builder, you can specify a *mode*, which is a comma-separated list having the following syntax:

**SYNTAX**

```
{ ? | NORMAL | QUERY-ONLY | CHECK-FIELDS }
  [ NO-FREEFORM-QUERY ]
```

- ? — The default. Changes the Query Builder to NORMAL mode. For queries, this means that the Fields button is disabled; for browsers, it means that all fields that are Query Builder options are enabled.
- **"NORMAL"** — Changes the Query Builder to NORMAL mode. For queries, this means that the Fields button is disabled; for browsers, it means that all fields that are Query Builder options are enabled.
- **"QUERY-ONLY"** — Specifies that the Query Builder be in QUERY-ONLY mode. This means that the fields button is disabled.
- **"CHECK-FIELDS"** — Specifies that the Query Builder be in CHECK-FIELDS mode. This means that if you try to leave the Query Builder and have not selected any fields, you are prompted to select fields.
- **"NO-FREEFORM-QUERY"**— Specifies that the Freeform Query button must be disabled.

## C.4.11 Window Save (adeuib/_winsave.p)

Use the adeuib/_winsave.p procedure to set an an internal flag that indicates whether the window has been modified since it was last saved or opened. AppBuilder checks this flag whenever it closes a window:

- If the flag is TRUE, the window has recently been saved (or just opened).
- If the flag is FALSE, the window needs to be saved.

Whenever you modify something in the design window that will be reflected in its saved procedure file, call adeuib/_winsave.p to set the flag to FALSE:

### Calling Sequence

```
RUN adeuib/_winsave.p
  ( INPUT ph_handle ,
    INPUT pl_saved
   ) .
```

### Input Parameters

The adeuib/_winsave.p procedure takes the following input parameters:

*ph_handle*

The widget-handle of the design window to mark as saved.

*pl_saved*

A logical value that indicates that the window has been saved or opened since it was last modified.

### Example

```
RUN adeuib/_winsave.p(h_deswin, FALSE).
```

## C.4.12 Reserved Names (adeuib/_namespc.p)

Use the adeuib/_namespc.p procedure to register the names of variables or procedures that AppBuilder users cannot use. For example, if you write an XFTR that creates a procedure, you do not want an AppBuilder user to create an internal procedure with the same name. Similarly, if the XFTR defines variables, you want to restrict AppBuilder users from using those variables:

### Calling Sequence

```
RUN adeuib/_namespc.p
  ( INPUT pi_context ,
    INPUT pc_mode ,
    INPUT pc_list
   ) .
```

### Input Parameters

The `adeuib/_namespc.p` procedure takes the following input parameters:

*`pi_context`*

The context of the procedure object to access. If this is unknown (?), then get the current procedure. If a procedure object cannot be found, returns "ERROR".

*`pc_mode`*

The action to perform. Valid actions are as follows:

**SYNTAX**

```
{ RESERVE | UNRESERVE }
[ { VARIABLE | PROCEDURE }
      [ NAME | NAMES ]
]
```

*`pc_list`*

A comma-separated list of names to add or remove.

If you try to reserve a name that is already in use, warnings are generated; however, the reservation is made, so that if you delete the widget that uses the name, you cannot use it again. You can unreserve any name, even one that it not reserved, but no warning is generated.

## C.5 Example Using AppBuilder's API

The following example shows how to call AppBuilder's API procedures:

*(1 of 2)*

```
/* This example shows how to replace the string "v-credit" to      */
/* "v-debit" in a trigger code block of a radio-set named rs-disc  */

DEFINE VARIABLE char-hndl    AS CHARACTER                    NO-UNDO.
DEFINE VARIABLE my-widget    AS CHARACTER                    NO-UNDO.
DEFINE VARIABLE trig-context AS INTEGER                      NO-UNDO.
DEFINE VARIABLE trig-code    AS CHARACTER                    NO-UNDO.

/* Get the context-id of the radio-set called rs-discounts         */
RUN adeuib/_uibinfo.p
   (INPUT ?,                    /* We don't have the context id but
                                   the radio-set is in the current
                                   design window                   */
    INPUT "rs-disc",            /* The name of the radio-set       */
    INPUT "CONTEXT",            /* Please return the context id    */
    OUTPUT my-widget).          /* This is the context-id we need  */

/* Fetch the trigger code to be modified                           */
trig-context = ?.               /* This is IMPORTANT, otherwise
                                   _accsect.p will attempt to use
                                   this trig-context instead of
                                   my-widget and
                                   TRIGGER:VALUE-CHANGED           */
RUN adeuib/_accsect.p
   (INPUT "GET",                /* We want to read the code        */
    INPUT INTEGER(my-widget),   /* The context id from above       */
    INPUT "TRIGGER:VALUE-CHANGED", /* The trigger we want          */
    INPUT-OUTPUT trig-context,  /* The trigger context so we can
                                   reference this block directly
                                   in the future. IMPORTANT: This
                                   must be ? intially or _accsect
                                   will attempt to use it instead
                                   of parameter 2                  */
    INPUT-OUTPUT trig-code).    /* The code block                  */

/* Replace the string */
trig-code = REPLACE(trig-code, "v-credit", "v-debit").
```

*(2 of 2)*

```
/* Store the trigger */
RUN adeuib/_accsect.p
   (INPUT "SET",                /* Write the code back             */
    INPUT ?,                    /* We have code context, use it    */
    INPUT ?,                    /* We have code context, use it    */
    INPUT-OUTPUT trig-context,  /* We are using it directly        */
    INPUT-OUTPUT trig-code).    /* We are writing this back        */

/* Mark the window as having been modified since last save         */

/* First get the handle of the design window                       */
RUN adeuib/_uibinfo.p
   (INPUT ?,                    /* We don't know the context id    */
    INPUT "WINDOW ?",           /* We want the handle of the
                                   design window                   */
    INPUT "HANDLE",             /* We want the HANDLE              */
    OUTPUT char-hndl).          /* Returns a string of the handle  */

/* Now indicate that the window needs to be saved again            */
RUN adeuib/_winsave.p (WIDGET-HANDLE(char-hndl), FALSE).
```

# Glossary

**4GL**

Acronym for Fourth-Generation Language. The term used for Progress Software Corporation's application programming language.

**ActiveX**

Microsoft's registered trademark for the third version of Object Linking and Embedding (OLE) controls. ActiveX objects conform to the Microsoft Component Object Model standard. ActiveX objects are still often called OCX objects.

**API**

Application Programming Interface. The specification for the run-time library.

**Asynchronous**

Lacking regular timing, not occurring at regular intervals. Most higher-level human activities are asynchronous.

**Atomic**

A term used to refer to inseparability. An atomic operation is one that must be done all at once.

**Basic Object**

A 4GL object; a user-interface element defined in the 4GL itself. Basic objects are distinct both from SmartObjects and from ActiveX objects.

**Buffer**

A multi-byte block of memory used for temporary storage.

**Class**

The term used, specifically in object-oriented programming, to describe an abstract object; a synonym for type or category. In programming, a class definition includes both data and behavior definitions.

**Client/Server Application**

A single application divided into two or more pieces for efficiency. In general, the client piece handles user-interface issues, and the server piece handles database issues. Clients and servers typically run on different computers and communicate over a network, but this is not essential. One server may support many clients. Clients are sometimes referred to as being “thin” or “thick”. Thin clients do less processing locally; the thinnest clients may be little more than terminals.

**Corruption**

Lost integrity, or the process of losing integrity. Data corruption can occur on two levels: physical and logical.

Physical corruption means the data itself has been inappropriately altered, for example by a hardware failure causing a dropped or stuck bit that creates a value 0 or 3 in place of a 1.

Logical corruption means different elements inappropriately get out of sync with one another, often because a multi-step operation was neither fully completed nor fully undone. No physical corruption is involved.

**Data-driven**

The quality of being guided by external data, rather than having all guidance within the program itself. The behavior of an interpreter can be thought of as data-driven.

**Data Stream**

The data that flows into and out of an application. A description of the data stream fully identifies the database, the tables, and the fields involved, as well as their ordering and permissions.

**Default**

The term that refers to what you get if you do not choose something else. Not everything has a default value.

**Design Window**

An AppBuilder synonym for workspace, but only for objects with a visible run-time representation. The workspace for a non-visible object is not a design window, but rather the Section Editor combined with a tree view of editable code sections.

**Display Buffer**

The area of memory used by the system to assemble and store text or video images to be displayed on terminal or monitor. Display-buffer sizes range from the 2KB text buffer of a traditional timesharing terminal to the 8MB (or larger) graphics buffer on a video adapter card.

**Drag-and-Drop Layout**

The method by which most layout editors work. The designer selects tool icons representing the various possible user-interface components and drag-and-drops them onto a workspace, visually arranging them into their final positions and sizes.

**Event**

Some occurrence defined as being interesting. Events may be generated by hardware or software. If you wish to handle an event in your application, you write a procedure usually known as a trigger, and attach it to that class of event. When a event of that type occurs, your trigger code is automatically executed.

**External Procedure**

A Progress-specific term, also called a procedure file. An external procedure is some amount of code, typically including functionally-related internal procedures and functions, that is stored in a file on disk. External procedures are identified by their filenames.

**Fit-and-Finish**

The elements that combine to create in the user a feeling that the application was or was not well-designed and carefully written.

**Free-form Query**

A query written by hand, rather than by using Query Builder. Freeform queries can have any structure the author likes, or none. Query Builder generates queries with a particular structure, and you can find it difficult or impossible to create certain kinds of unusual queries with it.

**Fully-qualified**

The state of being described uniquely. A fully-qualified database field reference is one that can map to only a single, specific field in a specific database, not to a field with the same name in some other database.

**Function**

As used by many C programmers: any subprogram. Specifically a subprogram identified by the reserved word FUNCTION. The way the Progress programming language uses the term is closer to the use in Pascal than in C. Functions always return a value (unlike procedures).

**Glue**

Code that helps integrate other, more specialized code. For example, the code that copies the results of a transformation from one place to another is usually thought of as glue. Also called supporting code.

**Granularity**

A term referring to the size of a unit of analysis, intervention, etc.

**Gridline**

The AppBuilder term for one of the imaginary lines that cover a Frame object at regular intervals, horizontally and vertically. Gridlines exist only at design time. You can set the intervals at which they occur, and can make them visible or invisible, and magnetic or not magnetic.

**Icon**

Generally, a non-text (graphic) symbol or an image used as a symbol.

**Identifier**

The name given an object, variable, etc. and used in code.

**Inherit**

In an object-oriented environment, to automatically get properties and, perhaps, values from an ancestor object or class upon creation.

**Instance**

The term used in object-oriented programming for a specific object, allocated in memory, that represents a class. A class is unique, but you can normally create an unlimited number of instances that are members of a particular class. The memory allocation for an instance is typically only enough for the instance's data; all instances of a object type share the same executable code.

**Internal Procedure**

A Progress-specific term. An internal procedure is a subprogram identified by the reserved word PROCEDURE. Also used more generally to refer to any PROCEDURE or FUNCTION in the abstract.

**Layout Editor**

In a RAP/RAD environment, the tool that allows a user-interface designer (or programmer) to quickly arrange user-interface elements within a context. Generally, layout editors resemble graphics editors, to which they are related.

**Look-and-Feel**

The industry-standard term referring to the style of an application and how it is put together.

**Master**

In ADM terms, a fully-qualified SmartObject, able to have instances made from it.

**Message**

A well-defined block of data that serves as a unit of communication between two program entities. The two principal ways of passing messages are by putting them in an agreed location (mailboxing), or by sending them as a packet over the network if the sending and receiving programs are on different systems.

**Method**

In object terminology a procedure or function, defined in a class, that implements a particular object behavior. In some object languages, methods are further divided into Public (callable from outside the object instance) and Private (callable only by another method within the same instance).

**Modal**

When referring to Windows and Dialog Boxes, the state of capturing all input. Because a modal window captures all user input, it must be dismissed before any other window can get input focus.

**Multiplexing**

Reusing the same resource for different tasks. In the context of application design, multiplexing most often involves the reuse of display space.

**Object-Oriented**

One of the two principal models of computer program, the other being the functional or procedural model. In the O-O model, an object combines state information (data) with a behavioral repertoire (procedures and functions) under a single identifier. Object programs tend by nature to be asynchronous, event-driven, and multi-pathed.

**Overlay**

A type of multiplexing in which the same area of memory is loaded with different program modules as they come into use. Designing for overlay requires that the sections to be overlaid be independent of one another, since only one can occupy the designated area of memory at a given time.

**Procedure File**

In 4GL terminology, a file that represents an external procedure.

**Procedure-Oriented**

One of the two principal models of computer program, the other being the object or object-oriented model. In this model, procedures are like stations on an assembly line, receiving data and transforming it before handing it off to the next station down the line. Procedural programs are able to handle both synchronous and asynchronous problems.

**Procedure, Structured**

In the AppBuilder environment, a file-level aggregation of source code—an external procedure— that was created and is maintainable by AppBuilder. AppBuilder requires that such files have a specific internal structure. The name has nothing to do with the concept of structured programming.

**Property**

A synonym for attribute or characteristic. Properties are usually represented by some variable value. In object programming, properties are defined at the class level and set at the instance level. The system uses the current values to modify the appearance or behavior of the object instance to which they belong.

**Query**

The code, typically in a non-procedural language, that describes a desired data stream.

**Race Condition**

The state of indeterminacy and uncertainty that results when an outcome is different depending on which of two blocks of code executes first, and the order of execution is not stable. The indeterminate nature of a race condition creates problems that may be **extremely** hard to solve or even reproduce.

**RAD**

Acronym for Rapid Application Development. A general term referring to tools and methods that help reduce application development time when compared to traditional hand-coding. See also RAP.

**RAP**

Acronym for Rapid Application Prototyping. A general term referring to tools and methods that help programmers produce prototype applications quickly for customer review and approval prior to beginning full-scale development. See also RAD.

**Signature**

In the ADM, a coded string that uniquely identifies a certain data stream. If the prospective sender and receiver do not have matching signatures for the data stream, it cannot be transferred between them because of the corruption that would occur.

**SmartContainer**

The ADM class that supports paging, and the placement of child objects. All the Smart organizers are members of class SmartContainer, but not all SmartContainers have an organizer role.

**SmartObject**

Progress Software Corporation's term for objects written to its Application Development Model standard.

**Subprogram Module**

A well-bounded section of code, often written as a Procedure or Function, that addresses some part of a program's overall task.

**Supporting Code**

Code that integrates other code. Supporting code is more often referred to as "glue".

**Syntax**

The term that refers to the grammatical structure of an expression as distinct from its meaning (semantics).

**Temp-Table**

The 4GL term for a temporary table, often created at the application level to subset the main database and so reduce the processing load created by traversing irrelevant records.

**Template**

A pattern for constructing something. In programming terms, a template is a block of incomplete code. It may be literally incomplete, or be complete but need revision for current purposes.

**THIS–PROCEDURE**

The 4GL handle that identifies the currently-executing procedure. In AppBuilder, THIS–PROCEDURE typically refers to the main organizer object being worked on. For example, when looking at a Window's SmartLinks, THIS–PROCEDURE always refers to the SmartWindow itself, not to any of the objects contained in it.

**Transaction**

A group of related operations that must be completed atomically. If any part of the transaction cannot be completed, all completed parts must be undone before execution can be allowed to proceed. A transaction left partly completed causes logical corruption that may be very hard to resolve without extensive and costly analysis.

**Trigger**

The general 4GL term for an event-handler procedure.

**Usability**

The characteristic of being easy and pleasant to learn and use. Software with high usability is easy to learn for anyone able to perform the underlying task, easy to use for both novice and expert, enhances productivity for both novices and experts, and has a positive effect on every user's sense of competence.

**User Interface**

The controls, methods, and processes made available to the user of (in this context) software for the purpose of doing work. User interfaces range from a blinking command-line cursor and dozens or hundreds of cryptic commands, to sophisticated simulations that are intuitively easy to use for anyone with the appropriate real-world experience.

**VIEW–AS**

A reserved term in the 4GL. You use it to indicate to the compiler the type of widget you wish to have generated to display some data.

**Widget**

To programmers, the generic term for any user-interface element.

**Workspace**

The place where you design and assemble the elements of your application. In AppBuilder, a workspace may involve visual layout, in which case the workspace itself—character Window or GUI Dialog Box, for example—becomes part of the result.

# Index

## A


Actions
  check source-code syntax 2–9
  choose colors for current object 2–11
  close character-mode test window 2–9
  configure alternate layouts 2–13
  connect to database 2–11
  copy selected object 2–7
  create new ADM class 2–10
  create new object file 2–5
  cut selected object 2–7
  debug current workspace 2–9
  delete an object instance 2–8
  disconnect database 2–11
  edit file-level properties 2–11
  edit traversal list 2–8
  edit widget properties 2–10
  export an object 2–8
  hide tool buttons 2–17
  hide/show Objects Palette 2–14
  insert whole file 2–8
  modify database connections 2–11
  open ActiveX property sheet 2–15
  open error-message window 2–15
  open existing object file 2–6
  open online help window 2–15
  open Section Editor window 2–14
  paste object 2–8
  place object in workspace 2–22
  place several objects of same type 2–22
  print object source 2–6
  review recent Cue Cards 2–15
  review recent messages 2–15
  run current workspace 2–9
  save all open files 2–6
  save object source file 2–6
  save object source under another name 2–6
  set Design Page 2–8
  set preferences 2–12
  start non-AppBuilder editor 2–10
  stop or undo object placement 2–22
  toggle layout-grid stickiness 2–12
  toggle layout-grid visibility 2–12
  undo latest operation 2–7
  view/modify common properties 2–14

ActiveX (OCX) object 8–2
  adaptive interface 8–2
  configuring 8–4
  creating and placing 8–3
  event handlers
    generic 8–9
  in master layout B–4
  properties stored in file 8–10
  tool 2–20
  using in application requires licence 8–2, 8–3

ActiveX (OCX) Spin Buttons
  example of use 8–4

ActiveX (OCX) Timer 8–15
  example of use 8–16

ActiveX object
property references in code 5–60

activex.cst C–3

ADM 1–3
architecture components 1–3
current version 1–3
paging 3–31

ADM Pages manager A–4
deleting a whole page A–6
deleting an object from a page A–6
exchanging contents of pages A–6

Advanced Procedure Settings A–7

Advanced Procedure settings
child objects, specifying allowed A–8

Alternate layout
definition B–4

Alternate layouts. *See* Layouts

API
add/remove SmartLinks C–25

AppBuilder
API C–23 to C–47
add/remove SmartLinks C–25
context IDs C–23
create objects C–40
create reserved identifiers C–45
delete objects C–42
generate table references C–32
get widget information C–33
invoke certain dialogs C–43
list of procedures in C–24
manage code sections C–27
set window-modified flag C–44
character-mode layout, support for B–18
concepts & terminology 1–4
external procedures 1–4
objects 1–4
custom object files C–2
customizing 1–20, 2–23, 2–28, C–2
default function type 2–25
double-click 2–25
introduction to C–2
layout grid 2–27
objects in object files C–3
designing for character environments B–18
event handling C–21
exiting 2–2
main window 2–3
programming interface C–23
setting personal preferences 2–23
starting 2–2
testing character-mode designs B–19
version number 2–15
WebSpeed, for 2–2
working with 1–16, 1–18

AppBuilder main window
buttons 2–16
current object fields 2–3
menu bar 2–4
status bar 2–4
tool bar 2–16

AppBuilder-maintained procedures C–32

Application
design considerations
single vs multi-window 1–17

Application Development Model. *See* ADM

AppServer
enabling support for A–21

Attributes, object. *See* Properties

Audience xvi

## B

B2B messaging
creating consumer instances 6–36
message types 6–35
SmartConsumers 6–32
SmartProducers 6–32
SonicMQ domains 6–37

Batch mode
containers for 3–27
operating in 6–39
SmartConsumesr 6–39

Bold typeface
as typographical convention xviii

Broker URL C–39

Buttons
AppBuilder main window 2–16

## C

Calculated fields
creating 4–45

Caution
guard against version skew 6–4
mapping to non-unique values 6–9
must use static data object 4–16
no type-checking in maps 6–6
objects incompatible 4–18
users might be confused 4–27
visible objects in non-visible container 3–29

Cautions
data validation 4–13
input focus 5–71
preventing object from gaining 5–33, 5–55
race condition 8–16
too many events to handle 8–16

Changing the data stream 4–33

Character applications in AppBuilder B–18, B–19

Character client B–20

Character mode
designing for 1–17

Character-mode layouts
double byte characters B–20
limitations on inheritance in B–9
running B–19
simulating B–18

Check Box 5–85
*See also* Toggle Box

Checking syntax 2–9

Child objects in workspaces
specifying which are allowed A–8

Code
testing 2–55

Code base
single, for different environments B–2

Code sections
navigating 2–38
reading and writing C–27 to C–32
Runtime attributes and AppBuilder settings B–12

Colors
choosing A–11
defining new A–12
how they are applied A–11
table of pre-defined A–12

Combo Box 5–50
ActiveX (OCX)
configuring 5–59
creating and placing 5–59
tool 2–20
basic 4GL
configuring 5–51
creating and placing 5–50

initial value of 5–54
tool 2–19
Drop-down style, defined 5–52, 5–61
Simple style, defined 5–52, 5–61

Common properties
displaying in a dialog A–21

Compiling and running 2–9

Configuring
SmartB2BObject instances 6–18
SmartBusinessObjects 4–19, 4–25
SmartConsumer instances 6–37
SmartProducer instances 6–34

Context IDs C–23

Control-frame properties A–3

Copying text
Section Editor 2–39

Creating
document-handling packages 6–29
document-routing packages 6–30
master SmartB2BObjects 6–16
Simple SmartContainers 3–28
SmartB2BObject instances 6–17
SmartBusinessObjects 4–18
SmartConsumer instances 6–36
SmartProducer instances 6–33
SmartSender instances 6–23
XML map file 6–11

Current object fields
AppBuilder main window 2–3

Custom Lists
assigning meaningful names A–15

Custom object files 2–23, C–2 to C–47

Custom objects
settable properties C–11 to C–18

Customizing AppBuilder C–2 to C–47

Cutting text
Section Editor 2–39

## D

Data access
AppServer connections 4–26
client-only object versions
filenames end in '_cl' 4–3
client-only version 4–3
efficiency versus completeness 4–20
foreign-field mapping 4–23
generic/general-purpose 4–4
partitions 4–25, 4–26
reducing complexity 4–17
relational mapping 4–23
single-transaction updates 4–16
synchronizing 4–18

Data Browser
basic 4GL 5–44
configuring 5–45
creating and placing 5–44
editing query and fields 5–46
tool 2–18
Smart 5–3
configuring 5–3, 5–10
creating and placing 5–3, 5–18
creating master 5–6
editing data stream in master 5–16
tool 2–21

Data fields
differences in appearance 5–25

Data Fields. *See* Data Viewer

Data pump
Smart
example of use 1–20
tool 2–20

DATA SmartLinks 4–20

Data stream
allowing user to filter 4–27
changing 5–16
changing contents of 4–33
preserving integrity of 1–15
selecting fields for browsing 4–42

Data transfer
SmartProducers/SmartConsumers 6–32

Data transformation
SmartB2BObjects 6–2
SmartSenders/SmartReceivers 6–22

Data validation
limitation with SmartDataObject 4–13

Data Viewer
basic 4GL 5–62
configuring 5–63
creating and placing 5–63
tool 2–18
Smart 5–21
adding/removing fields 5–27
appearance of fields 5–25
configuring 5–28
creating master 5–22
example of use 1–19, 3–39
tool 2–21

Database
add, modify, remove connections 2–11
connections for character-mode clients B–20
field names in Section Editor 2–43
selecting fields for browsing 4–42

Database tables
working copies A–31

Default Page 3–33
at startup 3–34

Deleting
whole procedure, trigger, etc. 2–43

Design Page
changing 3–33
limitations 3–32

Design windows 2–35

Development mode switch. *See* Remote development button

Dialog Box
basic 4GL
configuring 3–22
creating new 2–5
Smart
configuring 3–16
creating new 2–5
file-level properties, configuring 3–18

Dialogs, frequently-used
ActiveX (OCX) control frame properties A–3
ADM Pages manager A–4
advanced file-level properties A–7
Advanced Procedure Settings A–7
base procedure type A–7
method-libraries reminder, avoiding A–7
supported SmartLinks A–7
Color Selector and editor A–10
database field selector A–16
method library include-files list A–18
naming custom lists A–15
Procedure Settings A–19
Properties Window A–21
SmartInfo A–25
SmartLinks editor A–22
String Attributes A–28
Temp-tables Maintenance A–31
Text-style selector and editor A–34
Traversal-path editor A–30

Documents
XML mapping 6–3

Drop-down List
as style of Combo Box 5–52, 5–61
defined 5–52

Dynamic SmartDataObjects ?? to 4–16

## E

Edit
menus 3–11

Editing
ActiveX object properties 2–15
importance of using Section Editor 2–36
object properties 2–52
open editor window 2–14

Editor object
multi-line
buffer size 5–65, 5–66
configuring 5–64
creating and placing 5–64
tool 2–19
single-line
configuring 5–68
creating and placing 5–68
tool 2–20

Editor object, single-line 5–67

Ending AppBuilder session 2–2

Environment
character
designing for 1–17

Error messages
displaying descriptions xxiii

Event handlers
for XFTRs C–21 to C–23

Event names
in Section Editor 2–45

Examining procedure files 2–53

Example of use
data pump
Smart 1–20
Data Viewer
Smart 1–19, 3–39
Pushbutton 1–20, 3–39, 3–41
Smart 1–19
Radio Set 3–39
Rectangle 1–20, 3–39
SmartContainer tool 3–41
SmartDataField 5–41
SmartDataObject 1–20
SmartDataViewer 1–19, 3–39, 5–41
SmartFilter 7–5
SmartFolder 3–34
SmartLinks 1–19, 1–20
SmartPanel 1–19, 3–34, 3–39
SmartSelect 5–41
SmartWindow 1–19, 3–34
Spin Buttons (ActiveX) 8–4
Static Text 1–20
Timer object (ActiveX) 8–16

Extended features C–19
reserving names C–45

## F

Field lists
specifying 4–41

Fields
creating calculated 4–45
selecting for browsing 4–42

Field-selector dialog A–16

File-level properties A–19

Filename conventions
SmartB2BObjects 6–17
SmartBusinessObjects 4–19
SmartContainers 6–31
SmartSenders 6–24

Filenames
Section Editor 2–52

Files
custom object, contents of C–3
custom object, listed C–3
database tables
working copies A–31

Fill-in 5–67
configuring 5–68
creating and placing 5–68
tool 2–20

Filtering incoming messages 6–19, 6–25

Frame
basic 4GL
configuring 3–23
tool 2–18
creating 3–23
Smart
configuring 3–19
creating new 2–5

Functions
listing C–38

## H

Help 2–15
Progress Messages xxiii
review recent Cue Cards 2–15
review recent messages 2–15
see AppBuilder version 2–15
see system resources 2–15
with message numbers 2–15

HTML Mapping Object C–39

## I

Image object 8–10
*See also* Static Image

Include-files
list of A–18

Inheritance of layouts
preventing B–9

INMESSAGE SmartLinks 6–36

Instances
getting information C–32 to C–40

Inter-business communication 6–32

Internal procedures
listing C–38

Internationalization
no-translation attribute A–28
specifying extra space in literals table A–28

Italic typeface
as typographical convention xviii

## J

Java-based messaging 6–32

JMS Partitions 6–37

Joins
queries 4–38

## K

Keystrokes xviii

## L

Layout
align objects automatically 2–13, 2–14
alternate
creating B–7
definition B–4
configure alternate 2–13
definition B–2
master
definition B–3

elements of B–4
multiplexing 3–31
paging 3–31
selecting pre-defined alternate B–8
standard B–4
toggle grid stickiness 2–12
toggle grid visibility 2–12

Layouts
alternate B–16
Combo Boxes in B–17
modifiable properties in B–16
Static Text objects in B–17
design time view B–5
how they work internally B–11
inheritance from master B–8 to B–11
limitations B–9
preventing B–8, B–9
restoring B–9, B–10
limitations in alternate B–16
master
re-establishing inheritance from B–10
propagation of master changes B–8 to B–11
run-time customization B–17
SmartObject
applying at run time B–27
changing at run time B–29
creating B–21
initial run time B–28
instance options B–24
modifying B–22
SmartObjectLayouts B–21
switching at run time B–17

Limitations
data validation 4–13
double-byte characters B–20
ObjectNames in resource list 6–12
of data-stream integration 4–16
of static SmartDataObjects 4–3
SmartFolder 3–35
SmartRouters/SmartReceivers 6–29
SmartToolbar 7–2
XML mapping support 6–11

Line object
using Rectangle to simulate 3–24

List button
AppBuilder main window 2–16

Lock status
for inserting objects 2–17

Look-and-feel
3D 3–9
automatic resizing 5–32, 5–66
border thickness 3–25, 3–26
borders between rows and columns 5–48
column width 5–16
display as static text 5–33, 5–71
equal spacing 5–75
extend final column 5–13, 5–47
flat, unbordered style 7–15
hide scrollbar 5–14, 5–48, 5–78
horizontal vs vertical layout 5–75, 5–81
iconic label 7–16
keep existing layering of children 3–9
max button in title bar 3–9
message area 3–9
min button in title bar 3–9
minimal title bar 3–10
number of lines 5–52
numeric readout 5–82
re-arranging fields 5–29
remove from layout 5–33, 5–53, 5–67, 5–71, 5–75, 5–78, 5–82, 5–88, 8–14, A–3
reset to master layout properties A–10
re-using part of the window 3–31
reverse value progression 5–82
scroll bars 3–10
show border 5–14, 5–47, 7–4
show column labels 5–14, 5–48
show filled 3–26
show internal borders 5–14
show label 5–32, 5–40, 5–52
show title bar 5–14, 5–48

sorted data 5–39, 5–78
special highlighting 3–21, 3–27, 5–34, 5–55, A–10
state determines appearance 5–25
status area 3–10
sync with master layout 3–15
system colors instead of exact colors 7–15, 8–13
text label 7–16
thicker default-button border 7–15
title bar 3–9, 3–20
toolbar component 7–4
type of widget used 5–40, 5–52, 5–61
undefined value 5–39
visible border 3–20
visible signs of focus 7–16

## M

Macros A–15

Manual
syntax notation xix

Manual, organization of xvi

Master layout
*See also* Layouts
definition B–3
elements of B–4

Menu
creating 3–11
popup, creating 3–11

Menu bar
AppBuilder main window 2–4
creating 3–11

Message destination identifiers 6–19

Message system login 6–38, 6–39

Message types 6–35

Message-handling
algorithms 6–40
code samples 6–40
inbound process 6–42
internal details 6–40
outbound process 6–40

Messages
displaying descriptions xxiii

Monospaced typeface
as typographical convention xviii

Multi-line Editor 5–64

Multiple document types 6–27

Multiple document types/destinations 6–20

Multiple layouts
*See also* Layouts
coding B–12
internals B–11
reason for using B–2

## N

New object 2–5

## O

Object
ActiveX (OCX)
configuring 8–4
creating and placing 8–3
introduction to 8–2
ActiveX (OCX) timer 8–15
Check Box 5–85
choose colors 2–11
Combo Box 5–50
Combo Box (ActiveX)
configuring 5–59
creating 5–59
Combo Box (basic 4GL)
configuring 5–51
creating 5–50
Data Browser (basic 4GL) 5–44
configuring 5–45

creating and placing 5–44
Data Browser (Smart) 5–3
configuring 5–3, 5–10
creating and placing 5–3, 5–18
creating master 5–6
editing data stream in master 5–16
Data Fields 5–62
Data Viewer (basic 4GL) 5–62
configuring 5–63
creating and placing 5–63
Data Viewer (Smart) 5–21
adding/removing fields 5–27
configuring 5–27, 5–28
creating master 5–22
data-capture
introduction to 5–2
data-display
introduction to 5–2
deleting instance 2–22
Dialog Box (Smart)
configuring 3–16
file-level properties 3–18
Drop-down List 5–50
editing file-level properties 2–11
editing properties 2–52
Editor (multi-line) 5–64
configuring 5–64
creating and placing 5–64
Fill-in 5–67
configuring 5–68
creating and placing 5–68
Frame (basic 4GL)
configuring 3–23
creating and placing 3–23
Frame (Smart)
configuring 3–19
names in Section Editor 2–51
open 4GL Property Sheet 2–10
participating in traversal list 7–16
Pushbutton 7–12
configuring 7–13
creating and placing 7–13
Smart array 7–8
Query 4–34
creating 4–34
Radio Set 5–72
configuring 5–73
creating and placing 5–72
initial value 5–74
Rectangle
configuring 3–24
creating and placing 3–24
repeating a placement 2–22
Selection List 5–75
configuring 5–76
creating and placing 5–76
single-line Editor 5–67
Slider Control 5–79
boundary values 5–80
configuring 5–80
creating and placing 5–79
initial value 5–80
tic marks 5–80
Smart, data-driven list 5–36
SmartDataBrowser 5–3
configuring 5–3, 5–10
creating and placing 5–18
creating master 5–6
dynamic 5–3
editing master 5–16
static 5–6
SmartDataField 5–34
placing and configuring 5–35
SmartDataObject 4–2
configuring 4–11
creating master 4–4
SmartDataViewer 5–21
adding/removing fields 5–27
configuring 5–28
creating master 5–22
SmartDialog
configuring 3–16
creating instance 3–4
SmartFilter 4–27
SmartFrame
configuring 3–19
creating instance 3–4
SmartObject template 8–2
SmartPanel
configuring 7–9
creating and placing 7–9
SmartSelect 5–36
configuring 5–38
creating and placing 5–37
SmartToolbar 7–2
configuring 7–4
creating and placing 7–3

creating new master 7–5
SmartWindow
configuring 3–6
creating instance 3–4
opening property sheet for 3–6
special-purpose, where to find 2–22
Spin Buttons
example of use 8–4
Spin Control (ActiveX)
configuring 5–83
creating and placing 5–83
SpinControl (ActiveX) 5–82
Static Text 8–14
Toggle Box 5–85
configuring 5–86
creating and placing 5–85
Window (Smart)
configuring 3–6
workspace 2–35

Object behavior
accept dropped objects 3–9, 5–13, 5–32, 5–47, 5–53, 5–66, 5–70, 5–74, 5–78, 5–81, 5–87, 7–15
allow automatic re-sizing 5–32, 5–66, 5–70, 5–77, 5–81, 5–87
allow multiple selections 5–13, 5–47, 5–78
allow navigation between records 7–4
allow transactions 7–5
allow undefined value 5–39
allow user to change column width 5–16, 5–49
allow user to edit data 5–13, 5–33, 5–67, 5–71, 7–4
allow user to move 3–21, 3–27, 5–16, 5–34, 5–49, 5–55, A–10
allow user to re-order columns 5–15, 5–49
allow user to re-size 3–10, 3–21, 3–27, 5–16, 5–34, 5–49, 5–55, A–10
allow user to search on columns 5–16, 5–49
allow user to select 3–21, 3–27, 5–16, 5–34, 5–49, 5–55
appear at specific location 3–9
automatically clear field for input 5–32, 5–70
automatically populate fields 3–17, 5–14, 5–32, 5–39, 5–48, 5–53
automatically sort data 5–39, 5–54
automatically validate input 3–17, 5–14, 5–48
automatically wrap lines 5–67
automatically write out data changes 5–14, 5–47
cause ENDKEY condition 7–15
cause GO condition 7–15
change object scope 5–33, 5–53, 5–67, 5–71, 5–75, 5–78, 5–82, 5–88
create scrollbars when needed 3–10
data type 5–52, 5–69, 5–73, 5–77
decline input focus 3–10, 3–26, 5–32, 5–39, 5–47, 5–53, 5–66, 5–70, 5–74, 5–78, 5–81, 5–87, 7–15, 8–13, A–3
disallow min/max resizing 3–10
discard leading spaces 5–32, 5–70
display as (widget type) 5–40
display context-sensitive help A–3
display data type from schema 5–41
display field descriptor from schema 5–41
display MESSAGE output 3–9
display minimized in/above taskbar 3–10
display special help string 5–15, 5–49, 5–54
display status messages 3–10
display tooltip 5–41, 5–52, 5–65, 5–70, 5–74, 5–77, 5–81, 5–87, 8–13
emulate platform-native object 5–32, 5–71
format display 5–52, 5–69
indent next line 5–66
indicate input focus 7–16

initialize in hidden state 3–9, 3–26, 5–13, 5–32, 5–47, 5–53, 5–66, 5–71, 5–75, 5–78, 5–81, 5–87, 7–15, 8–13, A–3
initialize with hidden data 5–66, 5–70, 5–77, 5–81, 5–87
interpret ESC as CHOOSE 7–15
journal changes 5–33, 5–53, 5–67, 5–71, 5–75, 5–78, 5–82, 5–87
maintain child-object layering 3–9
minimize to icon 3–9
move to next field when full 5–32, 5–70
page between different layouts 3–31
page length 3–20
participate in traversal list 5–14, 5–33, 5–48, 5–53, 5–67, 5–71, 5–75, 5–78, 5–82, 5–87, A–3
predict user input 5–53, 5–54
prevent input echo 5–32, 5–70
provide scrollbars 5–14, 5–67
receive all RETURN events 7–15
resize to full screen 3–9
scroll by character or whole field 5–13, 5–47
stay in front of other objects 3–15
support SmartFilter in separate window 7–5
use help strings and validation from dictionary 3–17
use help strings and validation rules from dictionary 5–14, 5–48

Objects
action control
listed 1–5, 1–7, 1–10
ActiveX (OCX)
listed 1–11
adding template to New-object dialog C–8
adding to a tool menu C–6
basic 4GL
listed 1–5
control
introduction to 7–2
creating and placing 2–22
data-access
listed 1–8
data-display and -capture
listed 1–5, 1–9, 1–11
defined 1–13
Dialog Boxes
basic 4GL 3–22
Smart 3–15
for controlling actions 7–2
Frames
basic 4GL 3–22
Smart 3–19
organizer
creating procedure-based instance 3–4
introduction to 3–2
listed 1–5, 1–8
roles 3–3
other
listed 1–7, 1–10, 1–11
Rectangles 3–23
Smart
listed 1–8
SmartDialogs 3–15
SmartFrames 3–19
SmartPanel 7–8
SmartWindows 3–5
Static Image 8–10
visible vs non-visible 1–12
Windows
basic 4GL 3–22
Smart 3–5

Objects Palette 2–17
ActiveX (OCX) tool 2–20
cancelling tool selection 2–22
Check-Box tool
*See* Toggle Box
Combo-Box tool
ActiveX 2–20
basic 4GL 2–19
customizing 2–23
Data-Browser tool
basic 4GL 2–18
Smart 2–21
Data-Fields tool 2–18
Data-pump tool
Smart 2–20

Data-Viewer tool
basic 4GL 2–18
Smart 2–21
Dialog Box tool
basic 4GL 2–17
Smart 2–17
Editor (multi-line) tool 2–19
Editor (single-line) tool 2–20
Fill-in tool 2–20
Frame tool
basic 4GL 2–18
Smart 2–17
hide tool buttons 2–17, 2–23
hide/show 2–14
keep in front 2–23
multiple placement 2–22
options 2–23
pointer tool 2–18
Pushbutton tool
basic 4GL 2–19
Smart array 2–21
Query filtering tool
smart 2–21
Query tool 2–18
Radio-Set tool 2–19
Rectangle tool 2–18
save current position, shape as default 2–23
Selection-List tool 2–19
Slider-Control tool 2–19
SmartContainer tool 2–21
SmartDataBrowser tool 2–21
SmartDataField tool 2–21
SmartDataObject tool 2–20
SmartDataViewer tool 2–21
SmartDialog tool 2–17
SmartFilter tool 2–21
SmartFolder tool 2–21
SmartFrame tool 2–17
SmartObject tool 2–21, 8–2
SmartPanel tool 2–21
SmartToolbar tool 2–21
SmartWindow tool 2–17
Spin-Buttons tool
ActiveX 2–20
Static-Image tool 2–18
Static-Text tool 2–20
Timer tool
ActiveX 2–20
Toggle-Box tool 2–19
Tool 2–21
tools not on
basic 4GL Dialog Box 2–17
basic 4GL Window 2–17
SmartDialog 2–17
SmartFrame 2–17
SmartWindow 2–17
using 2–22
Window tool
basic 4GL 2–17
Smart 2–17

OCX (ActiveX) object 8–2

Open existing object 2–6

Organizer
Smart
example of use 1–19

Organizers
Simple SmartContainer 3–27

OUTMESSAGE SmartLinks 6–33, 6–40

## P

Page 0 (Zero)
always displayed 3–33
role of 3–33

Paged layouts
creating 3–31
creating pages 3–32
deleting pages 3–32
designing without SmartFolder 3–39
identifying pages 3–32
see all pages together 3–33
using whole windows 3–41

Paging
ADM Pages dialog
deleting objects A–6
moving objects between pages A–6
setting the Design Page A–6

setting the Startup Page A–6
changing Design Page 3–33
Design Page 3–32
Design Page, setting A–6
listing a page's objects A–5
moving objects between pages A–6
Page 0 is default page 3–33
placing objects 3–32
SmartFolder 3–34
limitations 3–35
Startup Page
setting A–6
startup page 3–34
whole-window 3–41
without SmartFolder 3–39

Partitions
configuring 4–26
JMS 6–34, 6–37
on AppServers 4–26

Pasting
Section Editor 2–39

Persistence
enabling A–21

Pick List. *See* Selection List

Placeholder
design-time A–28
SmartObject A–28

Pointer 2–17, 2–18

Preferences
default function type 2–25
double-click 2–25
layout grid 2–27
printing 2–28
Section Editor 2–28
setting 2–12, 2–23

Preprocessor names
Section Editor 2–49

Procedure settings A–19

Procedure-based object
AppServer support, enabling A–21
descriptive comments field A–20
location of ActiveX (OCX) properties A–20
location of executable A–20
persistence
specifying A–21

Procedures
files
inspecting code 2–53
running 2–55
saving 2–55
in AppBuilder API C–24
marking as modified C–44
Section Editor 2–46

progress.cst C–3
sample entry in C–18

Properties
viewing and editing 2–52

Properties Window A–21

Property sheet
SmartDataObject 4–9

Pushbutton 7–12
creating and placing 7–13
example of use 1–20, 3–39, 3–41
labeling 7–16
Smart array 7–8
example of use 1–19
tool
basic 4GL 2–19
Smart array 2–21
types supplied 7–12
close dialog or window 7–12
end application session 7–12
generic 7–12
invoke help system 7–12
move current-record pointer 7–12
view ADM page 7–12


## Q


Queries

building 4–36
establishing join criteria 4–37, 4–38
establishing search criteria 4–39
establishing sort criteria 4–40
modifying 4–36
Section Editor 2–50
setting options 4–41

Query
allowing user to filter 4–27
creating 4–34
filtering tool
Smart 2–21
tool 2–18

Query Builder
join mode 4–38
options mode 4–41
sort mode 4–40
table mode 4–36
using 4–35
where mode 4–39

Query Tuning options panel 4–42

Query-By-Example 4–27

## R

Radio Set 5–72
configuring 5–73
creating and placing 5–72
example of use 3–39
initial value 5–74
tool 2–19

Rectangle
configuring 3–24
creating and placing 3–24
example of use 1–20, 3–39
tool 2–18

Remote development
AppBuilder main window 2–16
saving files 2–55

Remote file management C–39

Reserved names C–45

ROUTER SmartLinks 6–36

Routing incoming documents 6–26

Run-time customization of user interface B–17

Run-time expressions
standard layouts B–4

## S

Save object 2–6

Section Editor 2–37
cutting, copying, pasting 2–39
database-field names 2–43
event names in 2–45
filenames/pathnames 2–52
invoking 2–37
navigating code sections 2–38
object names 2–51
preprocessor names 2–49
procedure/function calls 2–46
query definition 2–50
text-file insertion 2–42
using 2–39

Selection List 5–75
configuring 5–76
creating and placing 5–76
list of choices 5–77
tool 2–19
value of selection 5–77
value types 5–77

shared.cst C–3

Simple SmartContainers 3–27 to ??

Single-line Editor 5–67
configuring 5–68
creating and placing 5–68

Slider Control 5–79
bounds, upper and lower 5–80
configuring 5–80

creating and placing 5–79
initial value 5–80
scale markings 5–80
tool 2–19
upper and lower value limits 5–80

Smart
defined 1–11

smart.cst C–3

SmartB2BObjects 6–2 to 6–22
4GL/XML 6–2
configuring instances 6–18
creating instances 6–17
creating masters 6–16
do not use XMLMessage type 6–35
dynamic master 6–16
filename convention 6–17
instance properties
incoming 6–21
multiple producers 6–20
outgoing 6–19
inter-business messaging 6–2
selecting dynamic version 6–17
transformation 6–2

SmartBusinessObjects ?? to 4–27
and partitions 4–26
as DATA-TARGETs 4–21
bind connections to AppServers 4–26
can reduce complexity 4–17
configuration tasks 4–19
configuring 4–19, 4–25
creating and placing 4–18
filename convention 4–19
in Web applications 4–16
integration limit 4–16
whole-stream validation 4–20

SmartConsumers 6–32 to 6–40
configuring instances 6–37
creating instance 6–36
in batch mode 6–39
JMS Partitions 6–37
shutdown 6–38
SonicMQ domains 6–37
stopping when in batch mode 6–39

SmartContainer
tool 2–21

SmartContainers
design-time appearance 3–30
simple
configuring 3–29
creating 3–28
instance properties 3–31
simple non-visible 3–27

SmartDataBrowser 5–3
creating and placing 5–18
dynamic 5–3
creating and placing 5–3
display/update rules 5–5
editing master 5–16
navigation in 7–8
static 5–6
configuring 5–10
creating master 5–6
tool 2–21

SmartDataField 5–34
example of use 5–41
SmartSelect 5–36
tool 2–21

SmartDataObject 4–2, 4–9
as placeholder 4–11
configuring 4–11
connectionless 4–11
creating and placing 4–9
data validation limitation 4–13
designing 4–4
example of use 1–20
master
creating 4–4
editing 4–8
tool 2–20

SmartDataObjects
and DATA links 4–20
client-only version 4–3
dynamic
and WebClient 4–3
as generic data pumps 4–4

limitation 4–4
static
limitation 4–3
synchronizing 4–20

SmartDataViewer 5–21
adding/removing fields 5–27
configuring 5–27, 5–28, 5–29
fields layout 5–29
help string 5–33
initial value 5–33
creating master 5–22
example of use 1–19, 3–39, 5–41
state determines appearance 5–25
tool 2–21

SmartDialog
configuring 3–16
creating instance 3–4

SmartFilter 4–27
configuring 4–29
creating and placing 4–28
displaying in separate window 7–5
example of use 7–5
tool 2–21

SmartFolder
example of use 3–34
limitations 3–35
paging with 3–34
tool 2–21

SmartFrame
configuring 3–19
creating instance 3–4

SmartInfo dialog A–25

SmartLink types
DATA 4–20
INMESSAGE 6–36
OUTMESSAGE 6–33, 6–40
ROUTER 6–36

SmartLinks
adding, removing, modifying A–22
API routine to add, remove C–25
example of use 1–19, 1–20
list of A–22

SmartLinks editor A–22

SmartObject
creating new 2–5
placeholder, creating as A–28
property sheet A–26
tool 2–21, 8–2

SmartObject template 8–2

SmartObjects
advantages in using 1–12
creating under program control C–40 to C–42
deleting under program control C–42
listed 1–8
menu-button menu 1–13
multiple layouts B–21
non-visible
example of 1–13
synchronizing 1–15
ventilator-button menu 1–13
visible vs non-visible 1–12
what "Smart" means 1–11

SmartPanel 7–8
configuring 7–9
creating and placing 7–9
example of use 1–19, 3–34, 3–39
tool 2–21
types supplied 7–8

SmartProducers 6–32 to 6–40
JMS Partitions 6–34
message persistence 6–35
message types 6–35
SonicMQ domains 6–34

SmartReceivers 6–22 to 6–25

SmartRouters 6–26 to 6–31
role in message-handling 6–28

SmartSelect 5–36
configuring 5–38
creating and placing 5–37

example of use 5–41
filtering the data 5–38

SmartSenders 6–22 to 6–25
creating instances 6–23
filename conventions 6–24

SmartToolbar 7–2
configuring 7–4
creating and placing 7–3
creating new master 7–5
limitations 7–2
tool 2–21

SmartWindow
configuring 3–6
creating instance 3–4
example of use 1–19, 3–34

Source code
examining 2–9

Spin Buttons
tool
ActiveX 2–20

Spin Control (ActiveX) 5–82
configuring 5–83
creating and placing 5–83

Starting AppBuilder 2–2

Static Image object 8–10

Static Text
example of use 1–20
style selector and editor A–34
tool 2–20

Static Text object 8–14

Status bar
AppBuilder main window 2–4

Stop button
AppBuilder main window 2–55

String literals
specifying space allocation A–28

String-Attributes dialog A–28

Subscriptions
persistence 6–35
preserving across sessions 6–38

Synchronizing
with master layout B–10

Syntax notation xix

## T

Templates
adding to New-object list C–8

Temp-tables
defining and maintaining A–31

Text
tool 2–20

Text style
choosing A–34
defining A–35

Timer
tool
ActiveX 2–20

Timer object
interrupt-driven 8–15

Toggle Box 5–85
configuring 5–86
creating and placing 5–85
initial value 5–86
tool 2–19

Tool
ActiveX (OCX) object 2–20
Check-Box. *See* Toggle Box
Combo-Box
ActiveX 2–20
basic 4GL 2–19
Data-Browser
basic 4GL 2–18
smart 2–21
Data-pump
Smart 2–20
Data-Viewer

basic 4GL 2–18
Smart 2–21
Editor, multi-line 2–19
Editor, single-line 2–20
Fill-in 2–20
Frame
basic 4GL 2–18
Image 2–18
pointer 2–18
Pushbutton
basic 4GL 2–19
smart array 2–21
Query 2–18
Query-filtering
smart 2–21
Radio-Set 2–19
Rectangle 2–18
Selection-List 2–19
Slider-Control 2–19
SmartContainer 2–21
SmartDataBrowser 2–21
SmartDataField 2–21
SmartDataObject 2–20
SmartDataViewer 2–21
SmartFilter 2–21
SmartFolder 2–21
SmartObject 2–21, 8–2
SmartPanel 2–21
SmartToolbar 2–21
Spin-Buttons
ActiveX 2–20
Text 2–20
Timer
ActiveX 2–20
Toggle-Box 2–19

Tool bar
AppBuilder main window 2–16

Tool icons
adding to Objects Palette C–4

Tools
cancelling selection 2–22
latching and unlatching 2–22
list of 2–18
SmartB2BObject 6–16
SmartBusinessObject 4–18
SmartConsumer 6–36
SmartContainer 3–30
SmartProducer 6–33
SmartSender 6–23, 6–24
using 2–22
XML Schema Mapper PRO*Tool 6–11

Traversal path
editing A–30

Tree view 2–36

Typographical conventions xviii


## U


Undoing changes
Section Editor 2–43

Usability
avoid cross-connections 5–86
be sparing with colors A–10
be sparing with font changes A–36
fonts as corporate-identity signifier A–34
neatness 3–33
role of Pushbutton 7–12
role of Toggle Boxes vs Radio Set 5–85
use fonts sparingly A–34
violating expectations causes confusion 3–40


## W


Web browser C–39

Web preferences C–39

web.cst C–3

WebSpeed
AppBuilder for 2–2

Widget
open Property Sheet for 2–10

Widgets
*See also* Objects
adding to custom object file C–9
capturing changes made by user 1–15

defined 1–13
reading & writing 1–14
showing changes made by program 1–14

Window
basic 4GL
configuring 3–22
creating new 2–5
opening Property Sheet for 3–6
Smart
configuring 3–6
creating new 2–5

Workspaces 2–35
allowed objects in A–8
for non-visible objects 2–36
saving 2–55
testing 2–55


## X


XFTRs. *See* Extended features

.xmc file type 6–11

XML
and HTML 6–2
and W3C 6–2
defined 6–2
schema and mapping files 6–3
standards 6–2
standards committee 6–3

XML mapping
consumer map file extension 6–11
creating a map file 6–11
creating a record 6–8
deleting a record 6–8
editor 6–9
editor PRO*Tool 6–9
external references 6–31
finding a record 6–8
illustration 6–4, 6–5
inbound/outbound files different 6–9
internal references 6–31
iteration 6–7
mapping schema nodes 6–13
node to data column 6–6
with conversion 6–6
node to function 6–6
node to procedure 6–6
node to whole data object 6–7
options 6–5
producer map file extension 6–11
schema file extension 6–11
selecting resources 6–12
starting editor PRO*Tool 6–11

.xmp file type 6–11

.xsd file type 6–11