

Progress on the Web

© 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.

The implementation of the MD5 Message-Digest Algorithm used by Progress Software Corporation in certain of its products is derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm.

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

.

May 2001



Product Code: 4611
Item Number: 81070;9.1C

Contents

Preface . . . . . **ix**
Purpose ix
Audience ix
Organization of This Manual x
Typographical Conventions xi
Syntax Notation xii
Example Procedures xvi
Progress Messages xix
Other Useful Documentation xxi
Getting Started xxi
Development Tools xxii
Reporting Tools xxiii
4GL xxiv
Database xxv
DataServers xxv
SQL-89/Open Access xxvi
SQL-92 xxvi
Deployment xxvii
WebSpeed xxvii
Reference xxviii
SQL-92 Reference xxviii

1. Overview **1–1**
1.1 Differences Between WebClient and the Standard 4GL Client 1–2
1.2 Common WebClient Scenarios 1–3
1.2.1 ASP Scenario 1–3
1.2.2 IT Department Scenario 1–4
1.3 WebClient Files and Software 1–5
1.4 Personnel WebClient Involves 1–6

1.5 Hardware and Software Requirements for the End User 1–6
1.6 WebClient Architecture 1–7
1.6.1 WebClient Application Assembler 1–7
1.6.2 Using IntelliStream Technology 1–8
1.6.3 Using Non-IntelliStream Technology 1–11
1.6.4 Using IntelliStream and Non-IntelliStream Technologies Together 1–11
1.6.5 Running the Application 1–12
1.6.6 Single Sign-on 1–13
1.6.7 Server Sharing 1–13
1.7 Steps for Deploying an Application Using WebClient 1–14
1.8 Sample Application 1–14

2. Designing the Application 2–1
2.1 Segregating User-interface Logic and Business Logic 2–2
2.2 Accessing Databases Only Through an AppServer 2–2
2.3 Modularizing Code by Function 2–2

3. Designing Your End User's Experience 3–1
3.1 WebClient Installation 3–2
3.2 WebClient Application Installation 3–3
3.2.1 IntelliStream 3–4
3.2.2 External Installer 3–5
3.3 Updates for Your Application 3–5
3.4 Uninstalling Your Application 3–6
3.5 Selecting Deployment Servers 3–6
3.6 Starting WebClient Applications 3–6
3.7 Providing Documentation 3–7

4. Designing the Deployment Configuration 4–1
4.1 Selecting a Server to Host the Application Configuration File 4–2
4.2 Selecting a Server to Host the Codebase 4–3
4.3 Selecting a Server to Host a Non-IntelliStream Application Install 4–4
4.4 Determining the Best Location for the Codebase 4–4
4.4.1 Hosting the Codebase on an AppServer. 4–4
4.4.2 Hosting the Codebase on an Internet-based Server. 4–4
4.5 Selecting a Communication Protocol for Configuration File Downloads 4–5
4.6 Selecting a Communication Protocol for Codebase Downloads 4–6

5. Designing Security 5–1
5.1 Digitally Signing Files 5–2
5.1.1 The Development of Digital Signatures 5–2
5.1.2 How WebClient Uses Digital Signatures 5–5
5.2 Protecting Servers with User IDs and Passwords 5–10
5.2.1 Using Authentication 5–10
5.2.2 The Dangers of Embedding User IDs and Passwords in URLs 5–10
5.2.3 Prompting the End User for Authentication Information 5–11
5.3 Single Sign-on and Security Caching 5–13
5.3.1 Authentication Dialogs 5–14
5.4 Choosing a Secure Communication Protocol for Configuration File and Codebase Downloads 5–15

6. Developing the Application 6–1
6.1 Using the CODEBASE-LOCATOR Handle and its Attributes 6–2
6.1.1 Basic Attributes 6–2
6.1.2 Security-cache Attributes 6–4
6.1.3 Additional Attribute 6–6
6.2 Managing AppServer Connections 6–7
6.3 Implementing Single Sign-on 6–8
6.4 Using WebClient's Authentication Dialogs in Your Application 6–9
6.4.1 Displaying the AIA Authentication Dialog. 6–10
6.4.2 Displaying the AppServer Authentication Dialog 6–10
6.4.3 Displaying the AIA and AppServer Authentication Dialog 6–11
6.5 Using URLs in the PROPATH to Find and Download Image Files Over the Web 6–12
6.5.1 Using the SEARCH() Function. 6–12
6.5.2 Using the LOAD–IMAGE() Method 6–13
6.5.3 Handling Errors 6–14

7. Deploying an Application 7–1
7.1 Defining Your Application Using the WebClient Application Assembler 7–2
7.1.1 IntelliStream 7–3
7.1.2 External Installer 7–5
7.1.3 Using a Combination of IntelliStream and an External Installer 7–10
7.2 Running the WebClient Application Assembler 7–12
7.2.1 General Tab 7–15
7.2.2 Component Tab 7–16
7.2.3 Options Tab 7–17
7.2.4 WebClient Tab 7–19
7.3 Files Generated by the WebClient Application Assembler 7–19

7.4 Hosting the Application on a Server 7–21
7.4.1 Hosting the Configuration File 7–22
7.4.2 Hosting Your Application Component Files 7–23
7.4.3 Hosting Your External Installer 7–23
7.4.4 Hosting on a Web Server 7–24
7.5 Installing WebClient 7–27
7.5.1 Accessing the WebClient Installation Files 7–28
7.5.2 Customizing the WebClient Installation 7–28
7.6 Bootstrapping a WebClient Application 7–30
7.7 WebClient Application Updates 7–37

8. Your End User's Experience 8–1
8.1 Preparing Documentation for Your End Users 8–2
8.2 WebClient Application Manager 8–4
8.2.1 Using the WebClient Application Manager 8–4
8.2.2 Modes of Execution 8–5
8.2.3 Main Window 8–5
8.2.4 Edit Dialog Box 8–7

A. Deploying the Sample Application With IntelliStream A–1
A.1 Preparing the Web Server A–3
A.1.1 Setting Up the Java Servlet Engine and the AppServer Internet Adapter A–3
A.1.2 Setting Up MIME Types for the Application Configuration File A–3
A.1.3 Setting Up the WebClient Install and Bootstrap Mechanism . . A–3
A.2 Setting Up the Application on the AppServer Machine A–5
A.3 Setting Up the Application to Launch from the Web Server A–9
A.4 Installing and Running the Application on the Client Machine A–10

B. Deploying the Sample Application Without IntelliStream B–1
B.1 Preparing the Web Server B–2
B.1.1 Setting Up the Java Servlet Engine and the AppServer Internet Adapter B–2
B.1.2 Setting Up MIME Types for the Application Configuration File B–2
B.2 Setting Up the Application on the AppServer Machine B–3
B.3 Setting Up the Application on the Web Server Machine B–5
B.4 Installing and Running the Application on the Client Machine B–8

Glossary Glossary–1

Index Index–1

Tables

Table 1–1:	Comparing WebClient to the Standard 4GL Client	1–2
Table 1–2:	Steps for Using WebClient	1–14
Table 4–1:	Communications Protocols for Configuration File Downloads	4–5
Table 4–2:	Communication Protocols for Codebase Downloads	4–6
Table 6–1:	Basic Attributes of the CODEBASE-LOCATOR Handle	6–2
Table 6–2:	Security-cache Attributes of the CODEBASE-LOCATOR Handle	6–4
Table 6–3:	Additional Attribute of the CODEBASE-LOCATOR Handle	6–6
Table 7–1:	Registry Entries Made by WebClient	7–5
Table 7–2:	Vendor and Application Name Registry Key Values	7–8
Table 7–3:	Configuration File Registry Key Entries	7–9
Table 7–4:	WebClient Applications Registry Key Value	7–10
Table 7–5:	Component Registry Key Entry	7–11

Figures

Figure 4–1:	Describing the Configuration File Server to WebClient	4–2
Figure 4–2:	Describing the Codebase Server to WebClient	4–3
Figure 5–1:	Defining an Application as Signed in the Application Assembler's Generate Window	5–7
Figure 5–2:	Asking WebClient to Prompt for Configuration File Server Authentication Information	5–11
Figure 5–3:	Configuration File Server Authentication Information Dialog	5–12
Figure 5–4:	Asking WebClient to Prompt for Codebase Server Authentication Information	5–12
Figure 5–5:	Codebase Server Authentication Information Dialog	5–13
Figure 7–1:	WebClient Application Assembler	7–13
Figure 7–2:	WebClient Application Assembler General Tab	7–15
Figure 7–3:	Configuration File Locator Definition Dialog Box	7–15
Figure 7–4:	WebClient Application Assembler Component Tab	7–16
Figure 7–5:	Codebase Locator Definition Dialog Box	7–16
Figure 7–6:	Component Definition Dialog Box	7–17
Figure 7–7:	WebClient Application Assembler Options Tab	7–17
Figure 7–8:	WebClient Application Assembler WebClient Tab	7–19
Figure 7–9:	WebClient Application Assembler Generate Dialog Box	7–20
Figure 7–10:	Bootstrapping Process	7–33
Figure 8–1:	Sample Browser File Download Alert Box	8–2
Figure 8–2:	WebClient Tab of the WebClient Application Manager	8–6
Figure 8–3:	Applications Tab of the WebClient Application Manager	8–7
Figure 8–4:	Application Manager Edit Dialog Box	8–8
Figure 8–5:	General Tab Displaying Information About a 9.1C Application	8–9
Figure 8–6:	General Tab Displaying Information About a 9.1B Application	8–10
Figure 8–7:	Log Tab of the WebClient Application Manager	8–11
Figure 8–8:	Security Tab	8–12
Figure 8–9:	Component Tab	8–13

Preface

Purpose

This guide describes the architecture and options for enabling a standard Progress application for deployment over the World Wide Web using the Progress WebClient. It provides guidance on WebClient implementations that are typical in the following application scenarios:

- The Application Service Provider (ASP) deploying a rental application over the Internet
- An Information Technology (IT) department deploying corporate applications over an intranet

The overall goal is to provide a single set of procedures for implementing all types of WebClient applications, while also making appropriate recommendations for these typical scenarios.

Audience

The audience for this guide includes:

- Professional developers who use Progress software products to build, deploy, and manage applications across the Internet, intra- and extra-nets, multi-tier, client/server, and host/terminal computing environments
- Any service provider who deploys a Progress WebClient application developed by someone else

Organization of This Manual

This guide is organized in the following manner:

Chapter 1, “Overview”

Describes the architecture, requirements, and typical application scenarios for using the Progress WebClient to implement Progress on the Web, including an overview of typical implementation steps.

Chapter 2, “Designing the Application”

Describes the basic requirements and options available to prepare a standard Progress application for deployment over the Web using the Progress WebClient.

Chapter 3, “Designing Your End User’s Experience”

Describes the available delivery options for deploying a Progress WebClient application, including recommended combinations.

Chapter 4, “Designing the Deployment Configuration”

Describes and compares the options for configuring the deployment environment for a Progress WebClient application, including information on supported servers and server software.

Chapter 5, “Designing Security”

Provides an overview of security concepts and describes how the Progress WebClient architecture supports security for typical application scenarios.

Chapter 6, “Developing the Application”

Describes how to use the Progress 4GL to implement WebClient application design decisions and requirements.

Chapter 7, “Deploying an Application”

Provides a guide to the tasks you must complete and the tools you must use to implement your delivery design and deploy a WebClient application in your deployment environment.

Chapter 8, "Your End User's Experience"

Provides information that your application users might need to know to install and run your application, and to help you troubleshoot any WebClient applications running on their system.

Appendix A, "Deploying the Sample Application With IntelliStream"

Describes how to deploy the SportsPro sample application using Progress WebClient with IntelliStream™.

Appendix B, "Deploying the Sample Application Without IntelliStream"

Describes how to deploy the SportsPro sample application using Progress WebClient without IntelliStream.

"Glossary"

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 ... ] }
       }
```

Example Procedures

This manual provides numerous example procedures that illustrate syntax and concepts. Examples use the following conventions:

- They appear in boxes with borders.
- If they are available online, the name of the procedure appears above the left corner of the box and starts with a prefix associated with the manual that references it, as follows:
  - `e-` — *Progress External Program Interfaces*, for example, `e-ddeex1.p`
  - `lt-` — *Progress Language Tutorial*, for example, `lt-05-s3.p`
  - `p-` — *Progress Programming Handbook*, for example, `p-br01.p`
  - `r-` — *Progress Language Reference*, for example, `r-dynbut.p`

  If the name does not start with a listed prefix, the procedure is not available online.
- If they are not available online, they compile as shown, but might not execute for lack of completeness.

Accessing Files in Procedure Libraries

Documentation examples are stored in procedure libraries, `prodoc.pl` and `prohelp.pl`, in the `src` directory where Progress is installed.

You must first create all subdirectories required by a library before attempting to extract files from the library. You can see what directories and subdirectories a library needs by using the PROLIB –list command to view the contents of the library. See the *Progress Client Deployment Guide* for more details on the PROLIB utility.

Extracting source files from a procedure library involves running PROENV to set up your Progress environment, creating the directory structure for the files you want to extract, and running PROLIB.

1 ♦ From the Control Panel or the Progress Program Group, double-click the Proenv icon.

2 ♦ The Proenv Window appears, with the proenv prompt.

Running Proenv sets the DLC environment variable to the directory where you installed Progress (by default, `C:\Program Files\Progress`). Proenv also adds the DLC environment variable to your PATH environment variable and adds the bin directory `(PATH=%DLC%;%DLC%\bin;%PATH%)`.

3 ♦ Enter the following command at the proenv prompt to create the `prodoc` directory in your Progress working directory (by default, `C:\Progress\Wrk`):

```
MKDIR prodoc
```

4 ♦ Create the `langref` directory under `prodoc`:

```
MKDIR prodoc\langref
```

5 ♦ To extract all examples in a procedure library directory, run the PROLIB utility. Note that you must use double quotes because “Program Files” contains an embedded space:

```
PROLIB "%DLC%\src\prodoc.pl" -extract prodoc\langref\*.*
```

PROLIB extracts all examples into `prodoc\langref`.

To extract one example, run PROLIB and specify the file that you want to extract as it is stored in the procedure library:

```
PROLIB "%DLC%\src\prodoc.pl" -extract prodoc/langref/r-syshlp.p
```

PROLIB extracts `r-syshlp.p` into `prodoc\langref`.

Extracting Source Files from Procedure Libraries on UNIX Platforms

To extract p-wrk1.p from prodoc.pl, a procedure library, follow these steps at the UNIX system prompt:

1 ♦ Run the PROENV utility:

```
install-dir/dlc/bin/proenv
```

Running proenv sets the DLC environment variable to the directory where you installed Progress (by default, /usr/dlc). The proenv utility also adds the DLC environment variable to your PATH environment variable and adds the bin directory (PATH=%DLC%;%DLC%\bin;%PATH%).

2 ♦ At the proenv prompt, create the prodoc directory in your Progress working directory:

```
mkdir prodoc
```

3 ♦ Create the proghand directory under prodoc:

```
mkdir prodoc/proghand
```

4 ♦ To extract all examples in a procedure library directory, run the PROLIB utility:

```
prolib $DLC/src/prodoc.pl -extract prodoc/proghand/*.*
```

PROLIB extracts all examples into prodoc\langref.

To extract one example, run PROLIB and specify the file that you want to extract as it is stored in the procedure library:

```
prolib $DLC/src/prodoc.pl -extract prodoc/proghand/p-wrk-1.p
```

PROLIB extracts p-wrk-1.p into prodoc/proghand.

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. On the Windows platform, 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**).

On the UNIX platform, you can use the Progress PRO command to start a single-user mode character Progress client session and view a brief description of a message by providing its number. Follow these steps:

1 ♦ Start the Progress Procedure Editor:

```
install-dir/dlc/bin/pro
```

2 ♦ Press **F3** to access the menu bar, then choose Help→ Messages.

3 ♦ Type the message number, and press **ENTER**. Details about that message number appear.

4 ♦ Press **F4** to close the message, press **F3** to access the Procedure Editor menu, and choose File→ Exit.

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 AppBuilder Developer's Guide (Windows only)

A programmer's guide to using the Progress AppBuilder visual layout editor. AppBuilder is a Rapid Application Development (RAD) tool that can significantly reduce the time and effort required to create Progress applications.

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 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

Overview

WebClient lets you, the application deployer, take a Web-enabled Progress application, deploy and configure its back-end components as desired, package its front-end components, and host these packages on a server. Similarly, you can package updates to the application front end and place them on a server as well. The end user, typically using a Web browser, can request to run the application. In response, WebClient downloads the initial front-end package, installs and runs the application, downloads update packages, and installs the updates — all with a high degree of transparency.

This chapter discusses:

- Differences between WebClient and the standard Progress 4GL Client
- Typical WebClient scenarios
- Personnel WebClient involves
- WebClient files and software
- Hardware and software requirements for the end user
- WebClient architecture
- Steps for using WebClient
- Sample application

1.1 Differences Between WebClient and the Standard 4GL Client

Table 1–1 compares WebClient and the standard Progress 4GL client.

Table 1–1: Comparing WebClient to the Standard 4GL Client

Feature	WebClient	Standard 4GL Client
Execution	Contains no compiler; runs as a reduced-size executable.	Can compile source and runs as a complete 4GL client executable.
Data Access	Supports TEMP–TABLE access, but does not support direct Progress Database or DataServer access; relies on a connected AppServer to provide indirect access to all databases and DataServers.	Supports direct access to all Progress data objects, as well as indirect access to databases and DataServers using a connected AppServer.
Customization	Cannot be customized using PROBUILD.	Can be customized using PROBUILD.
User Interface	Supports only the graphical user interface (GUI) and batch modes in Windows.	Supports all 4GL user-interface modes.
Method of Launch	Is typically launched by accessing a Web page or a shortcut that runs the WebClient Initializer, which performs certain housekeeping tasks and automatically executes WebClient on the end user's behalf. The 4GL procedures that comprise the application do not have to be installed on the end user's machine prior to run time.	Is typically launched by executing a script or shortcut you provide that executes the 4GL client on the end user's behalf. The 4GL procedures that comprise the application must be installed on the end user's machine prior to run time.

1.2 Common WebClient Scenarios

WebClient is specifically designed for:

- Application service providers (ASPs) renting out applications over an extranet
- Corporate Information Technology (IT) departments deploying applications internally over a corporate intranet

1.2.1 ASP Scenario

As an ASP, you rent applications to end users, who install and run the applications over the Web or an extranet. Your end users might perform tasks similar to the following:

1. Identifying the application

   End users identify the application on your Web site as a likely business solution through a Web search.

2. Evaluating the application

   End users sample the application through a live demonstration, installed and executed directly on their systems over the Web. You make this option available by providing the application in a demo mode, complete with online documentation and sample data.

3. Subscribing to the application

   You allow end users access to a measured level of application functionality based on a price-driven subscription agreement. You maintain and administer the application's back-end (its servers, databases, etc.)

4. Downloading, installing, and launching the application

   Your Web site grants end users access to the application, including the purchased level of functionality controlled by a version-based license. This allows them to download and install the application directly from a server, complete with all supporting Progress components. End users then run the application, which access the business logic on your back-end server.

1.2.2 IT Department Scenario

As an IT department, you install and run your application entirely over a corporate intranet. You or your end users might perform tasks similar to the following:

1. Identifying the application

   You search the Web to identify an application to deploy using WebClient.

2. Evaluating the application

   Your end users sample the application through a live demonstration, installed and executed directly on their systems over the Internet.

3. Purchasing the application

   You purchase the application.

4. Deploying the application

   You deploy the application on a local machine.

5. Downloading, installing, and launching the application

   Your end users access the local machine to download, install, run, and update the application, as in the ASP model.

1.3 WebClient Files and Software

The following files and software support your WebClient application deployment:

- **Application** — The application to be deployed using WebClient.
- **WebClient Executable (`prowc.exe`)** — The WebClient main executable, this program is responsible for running the application on the user's machine.
- **WebClient Initializer (`prowcini.exe`)** — Installed with WebClient, the Initializer is responsible for coordinating the installation and execution of a WebClient application.
- **WebClient Application Assembler (`prowcappmgr.exe`)** — A utility used to define the application to WebClient and to package the front-end of the application for downloading.
- **WebClient Application Manager (`prowcam.exe`)** — A utility for customizing and troubleshooting your WebClient applications on the end user's machine. It is typically used when the end user talks with your technical support staff.
- **Application Configuration (`.prowcapp`) file** — The configuration file created by the Application Assembler and downloaded to the end user's machine at the initial download and each time the end user starts the application. This file provides WebClient with all the information it needs to know about the application.
- **WebClient Project (`.wcp`) file** — The file that contains the WebClient application install definition for the latest version of an application.

  In the Windows Explorer, if you right-click on a project file, the Application Assembler starts.
- **Microsoft Cabinet (.cab) files** — Files into which the procedure files, image files, etc., comprising the front-end of the application are placed to prepare them for downloading to the end user's machine.
- **Microsoft Authenticode™ Technology** — The software that creates and validates digital signatures.
- **Test-certificate generator (`maketestcert.bat`)** — A batch file that simplifies creating test public-key certificates used for testing digitally signed applications.

1.4 Personnel WebClient Involves

Using WebClient typically involves the following people:

- **Application deployer** — The person who reads the WebClient documentation and uses WebClient to deploy the application. This person might also serve as the application designer, application developer, and system administrator.
- **End user** — The person who uses the application set up by the application deployer.
- **Out-hoster** — A third party who hosts the application on their Web site. An out-hoster might be used when the Web site of the application deployer's company cannot host the application.

  NOTE: Since out-hosters typically lack knowledge of WebClient and Progress, they must work closely with the application deployer.

1.5 Hardware and Software Requirements for the End User

To use WebClient, the end user **must**:

- Run one of these operating systems on their machine:
  - Windows 95
  - Windows 98
  - Windows ME
  - Windows NT 4.0
  - Windows 2000
- Have sufficient disk space on their system.
- Use an account with Administrator privileges.
- Shut down or stop all nonessential programs and processes.
- Use Netscape 4.x (or later) Microsoft Internet Explorer 4.x (or later) if they run the installer over the Web.

  NOTE: These browsers support ActiveX control or plug-in technology to run a Web installation.

1.6 WebClient Architecture

This section discusses the following aspects of WebClient architecture:

- WebClient Application Assembler
- Using IntelliStream™ technology
- Using non-IntelliStream technology
- Using IntelliStream and non-IntelliStream technology together
- Running the application
- Digital signatures
- Single sign-on
- Server sharing

1.6.1 WebClient Application Assembler

The Application Assembler lets you easily create the files you need to deploy and update WebClient applications. After you complete your application development and compile your application code, you use the Application Assembler to generate files for deploying and updating the front end of your applications. The Assembler assumes the following responsibilities pertaining to deploying and updating of your WebClient application:

- Creating and updating a WebClient project (`.wcp`) file
- Generating an application configuration (`.prowcapp`) file for deploying the application, and packaging it into a cabinet (`.prowcapc`) file.
- Identifying components and generating cabinet (`.cab`) files that contain your application files and updates to your application files
- Providing a set of features to support your application's installation

For more information, see Chapter 7, "Deploying an Application."

Use the Application Assembler, you can implement installations and updates that use IntelliStream™ technology, non-IntelliStream technology, or a combination of the two, as the following sections explain.

1.6.2 Using IntelliStream Technology

With IntelliStream technology, the application deployer can include in the initial download only those application files needed to start the application. Then, as the end user runs other parts of the application, WebClient automatically downloads and installs the additional files as needed.

IntelliStream also works every time the end user starts an application by checking for updated application components, downloading them, and installing them — all before the application starts.

When an application component reaches the end user's machine, either as part of the initial download or as part of an update, the files within the component are cached.

For more information on IntelliStream, see Chapter 7, "Deploying an Application."

Advantages of Using IntelliStream

Using IntelliStream provides the following advantages:

- The initial download and subsequent upgrades do not requiring special installation procedures or update procedures.
- The initial download need not contain all components of the application. Thus, the end user's machine is not cluttered with application components that are never used.
- When the application deployer updates the application, WebClient on the end user's machine automatically downloads only the application-component elements that have changed, making downloads faster.
- Application components (for the initial download and for updates) can be hosted on an AppServer as well as on a Web server.

Codebase Locator

As part of defining the application to WebClient, the application deployer, using the Application Assembler, describes the server that contains the application components (for the initial download and for updates) to be downloaded. This description is called the codebase locator, which is used by WebClient to access this server, called the codebase server.

For more information, see Chapter 4, "Designing the Deployment Configuration."

Components and Download Mode

To download the files of the client portion of the application, WebClient places them in Microsoft cabinet files. Typically, a single cabinet file might contain several procedure (`.r`) or procedure library (`.pl`) files, image files, etc. A single cabinet file comprises an application component.

The application deployer assigns each application component one of the following download modes:

- `At Startup` — The component is included in the initial download.
- `As Needed` — The component is automatically downloaded by WebClient when the end user uses the part of the application that references it.
- `Ask User at Startup` — At the time of the initial download, WebClient asks the end user which as-requested components they want. WebClient then downloads them (along with the at-startup components).

For more information, see Chapter 7, “Deploying an Application.”

Component Update Files

When the application deployer defines a new version of an application to the Application Assembler, the Application Assembler automatically creates a *component update file* for each component whose new version is different from a previous version. (A new version of a component is different from a previous version when any application file residing in the component has changed.) Each component update file represents the delta (difference) between the new version of a component and the previous version of the same component.

When WebClient on the end user’s machine determines that the end user’s version of the application needs updating, WebClient downloads the component update files that update the version on the end user’s machine to the newest version.

For more information, see Chapter 7, “Deploying an Application.”

Determining if a Resource was Modified

When the application deployer generates a new version of an application using the Application Assembler, the Application Assembler determines if an r-code resource has changed since the last version by comparing the MD-5 value for the r-code with the MD-5 value from the previous version, which the Application Assembler stores in a version-information file. (MD-5 is an algorithm that reads a file and produces a short mathematical value. If a file changes even slightly, MD-5 produces a different mathematical value. By comparing the MD-5 values of two files, you can quickly determine if they are identical or different.) For more information on the version-information file, see Chapter 7, "Deploying an Application."

When compiling procedures, be sure to specify the Generate MD-5 option. If you do not, MD-5 values are not inserted into r-code resources, forcing the Application Assembler to check for changes much more coarsely (by comparing time stamps, which might cause unnecessary downloads.

Obsolete Versions

When a new version of the application is defined, certain previous versions can be defined as *obsolete*. For example, you might define as obsolete a version you no longer support that all end users have upgraded from. When WebClient generates a new version, it does not produce component update files for obsolete versions.

For more information, see Chapter 7, "Deploying an Application."

System Tasks and 4GL Routine Facilities

The IntelliStream System Tasks facility (in the Application Assembler) can perform an array of built-in installation tasks, such as creating desktop shortcuts and registering system files. In addition to the System Tasks facility, WebClient can invoke a 4GL routine you supply to perform additional installation tasks. For more information on these facilities, see Chapter 7, "Deploying an Application."

Digital Signatures

WebClient lets the application deployer digitally sign each file to be downloaded. Then, when the end user downloads a file, WebClient on the end user's machine validates the digital signature.

For complete information on digital signatures in general and how WebClient uses them in particular, see Chapter 5, "Designing Security."

1.6.3 Using Non-IntelliStream Technology

The non-IntelliStream initial download (which includes WebClient and the application) can use Web-based or CDROM-based *external installer* technology — that is, install programs, such as InstallShield and similar technologies, that can install and uninstall applications.

The CDROM option is for customers who have slow communication lines. But the focus of WebClient is to deploy to end users over the Web.

The WebClient distribution includes an installation image for installing WebClient on the end user's machine.

To prepare the application for the initial download, the application deployer prepares an installation procedure using InstallShield One-Click Install technology or any other technology executable from a Web browser.

You can customize the WebClient installation so that it automatically starts the application installation.

To start the initial download using a Web browser, the end user goes to the application deployer's Web site and clicks on a button. This causes WebClient and the application to be downloaded and installed on the end user's machine.

Non-IntelliStream updates can use InstallShield One-Click Install technology or any other web-enabled install technology. The efficiency of this update depends on your installer technology.

1.6.4 Using IntelliStream and Non-IntelliStream Technologies Together

For the initial download and for updates, the application deployer can use the IntelliStream technique, an external installer, or a combination of the two.

The application deployer might use IntelliStream for initial downloads and updates:

- To minimize the time required for downloads
- To avoid having to write an InstallShield script, which can be complicated
- To avoid having to buy the InstallShield® product

The application deployer might use an external installer for initial downloads and updates if InstallShield is already being used, if minimizing downloading time is not critical, or if there are specific installation requirements that IntelliStream cannot handle.

The application deployer might combine the two techniques. For example, an initial download might use an external installer, while updates might use IntelliStream, because it minimizes the number of files that need to be downloaded.

1.6.5 Running the Application

Once WebClient and the application are installed, the end user starts the application in one of the following ways:

- Going to the application deployer's Web site and clicking on a button
- Clicking on a shortcut that was set up by the installation procedure and that resides on the desktop or in a Program Folder
- Going to a Web browser and entering the URL of a bootstrap file on the application deployer's server.

Each time the end user starts the application, WebClient on the end user's machine compares the local version of the application to the latest version on the application deployer's Web site. If the versions differ, WebClient on the end user's machine updates the local version to match the latest version.

Configuration File Locator

When the application deployer defines the application to WebClient using the Application Assembler, the Application Assembler writes this information to an application configuration file, which is downloaded as part of the initial download and each time the end user starts the application.

As part of defining the application, the application deployer, using the Application Assembler, describes the server from which the application configuration file is downloaded. This description is called the configuration file locator and is used by WebClient to access the application configuration file on the configuration file server.

For more information, see Chapter 4, "Designing the Deployment Configuration."

Acceptable-to-Run Versions

When a new version of the application is defined, a previous version that still runs acceptably can be defined as *acceptable-to-run*. For example, an acceptable-to-run version might contain a cosmetic bug that the most recent version fixes.

When the end user starts the application, WebClient on the end user's machine downloads the latest application configuration file and checks if any updates are needed. At that point, WebClient might discover that the end user's version is defined (in the application configuration file) as acceptable-to-run. If this occurs, the end user is asked if they want to download the newer version. If they do, WebClient downloads the updates and applies them.

For more information on acceptable-to-run versions, see Chapter 7, "Deploying an Application."

1.6.6 Single Sign-on

Progress Software Corporation recommends that each server the application accesses be protected with a user ID and password. To simplify access to servers, multiple servers might be protected by the same user ID and password. To facilitate single sign-on for the end user, which eliminates the need for the end user enter the same user ID and password multiple times, WebClient caches authentication information, such as user IDs and passwords, that the end user enters, and provides access to this cache to the application.

If the end user chooses, WebClient maintains the authentication cache across end user sessions. This is called *persistent caching*. Persistent caching can be disabled by the application deployer, and the persistent cache can be cleared by the end user.

For more information on single sign-on, see the "Implementing Single Sign-on" section of Chapter 6, "Developing the Application."

1.6.7 Server Sharing

A single server might be shared by WebClient and the application. For example, an AppServer that contains application components downloaded by WebClient might also contain business logic accessed by the same application. When WebClient and the application access the same server, they can share a connection.

For more information on server sharing, see the "Managing AppServer Connections" section of Chapter 6, "Developing the Application."

1.7 Steps for Deploying an Application Using WebClient

Table 1–2 describes the steps for deploying an application using WebClient.

Table 1–2: Steps for Using WebClient

Step	For Information, See...
1 ♦ Read the documentation.	*Progress on the Web*
2 ♦ Deploy the sample application in-house.	Appendix A, "Deploying the Sample Application With IntelliStream" Appendix B, "Deploying the Sample Application Without IntelliStream"
3 ♦ Redesign an existing application or design a new application.	Chapter 2, "Designing the Application"
4 ♦ Design the mechanism for the initial install and the subsequent updates.	Chapter 3, "Designing Your End User's Experience"
5 ♦ Design the deployment configuration.	Chapter 4, "Designing the Deployment Configuration"
6 ♦ Design security into the application.	Chapter 5, "Designing Security"
7 ♦ Modify the code of an existing application or code a new application.	Chapter 6, "Developing the Application"
8 ♦ Deploy the application.	Chapter 7, "Deploying an Application"
9 ♦ Prepare end-user documentation.	Chapter 8, "Your End User's Experience"

1.8 Sample Application

For information on running the sample application, see Appendix A, "Deploying the Sample Application With IntelliStream" and Appendix B, "Deploying the Sample Application Without IntelliStream."

2

Designing the Application

Progress WebClient takes the standard Progress 4GL client and extends it to the World Wide Web and beyond, letting you deploy applications on the Internet, on an intranet, or on an extranet. Yet, despite the increased flexibility, designing (or redesigning) an application for WebClient is not complicated. You need only follow a few guidelines, which this chapter explains.

The guidelines are:

- Segregating user-interface logic and business logic
- Accessing databases only through an AppServer
- Modularizing code by function

2.1 Segregating User-interface Logic and Business Logic

You must distribute the application's code among client and servers so that user-interface logic resides on the client while business logic resides on one or more AppServers.

To help you segregate user-interface logic and business logic, Progress provides SmartObjects™. While not required, SmartObjects can greatly simplify the process of assembling the WebClient application. For more information, see the chapter on SmartObjects in the *Progress AppBuilder Developer's Guide*.

2.2 Accessing Databases Only Through an AppServer

Databases must be accessed through an AppServer. They cannot be accessed directly by the client. That is, all database data presented by the application to the end user must originate on an AppServer, while all database data input by the end user to the application must be sent to an AppServer for evaluation and storage. To pass relational-database data (such as rows of data) between AppServer and client, you can use TEMP–TABLE parameters in remote procedures. For more information on TEMP–TABLE parameters and remote procedures, see the *Progress Programming Handbook*. In addition, the SmartObjects provided by Progress for packaging business logic on an AppServer also provide access to any databases or DataServers the application requires.

2.3 Modularizing Code by Function

Imagine you have just designed a new application's procedures and user-defined functions, and you are now assigning each piece of code to a procedure (`.p`) file. (Or perhaps you are examining an existing application and noting how the existing procedures and user-defined functions are assigned to procedure files.). At application-deployment time, you might want only certain *components* (groups of application files) to download initially and might want other components to download only when they are called for the first time. At design time, you can make deploying the application more efficient by the way you assign code to procedure files. Specifically, avoid assigning to the same procedure file pieces of code that support dissimilar functions, since you might want unrelated modules of an application to download at different times.

NOTE: There is no harm, however, in assigning to separate procedure files pieces of code that support similar functions, since a single component can contain multiple files.

For example, consider an insurance application that supports several different lines of insurance: life, health, automobile, fire, and disability. Assume the application has separate code supporting each line of insurance, plus general-purpose routines used throughout the application. To modularize this application by function, make sure that:

- A single procedure file does not contain routines for more that one line of insurance.
- A procedure file containing general-purpose routines does not contain other routines.

NOTE: If an existing application consists entirely of procedures and user-defined functions that are completely unmodular, with every routine calling every other routine, it might be difficult to have the application download incrementally. You can, however, deploy the application as a single large component. For more information, see Chapter 7, "Deploying an Application."

3

Designing Your End User's Experience

Before you begin your application development, you must make decisions about the following:

- How end users install WebClient
- How end users install WebClient applications
- How to provide updates for your application
- How end users access and start your application
- The type of server to use to host and deploy your application
- What type of documentation you need to provide

The decisions you make impact your end users and how you develop and deploy your application. To help you make these design decisions, this chapter provides information about:

- WebClient installation
- WebClient application installation
- Updates for your application
- Uninstalling your application
- Selecting deployment servers
- Starting WebClient applications
- Providing documentation

3.1 WebClient Installation

You must provide a way for your end users to install WebClient on their machines. You can do one of the following to provide a WebClient installation:

- Host an installation on a Web server
- Point your users to the Progress WebClient installation page
- Provide your users with a CD-ROM with a WebClient install program

Progress provides the following WebClient installers:

- A stand-alone InstallShield image that you can use to create an installation CD-ROM
- A One-Click Install image that you can use to host a Web-enabled installation

Both installers install WebClient on the user's machine but provide a different experience for your end user. If you provide a:

- CD-ROM installation, your end users use a setup program to install WebClient
- Web-enabled installation, your end users would go either to a Web page you provide or to the Progress Web site that provides a link for installing WebClient

Progress Software strongly recommends that you customize the One-Click Install image to automatically install the application after installing WebClient and to start the application after it is installed.

Depending on whether you customize your WebClient installation, your end users can have the following experience:

- If you do not customize your WebClient installation, your users need to initiate installing and starting a WebClient application as a separate step.
- If you do customize your WebClient installation, the application installation can follow in a continuous process that installs and starts the application with little interaction by the user.

NOTE: You cannot customize the CD-ROM installation.

For more information about WebClient installers provided by Progress and how to customize and prepare a WebClient installer for deployment, see Chapter 7, "Deploying an Application."

3.2 WebClient Application Installation

When you design your application, you must decide on the installer technology you want to use. You can deliver your application using CD-ROM, over the Internet, or a company intranet. Progress offers great flexibility for installing your application. You can use:

- The Progress WebClient IntelliStream technology to install front-end components for your application and perform optional System Tasks to satisfy basic installation requirements. This can also include a Progress 4GL install procedure to complete installation tasks that IntelliStream could not complete.

- An external installer, such as InstallShield® Professional, which includes the One-Click Install™ technology for installing the entire front end of your application over the Web, or technology to create a CD-ROM for your application installation.

NOTE: The technology you use to install the WebClient should not influence which technology to choose for installing your WebClient application.

For more information about using Progress WebClient IntelliStream, see the "IntelliStream" section. For more information about using an external installer, see the "External Installer" section.

Progress Software recommends that you provide a Web installation: IntelliStream or a Web-enabled external installer, as the typical approach. For users without access to high-speed Internet connections, you might want to provide a CD-ROM installation. You should provide all updates over the Web.

When designing your WebClient application delivery method, consider the following recommendations by Progress Software:

- Use Progress IntelliStream technology to install your application over the Internet or an intranet when you want to take advantage of one or more of the benefits provided by the IntelliStream technology that can:

  - Install only the required front-end components of your application initially and then install other components only when a user needs the functionality

  - Optionally install your application from an AppServer with integrated security

  - Automatically start the application after the install

  - Save you from buying and learning another installer technology

- Use an external install product, such as InstallShield® One-Click Install™ technology, to install the application from a Web page when IntelliStream does not support functionality you need to install the application.
- Use an external install product, such as InstallShield, when you want your end users to install the entire application using a CD-ROM.
- If you need to provide both a CD-ROM and Web installation for different customers, you can:
  - Use an external install product, such as InstallShield, for CD-ROM installs and IntelliStream for installs over the Internet or an intranet. Using this design gives your end users who have fast Internet access the advantage of using the IntelliStream features.
  - Use an external install product, such as InstallShield, for CD-ROM installs and the One-Click technology for installs over the Internet or an intranet.

3.2.1 IntelliStream

You can use IntelliStream to have WebClient install your application automatically with minimal interaction by your end users. When you use IntelliStream, WebClient reads the application configuration file to determine which front-end components to download initially. Depending on how you implement your application installation, your users might need to respond to dialog boxes that prompt them for security information and information about the components they want to download. For more information, see Chapter 5, "Designing Security," Chapter 2, "Designing the Application," and Chapter 7, "Deploying an Application."

The IntelliStream features include an installer that is built into WebClient to install and update your application by downloading the required application components or component updates.

IntelliStream provides the following features:

- Automatic download of component (`.CAB`) files
- Creation of an Add/Remove Program entry for uninstalling the application

Optionally, IntelliStream can:

- Set up application shortcuts on the desktop or in the Program folder
- Run `ini2reg` to store your `.ini` file information in the registry

- Install and register application and system files
- Run a 4GL install procedure that can complete application-specific installation tasks

When you use IntelliStream, you use the WebClient Application Assembler to define and generate the application component (`.CAB`) files and to choose the optional installation tasks that you want IntelliStream to perform.

3.2.2 External Installer

If you decide to use an external installer, such as InstallShield or a similar type of install program, Progress Software recommends that your installer perform all the installation tasks required to install the entire application. When you use an external installer, the installer should:

- Copy all required files to the end user's machine
- Create registry entries for WebClient and your application
- Provide an uninstall for the application, typically using the same technology as the install

Optionally, the installer can create a shortcut for running the application and can perform any additional tasks you want the installation to complete.

For more detailed information about installing your application, see Chapter 7, "Deploying an Application."

3.3 Updates for Your Application

You must choose a technology to update your application. You can provide updates using:

- **Progress IntelliStream technology —** When you use IntelliStream, you use WebClient Application Assembler to generate the deployment files necessary to update your application.

  NOTE: You can use IntelliStream to update your application even if you use an external installer to perform the initial installation. Progress Software recommends that you always update your application using IntelliStream regardless of whether you used IntelliStream or an external installer to perform the initial installation.

- **The same update method as your external installer technology** — When you use an external installer technology, you must provide a Web-enabled update install.

For more detailed information about updating your application, see Chapter 7, "Deploying an Application."

3.4 Uninstalling Your Application

Typically, uninstall is also a feature of your installer technology. If you use:

- IntelliStream to install your application, WebClient provides the uninstall technology for your application.
- An external installer to install your application, your uninstall program must be made available to remove the application from the user's machine.
- An external installer to install your application and IntelliStream to update your application, the uninstall for your external installer must call WebClient Initializer in uninstall mode to remove any update files and registry key values that IntelliStream added.

For more information, see Chapter 7, "Deploying an Application."

3.5 Selecting Deployment Servers

You have a number of options when deciding on the type of server you want to use to deploy your application. You can use an Internet-based server: a Web server or file server, or you can use a Progress AppServer. Each server offers different advantages and supports different features.

For more information, see Chapter 7, "Deploying an Application" and Chapter 4, "Designing the Deployment Configuration."

3.6 Starting WebClient Applications

You can design your WebClient application deployment so that the application is started using:

- A URL to a bootstrap Web page so that the user can start the application from a Web browser. You must use this bootstrap process to install and run the application initially, and whenever you change the location of the configuration file.

  NOTE: Your users also can use this bootstrap Web page to initiate the installation of WebClient.

- A desktop shortcut that the user can use to start the application directly from the Windows desktop after the application is installed.

For more information, see Chapter 7, "Deploying an Application" and Chapter 4, "Designing the Deployment Configuration."

3.7 Providing Documentation

You must decide how you want to provide documentation for your end users. You have a number of options that can include printed or electronic documentation. Whatever you decide, the documentation must explain:

- How to install WebClient.
- How to install and uninstall the application.
- How to access and run the application.
- Optionally, how to manage application functions using the WebClient Application Manager that is installed with WebClient. For example, you must provide instructions to you users for setting user-specific startup parameters required by your application.

This chapter provides an overview of deployment design considerations you must make for your application. For more detailed information about deploying your WebClient applications, see Chapter 7, "Deploying an Application" and Chapter 8, "Your End User's Experience."

4

Designing the Deployment Configuration

Designing the deployment configuration is not complicated. This chapter describes your tasks, which might include:

- Selecting a server to host the application configuration file
- Selecting a server to host the codebase
- Selecting a server to host a non-IntelliStream application install
- Determining the best location for the codebase
- Selecting a communication protocol for configuration file downloads
- Selecting a communication protocol for codebase downloads

4.1 Selecting a Server to Host the Application Configuration File

WebClient requires that the application configuration file be downloaded from a standard *Internet-based server*, which can consist of a Web server or a file server.

You might host the application configuration file on a server you have already set up. For example, if you have already set up a WebServer with the AppServer Internet Adapter (AIA), you might host the configuration file on this Web server. Or, if you already have set up a file server (and are working entirely within a corporate intranet and are using file sharing), you might host the configuration file on the file server.

If the server you select already has security set up, the security can protect the application configuration file at little or no additional cost.

Once you select a server, then at application-deployment time, you describe the server to WebClient using the WebClient Application Assembler's Configuration File Locator Definition window, as shown in Figure 4–1.

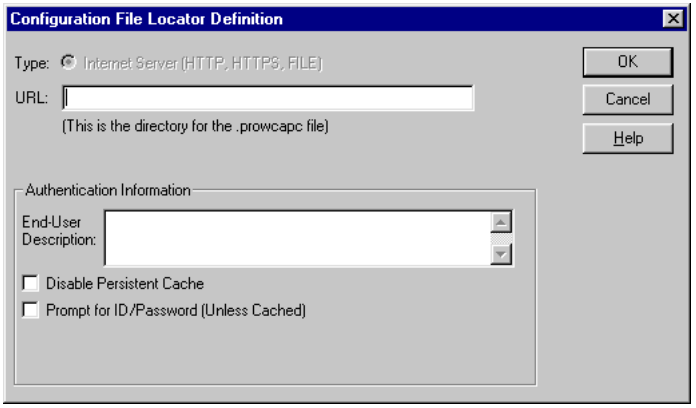

Figure 4–1: Describing the Configuration File Server to WebClient

4.2 Selecting a Server to Host the Codebase

If you are using IntelliStream, the codebase (application components for the initial download and for updates) can reside on a standard Internet-based server or on an AppServer.

Just as with the application configuration file, you might host the codebase on a server you have already set up. For example, if you have already set up an AppServer with the business logic the application accesses, you might host the codebase there.

And just as with the server for the application configuration file, if the server you select already has security set up, the security can protect the codebase at little or no additional cost.

Once you select a server, then at application-deployment time, you describe the server to WebClient using the Application Assembler's Codebase Locator Definition window, as shown in Figure 4–2.

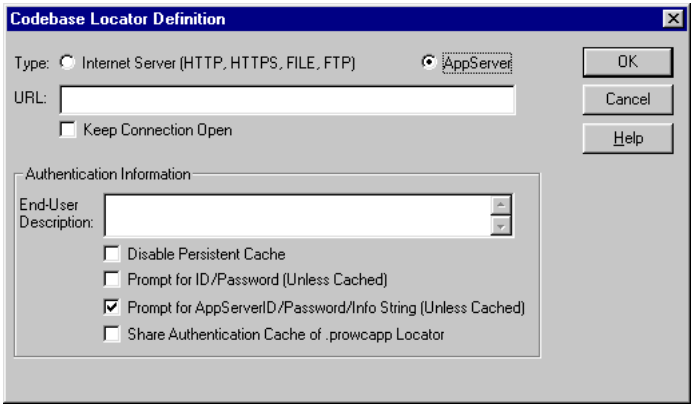

Figure 4–2: Describing the Codebase Server to WebClient

4.3 Selecting a Server to Host a Non-IntelliStream Application Install

Just as with the application configuration file, WebClient requires that a non-IntelliStream application install be downloaded from a standard Internet-based server.

And just as with the application configuration file, a non-IntelliStream application install can reside on a server you have already set up. For example, if you already have set up a file server (and are working entirely within a corporate intranet and are using file sharing), you might host the non-IntelliStream application install on the file server.

And just as with the application configuration file server, if the server you select already has security set up, the security can protect the non-IntelliStream application install at little or no additional cost.

4.4 Determining the Best Location for the Codebase

This section applies if you are using IntelliStream.

Given that the application configuration file must reside on an Internet-based server and that the application's business logic must reside on an AppServer, how do you determine the best location for the codebase?

4.4.1 Hosting the Codebase on an AppServer

Hosting the codebase on an AppServer has the following advantages:

- If the codebase can reside on the same AppServer that hosts the business logic, both sets of code reside on the same AppServer, which simplifies administrative tasks.
- If the AppServer hosting the business logic already has security (such as user IDs and passwords), placing the codebase on that AppServer lets the codebase be protected by the same security at little or no additional cost.

But hosting the codebase on an AppServer has the following disadvantage: It might require configuring an additional AppServer if the AppServer hosting the business logic cannot handle the additional strain of hosting the codebase without unacceptable performance degradation.

4.4.2 Hosting the Codebase on an Internet-based Server

Hosting the codebase on an Internet-based server has the following advantage: The application configuration file and the codebase can be managed consistently.

4.5 Selecting a Communication Protocol for Configuration File Downloads

When WebClient downloads the application configuration file from its server, the download uses a communication protocol based on the server's type. Recall that the application configuration file must reside on an Internet-based server, which can consist of a Web server or a File server. Table 4–1 lists the protocols for each server type.

Table 4–1: Communications Protocols for Configuration File Downloads

Server Type	Protocol
Web	HTTP or HTTPS
File	File

If the application configuration file resides on a Web server, downloads can use the HTTP or HTTPS protocol, whichever you prefer. The HTTPS protocol encrypts the download, which increases security. The HTTP protocol does not encrypt the download, but is somewhat faster.

If the application configuration file resides on a File server, downloads use the File protocol.

WebClient knows which protocol to use based on the URL you enter in the Application Assembler's Configuration File Locator window. For more information on URLs and communication protocols, see the "Appserver Internet Adapter" chapter of the *Progress Version 9 Product Update Bulletin*.

NOTE: Progress Software Corporation recommends that for access over the Web, you select a communication protocol that encrypts downloads. For more information, see Chapter 5, "Designing Security."

4.6 Selecting a Communication Protocol for Codebase Downloads

This section applies if you are using IntelliStream.

When WebClient downloads application components from their server, the download uses a communication protocol based on the server's type. Recall that the codebase can reside on an Internet-based server (which can consist of a Web server or a File server) or on an AppServer.

Table 4–2 lists the communication protocols for each server type.

Table 4–2: Communication Protocols for Codebase Downloads

Server Type	Protocol
Web	HTTP or HTTPS
File	File
AppServer	HTTP, HTTPS, or AppServer

If application components reside on a Web server, downloads can use the HTTP or HTTPS protocol. To choose between them, consider the following:

- The HTTPS protocol encrypts the download, which increases security.
- The HTTP protocol does not encrypt the download, but is somewhat faster.

If application components reside on a File server, downloads use the File protocol.

If application components reside on an AppServer, downloads can use the HTTP, HTTPS, or AppServer protocol. To choose among these protocols, consider the following:

- Using the HTTP or HTTPS protocol with an AppServer requires using the AIA.
- The HTTPS protocol encrypts downloads, which increases security.
- The HTTP protocol does not encrypt downloads, but is somewhat faster then HTTPS.
- The AppServer protocol does not use the AIA and does not encrypt downloads.

WebClient knows which protocol to use for codebase downloads based on the URL you enter in the Application Assembler's Codebase Locator window. For more information on URLs and communication protocols, see the "AppServer Internet Adapter" chapter of the *Progress Version 9 Product Update Bulletin*.

5

Designing Security

Gone are the days when you could ignore security and get away with it. Everyday, it seems, the media report yet another software virus or Web site break-in. Yet, if you apply security crudely, your end users might complain of having to enter the same logon ID and password multiple times. What you and your end users want is tight security with a clean user interface.

To achieve this goal, use the security tools described in this chapter. The tools are:

Security Tool	Description
Digital Signatures	The application deployer digitally signs each cabinet file to be downloaded, and the end user verifies each digital signature when the file downloads.
User IDs and Passwords	The application deployer assigns a user ID and password to each server accessed by WebClient and the application.
Security Caching	WebClient caches each user ID and password entered by the end user and retrieves cached authentication information to access additional objects that require the same user ID and password.
Secure Communication Protocols	The application deployer chooses communication protocols that can encrypt downloads.

5.1 Digitally Signing Files

When end users download files supposedly prepared by you, they want to be absolutely sure that each file has:

- *Authenticity* — Assurance that the file was truly prepared by you
- *Integrity* — Assurance that the file has not been tampered with

To provide these assurances, you can digitally sign each file to be downloaded. If you do, WebClient requires the end user to verify the digital signature of each file downloaded.

NOTE: Progress Software Corporation (PSC) strongly encourages the application deployer to digitally sign each file to be downloaded.

5.1.1 The Development of Digital Signatures

This section summarizes existing ciphers, their shortcomings, the invention of asymmetric ciphers, and how they provide digital signatures.

Existing Ciphers

Until the mid 1970s, the only ciphers available used a single key to encrypt and to decrypt — hence the name *symmetric ciphers*. An example is the *Data Encryption Standard* (DES), developed by IBM in 1977 and adopted soon after by the United States National Institute of Standards and Technology (formerly known as the National Bureau of Standards). Each DES-encrypted message uses one of 72 quadrillion (72×10^{15}) keys. In the hope that cryptologists would study it, try to break it, and uncover any weaknesses it might have, its developers published its algorithm, a practice common in cryptology. The security of DES rests in the belief that although you could crack a DES-encrypted message by trying all 72 quadrillion keys, doing so would take an unreasonably long amount of time, given the hardware and software available.

Shortcomings of Existing Ciphers

Since the development of DES in 1977, hardware and software have become so much faster that cryptologists now believe cracking a DES-encrypted message by trying all the possible keys within a reasonable amount of time is now possible. For this reason, NIST is currently soliciting proposals for a replacement for DES that is significantly stronger.

Symmetric ciphers in general have additional problems:

- As you increase the number of people in a group using a particular symmetric cipher, the number of keys required increases exponentially, since a unique key is required by each unique pair of people who wish to communicate. In other words, symmetric ciphers do not scale well.
- To use a symmetric cipher to communicate with someone you have never met, the two of you must first exchange keys, which can be difficult — as countless spy novels attest to.
- Storing keys conveniently and securely can be difficult. Storing them centrally might make them convenient but vulnerable to attack, while storing them in distributed locations might make them secure but inconvenient to retrieve.

Asymmetric Ciphers

To get around these problems, researchers in the mid 1970s developed *asymmetric ciphers*, which use different keys to encrypt and to decrypt. With asymmetric ciphers, each person within a group is assigned a unique pair of keys consisting of *public ke*y and a *private key*, where each public key is known to everyone, while the corresponding private key is known only to its owner. After a message is composed, it is encrypted by a process that uses the recipient's public key. In order for the recipient to read the message, it is decrypted by a process that uses the recipient's private key. The security of this arrangement rests on the belief that if a message is encrypted by a process that uses a particular public key, decrypting the message without using the corresponding private key, which is known only to the recipient, is *computationally unfeasible* — that is, not possible within a reasonable amount of time using available hardware and software.

Digitally Signing Documents

Some PKI software allows a document to be *digitally signed*, a process that involves applying the signer's private key. A digitally-signed document can be verified by anyone by following a process that involves applying the signer's public key. The security of this arrangement rests on the belief that if a digitally-signed document can be verified by a process that involves applying a particular public key, it would be computationally unfeasible to create the digitally-signed version except by a process that involves applying the corresponding private key, which is known only to its owner.

The overall system required to provide public-key encryption and digital-signature services is called *Public Key Infrastructure* (PKI). The PKI software WebClient uses, which employs Microsoft Authenticode Technology, allows documents to be digitally signed.

The Need for Trust

But public keys, private keys, and digital signatures alone do not ensure security. Consider an intruder who impersonates a legitimate member of a group. Suppose the intruder represents her own public key as the public key of the victim. And suppose the intruder digitally signs, with her own private key, files purporting to be from the victim. If you download these files, using the public key the intruder supplies and represents as the victim's, you might unknowingly install software that sends confidential information on your computer to the intruder, or that even destroys vital information on your computer. To avoid this, PSC recommends that you accept a public key only when you trust its authenticity and integrity.

Public-key Certificates

One way to distribute a public key while ensuring its authenticity and integrity is to package it in a *public-key certificate*, a document that contains a public key and that is digitally signed by a trusted *certificate authority* (CA). Before digitally signing a public-key certificate, the CA verifies the public key's authenticity and integrity. By using public keys, private keys, digital signatures, public-key certificates, and certificate authorities, members of a group can communicate with confidence that the messages they receive, whether email messages or downloaded files, have authenticity and integrity.

5.1.2 How WebClient Uses Digital Signatures

Now, apply your knowledge of digital signatures to WebClient. If you want to digitally sign each cabinet file to be downloaded and want the end user to verify the digital signature of each cabinet file downloaded, who needs which key?

- To digitally sign a cabinet file to be downloaded, you need:
  - Your private key
  - Your public key in the form of a public-key certificate.
- To verify the digital signature of a downloaded cabinet file, the end user needs your public key in the form of a public-key certificate.

So, in order to use digital signatures, you need a private key, a public key, and a public-key certificate, while your end user needs your public-key certificate. This subsection covers:

- How to get a private key, a public key, and a public-key certificate
- How to define an application as signed
- What WebClient does differently when an application is defined as signed
- How your public-key certificate gets to your end user
- Changing the definition of an application from signed to unsigned or from unsigned to signed
- How to create fake public-key certificates for testing

Getting a Private Key, Public Key, and Public-key Certificate

To get a private key, public key, and public-key certificate:

1 ♦ Select a PKI vendor (CA) whose software is compatible with Microsoft Authenticode Technology and request a "software publishing digital certificate."

To get names of CAs, ask your PSC product-marketing representative.

2 ♦ Get the software that generates and securely stores public keys and private keys on your system.

You can typically get the software from Microsoft or download it from the CA's Web site. You might have to provide a name for the certificate storage location.

3 ♦ Fill out the CA's request for information about you, your company, and how you are going to pay.

4 ♦ Submit the requested information and the stored public key to the CA.

Steps 2–4 are typically handled through a Web site.

5 ♦ Wait for the CA to verify your identity.

NOTE: The CA might use phone calls or personal visits to verify the information you supply.

6 ♦ If the CA can prove your individual and corporate identity, they will contact you and tell you how to obtain your digital certificate.

Typically, this involves same software and the same Web site as steps 2–4.

The digital certificate will be stored on your system in the same named certificate location as was used for the initial public-private key generation.

You can repeat Steps 2–6. And you can have digital certificates issued by multiple CAs for a single public-private key pair.

Defining an Application as Signed

Now that you have a private key, public key, and public-key certificate, you can define an application as signed. To do so, in the Web Client Application Assembler's Generate window, in the Digital Signature rectangle, check the From Registry (if the digital signature information resides in the registry) or From File (if the digital signature information resides in a file) radio button, as shown in Figure 5–1.

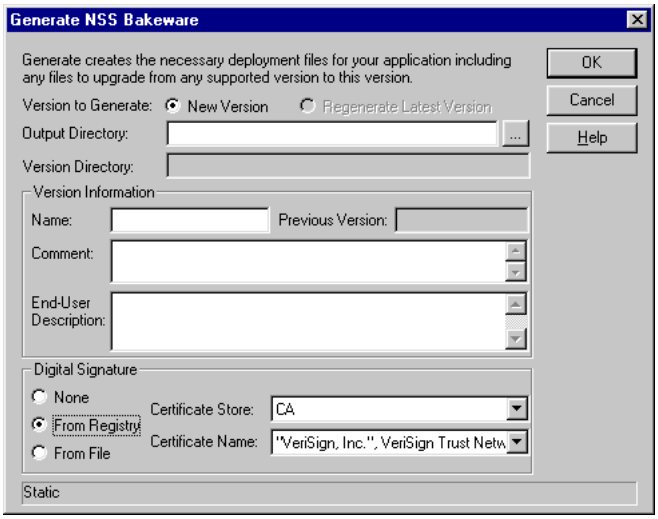

Figure 5–1: Defining an Application as Signed in the Application Assembler's Generate Window

What WebClient Does Differently for an Application Defined as Signed

If you define an application as signed, when you generate the application, the Application Assembler:

- Places a copy of the application configuration file into a cabinet (`.prowcapc`) file, adds a copy of your public-key certificate to the cabinet file, and digitally signs the cabinet file
- Places a copy of your application's files into cabinet (`.cab`) files and digitally signs each cabinet file

By contrast, if you define an application as unsigned, when you generate the application, the Application Assembler:

- Places a copy of the application configuration file into a cabinet file and leaves the cabinet file unsigned
- Places a copy of your application's files into cabinet files and leaves the cabinet files unsigned

How Your Public-key Certificate Gets to the End User

When an application is defined as signed and the end user downloads a signed configuration or component cabinet file (each of which contains your public-key certificate), WebClient on the end user's machine:

1. Extracts the digital signature and your public-key certificate from the cabinet file
2. Verifies the digital signature of the cabinet file using your public-key certificate

   Also verifies your public key certificate through its issuer's root public-key certificate. The issuer's root public-key certificate can be obtained from the cabinet file itself or from the certificate store used by Microsoft Internet Explorer.
3. Displays the information on the certificate and asks the end user if they trust it

   NOTE: If the end user says "No," the process aborts.
4. Optionally stores your public-key certificate into the Digital Certificate store of Internet Explorer.

Changing the Definition of an Application from Signed to Unsigned and from Unsigned to Signed

An application can change from being signed to unsigned and from unsigned to signed. That is, if the previous version of an application is defined as signed, you can define the current version as unsigned. And if the previous version of an application is defined as unsigned, you can define the current version as signed.

NOTE: PSC recommends not changing the status of an application from signed to unsigned.

If the end user downloads an application configuration file that is unsigned (indicating that the current version of the application is defined as unsigned) and the previous version of the application is defined as signed, WebClient asks the end user to confirm that it is OK for the application to be changing from signed to unsigned.

NOTE: Unless the end user has been informed of this change by a trusted source, PSC recommends that you instruct your end user to reject the change and contact you.

Creating Test Public-key Certificates

WebClient includes a batch file, `MakeTestCert.bat`, that makes it easier for you to create fake public-key certificates for testing. For more information, see the comments in the file, which resides at $DLC/`bin`, where $DLC represents the Progress installation directory.

NOTE: You cannot generate test certificates using `MakeTestCert.bat` on a machine running Windows 98. Also, you cannot generate a test certificate on one machine (say, one not running Windows 98) and use it on another machine (say, one running Windows 98).

5.2 Protecting Servers with User IDs and Passwords

This section covers authenticating end users accessing Internet-based servers and Progress Appservers and focuses on the security implications of embedding authentication information, such as user IDs and passwords, in a URL.

5.2.1 Using Authentication

For each Internet-based server and AppServer accessed by WebClient or the application, user IDs and passwords can be set. This can be done by you, or, in the case of an out-hosted server, by the out-hoster. Although user IDs and passwords are not required, using them strengthens security.

NOTE: PSC recommends that you seriously consider protecting each server accessed by WebClient or the application with a user ID and password.

For more information on user IDs and passwords for a particular Internet-based server, see the documentation for that server. For more information on setting user IDs and passwords for an AppServer, see the *Building Distributed Applications Using the Progress AppServer* manual.

5.2.2 The Dangers of Embedding User IDs and Passwords in URLs

Each Internet-based server or AppServer a WebClient application wants to access has a URL, which you specify to WebClient at application-definition time using the Application Assembler. If the Web server or Appserver requires a user ID or password, you can embed the authentication information in the URL. But doing so weakens security, since it might reveal user IDs and passwords to unauthorized personnel.

NOTE: PSC recommends that user IDs and passwords not be embedded in server URLs.

5.2.3 Prompting the End User for Authentication Information

Instead of embedding authentication information in a server URL, you can ask WebClient to prompt the end user for them when they are needed. How you do this depends on whether the server is an application configuration-file server (from which the application configuration file is downloaded) or a codebase server (from which application components are downloaded):

- For an application configuration file server, in the Application Assembler's Configuration File Locator Definition window, in the Authentication Information rectangle, fill in the End-User Description field and check the "Prompt for ID/Password (Unless Cached)" box.

  Figure 5–2 shows the WebClient Application Assembler's Configuration File Locator Definition window.

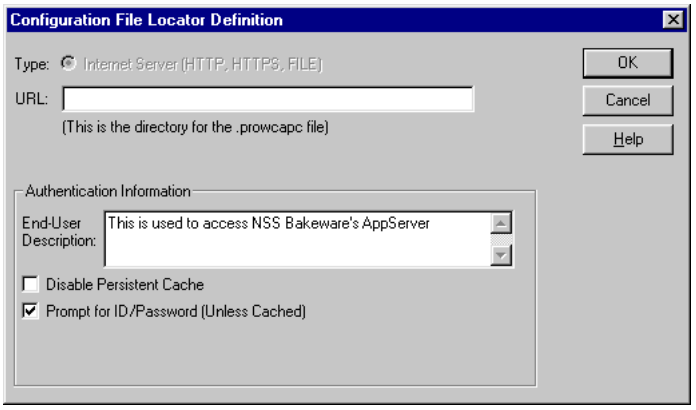

Figure 5–2: Asking WebClient to Prompt for Configuration File Server Authentication Information

NOTE: WebClient uses this information only if the end user starts an already-installed WebClient application using a shortcut. For more information on starting an already-installed application using a shortcut, see Chapter 7, "Deploying an Application."

Figure 5–3 shows the subsequent authentication dialog displayed to the end user.

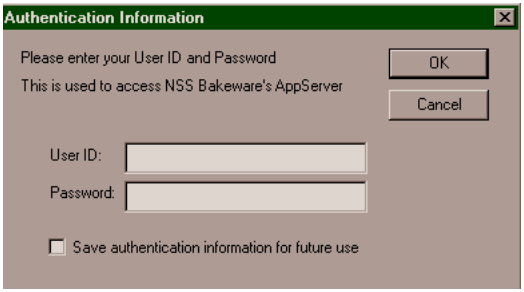

Figure 5–3: Configuration File Server Authentication Information Dialog

- For a codebase server, in the Application Assembler's Codebase Locator Definition window, in the Authentication Information rectangle, fill in the End-User Description field. Then, check the "Prompt for ID/Password (Unless Cached)" button. Finally, if the codebase server is an AppServer, check the "Prompt for AppServerID/Password/Info String (Unless Cached)" button.

  Figure 5–4 shows the Application Assembler's Codebase Locator Definition window.

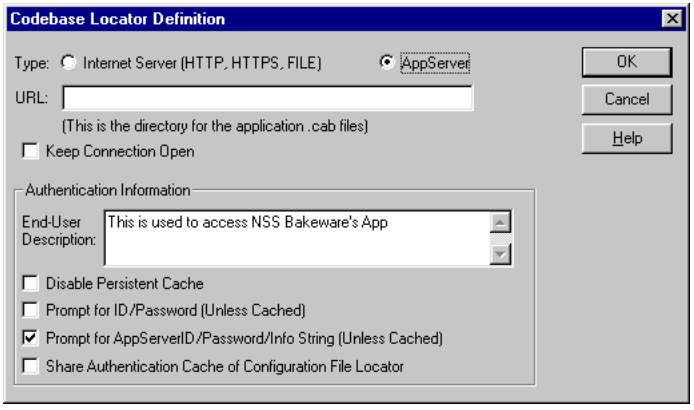

Figure 5–4: Asking WebClient to Prompt for Codebase Server Authentication Information

Figure 5–5 shows the subsequent authentication dialog displayed to the end user.

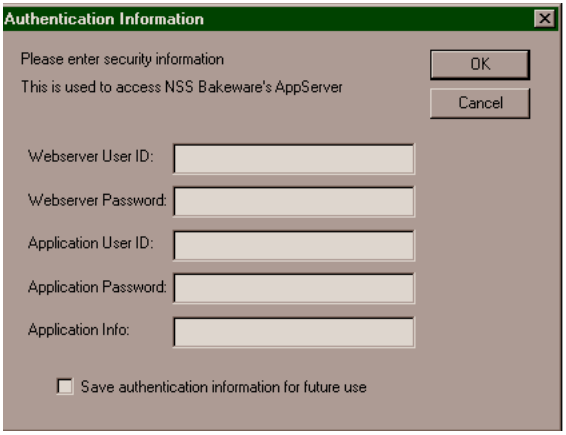

Figure 5–5: Codebase Server Authentication Information Dialog

5.3 Single Sign-on and Security Caching

When WebClient prompts the end user for authentication information to connect to a server to download the application configuration file or application components, WebClient can cache the authentication information and make it available to the application. Similarly, when the application prompts the end user for authentication information to connect to a server that contains business logic, the application can make the authentication information available to WebClient. By default, WebClient maintains a separate security cache for the application configuration-file server and the codebase server.

Security caching lets you implement *single sign-on*, which means the end user is not prompted multiple times for the same authentication information. Single sign-on is useful when:

- Connecting to the same server multiple times during one session or across multiple sessions. Suppose a single AppServer contains application components to be downloaded and business logic that the application uses. If WebClient downloads application components before the application accesses the business logic, WebClient can automatically make the authentication information available to the application. The application can then connect to the AppServer to access the business logic without having to reprompt the end user for the authentication information. Similarly, if the application accesses the business logic before WebClient downloads application components, the application can make the authentication information available to WebClient. WebClient can then connect to the AppServer to download application components without having to reprompt for the authentication information.

- Connecting to multiple servers that use the same authentication information. Suppose the application configuration file to be downloaded and the application components to be downloaded reside on separate servers but are protected by the same authentication information. You can ask WebClient to make the configuration file's authentication information available to the application as codebase authorization information (but not the other way around).

NOTES: Authentication information is always stored encrypted.

Sharing the configuration file cache with the codebase cache works only if the application is launched from a shortcut, not from a Web browser — because in the latter case, the configuration file is downloaded by the Web browser, whose cache WebClient cannot access.

By default, WebClient does **not** maintain security caches on a particular machine across sessions. This is called the *persistent cache*, which the end user must specifically request.

You can tell WebClient to disable the persistent cache. If you do so, the end user does not have the option of saving authentication information across sessions, and WebClient deletes the security caches at the end of each WebClient session.

NOTE: If persistent caching is not disabled, an end user enters authentication information for particular servers, and the end user chooses to cache them persistently, a subsequent end user starting a new WebClient session at the same machine and logging in as the original end user can access those servers without having to re-enter the authentication information.

5.3.1 Authentication Dialogs

Suppose the same AppServer contains application components to be downloaded and business logic to be accessed by your application. Suppose further that your application needs to access the AppServer before WebClient downloads any `As Needed` application components from the AppServer. (Your application knows this because the security cache does not contain authentication information (such as user IDs or passwords).) Your application wants to prompt the end user for the security information. To do so, your application can display the security dialogs that WebClient uses. This will provide the end user with a consistent interface.

For information on calling the authentication dialogs WebClient uses from the application, see the "Using WebClient's Authentication Dialogs in Your Application" section of Chapter 6, "Developing the Application."

5.4 Choosing a Secure Communication Protocol for Configuration File and Codebase Downloads

There might be unauthorized personnel listening in on your application's network traffic. While detecting such eavesdropping is not easy, you can make the eavesdropper's job much more difficult by encrypting application configuration file and application-component downloads — which you can do by using a secure communications protocol. An example of such a communications protocol is HTTPS, which encrypts network traffic using a combination of symmetric and asymmetric ciphers. (For more information on symmetric and asymmetric ciphers, see the "Digitally Signing Files" section).

NOTE: If application components to be downloaded reside on an AppServer and you select the HTTPS protocol (or the HTTP protocol) for application-component downloads, you must use the AppServer Internet Adapter (AIA).

For more information on choosing a communication protocol for application configuration file and codebase downloads, see Chapter 4, "Designing the Deployment Configuration." For more information on the HTTPS protocol and the AIA, see the "Appserver Internet Adapter" chapter in the *Progress Version 9 Product Update Bulletin*.

6

Developing the Application

Although coding an application for WebClient closely resembles coding an application for the 4GL client, WebClient provides several additional features you can take advantage of.

This chapter covers:

- Using the CODEBASE-LOCATOR handle and its attributes
- Managing AppServer connections
- Implementing single sign-on
- Using WebClient's authentication dialogs in your application
- Using URLs in the PROPATH to find and download image files over the Web

6.1 Using the CODEBASE-LOCATOR Handle and its Attributes

Once you describe the codebase server to WebClient (using the WebClient Application Assembler's Codebase Locator Definition window) and this description is stored in the application configuration file, WebClient makes this description available to the client at run time in attributes of the CODEBASE-LOCATOR handle. This section discusses these attributes one group at a time.

6.1.1 Basic Attributes

Table 6–1 lists the basic attributes of the CODEBASE-LOCATOR handle.

NOTE: These attributes are read-only.

Table 6–1: Basic Attributes of the CODEBASE-LOCATOR Handle *(1 of 2)*

Name	Data Type	Description
LOCATOR-TYPE	Character	"AppServer" or "InternetServer"
URL	Character	URL of the server
NEEDS-PROMPT	Logical	If TRUE, enables prompting for a user ID and password for accessing the codebase server. If LOCATOR-TYPE is "InternetServer," the prompt is for accessing a codebase residing on an Internet-based server If LOCATOR-TYPE is "AppServer," the prompt is for accessing the AIA (AppServer Internet Adapter) for a codebase residing on an AppServer
NEEDS-APPSERVER-PROMPT	Logical	If TRUE, enables prompting for a user ID, password, and info string for accessing an AppServer Applies only if LOCATOR-TYPE is "AppServer"

Table 6–1: Basic Attributes of the CODEBASE-LOCATOR Handle *(2 of 2)*

Name	Data Type	Description
END-USER-PROMPT	Character	Text string for the security prompt of an authentication dialog Used only if WebClient must prompt for security information
KEEP-CONNECTION-OPEN	Logical	If TRUE, any connection WebClient creates to download `As Needed` components will be kept open until all `As Needed` components have been downloaded to the end user's machine
PERSISTENT-CACHE-DISABLED	Logical	If TRUE, the user is not given the option of saving authentication information past the end of the session

6.1.2 Security-cache Attributes

Table 6–2 lists the attributes of the CODEBASE-LOCATOR handle that relate to the security cache.

NOTE: These attributes are all read-write.

Table 6–2: Security-cache Attributes of the CODEBASE-LOCATOR Handle
(1 of 2)

Name	Data Type	Description
URL-USERID	Character	User ID for accessing the server If LOCATOR-TYPE is “InternetServer,” the user ID is for accessing a codebase residing on an Internet-based server Unused if the URL does not take a user ID (for example, for a URL of type FILE) If LOCATOR-TYPE is “AppServer,” the user ID is for accessing AIA for a codebase residing on an AppServer
URL-PASSWORD	Character	Password for accessing the server If LOCATOR-TYPE is “InternetServer,” the password is for accessing a codebase residing on an Internet-based server Unused if the URL does not take a password (for example, for a URL of type FILE) If LOCATOR-TYPE is “AppServer,” the password is for accessing AIA for a codebase residing on an AppServer
APPSERVER-USERID	Character	User ID for connecting to the AppServer Unused if LOCATOR-TYPE is “InternetServer”
APPSERVER-PASSWORD	Character	Password for connecting to the AppServer Unused if LOCATOR-TYPE is “InternetServer”

Table 6–2: Security-cache Attributes of the CODEBASE-LOCATOR Handle
(2 of 2)

Name	Data Type	Description
APPSERVER-INFO	Character	Info string to use for connecting to the AppServer Unused if LOCATOR-TYPE is "InternetServer"
KEEP-SECURITY-CACHE	Logical	If TRUE, WebClient saves the security-cache attributes and this attribute, restoring them the next time the end user re-runs the application. The default is FALSE Set to TRUE if the end user, when asked for authentication information, sets the "Save Authentication Information for Future Use" toggle Can also be set to TRUE by the application, depending on the security dialog used The application might set this before the application does. For more information, see the "Managing AppServer Connections" section **NOTE:** Available only when the PERSISTENT-CACHE-DISABLED attribute is FALSE

6.1.3 Additional Attribute

Table 6–3 lists an additional attribute of the CODEBASE-LOCATOR handle.

NOTE: This attribute is read-write.

Table 6–3: Additional Attribute of the CODEBASE-LOCATOR Handle

Name	Data Type	Description
SERVER	Handle	Handle to the AppServer WebClient should use to perform the next application-component download Initialized to the unknown (?) value

At startup, WebClient initializes SERVER to the unknown (?) value.

The main purpose of this is to use the same AppServer connection for business logic as for downloading `As Needed` components. To do this, the application sets this once the AppServer connection is established.

If SERVER is not set by the application, then, after WebClient downloads the first `As Needed` component, WebClient sets SERVER to the value of the AppServer's handle if all of the following are true:

- The server's TYPE is "AppServer."
- KEEP-CONNECTION-OPEN is TRUE.
- At least one `As Needed` component remains to be downloaded.

6.2 Managing AppServer Connections

To download `As Needed` application components from a codebase residing on an AppServer, WebClient, by default, creates a new connection to the AppServer, downloads the components, and terminates the connection.

NOTE: To download `At Startup` application components before the application starts, WebClient creates a new connection to the codebase server.

If you want, WebClient can keep the connection used for downloading `As Needed` application components open and reuse the connection to download additional components until no more remain to be downloaded. To communicate this preference to WebClient, in the Application Assembler's Codebase Locator window, check the Keep Connection Open toggle box.

And if you want, WebClient can download `As Needed` application components using an AppServer connection you already have open. (Perhaps your business logic and the codebase reside on the same AppServer, and you have already connected to this AppServer to access the business logic.) To communicate this preference to WebClient, just after you connect to the AppServer, assign the connection's handle to the SERVER attribute of the CODEBASE-LOCATOR handle.

If you do so:

- The application has full responsibility for managing the connection.
- The application configuration file's information regarding the codebase server is not used.
- If the connection breaks, the download fails.
- If there is an outstanding ASYNC request on the handle of the existing connection at the moment WebClient needs it for the download, WebClient raises an error on the RUN statement.

  NOTE: For this reason, Progress Software Corporation recommends that an AppServer connection used for ASYNC requests not be used for downloading application components.

6.3 Implementing Single Sign-on

The application can use attributes of the CODEBASE-LOCATOR handle to implement *single sign-on* — which eliminates the need for the end user to enter the same user ID or password multiple times. Single sign-on is illustrated by the following scenarios:

- WebClient prompts the end user for authentication information and stores the end user's responses in the security cache. Then, the application accesses the cached authentication information using the CODEBASE-LOCATOR attributes to connect to the AppServer if both the codebase and business the logic reside on the same server.

- The application deployer checks the Share Authentication Cache of Configuration File Locator toggle box (in the Application Assembler's Codebase Locator Definition window). After the end user enters values for URL-USERID and URL-PASSWORD (or values for these items already reside in the persistent cache) to download the configuration file, WebClient makes these values available through the CODEBASE-LOCATOR attributes. These can be used to connect to the AppServer through AIA if the same Web server is used for AIA and the configuration file.

If the application requests authentication information before WebClient does, the application might set some or all of the security-cache attributes, which WebClient could then pick up. This might happen if WebClient starts the application, finds that nothing has changed since the last time it ran, and so does not download any components or updates. Then, the application starts, connects to a server, discovers it is the first to connect, and populates these attributes.

If the application sets any of these attributes, the next time WebClient needs to connect to a server, WebClient uses these cached values, rather than reprompting the end user. For more information on these attributes, see the "Using the CODEBASE-LOCATOR Handle and its Attributes" section.

6.4 Using WebClient's Authentication Dialogs in Your Application

Suppose your application needs to access the AppServer and the following are all true:

- The same AppServer contains application components to be downloaded and business logic to be accessed by your application.
- The security cache does not contain authentication information, and the application detects this.

  NOTE: This would occur when WebClient has not yet downloaded any `As Needed` application components from the AppServer.

To access the AppServer in this situation, your application can prompt the end user for security items by displaying the security dialogs that WebClient uses. This provides the end user with a consistent interface.

These routines reside, in the form of procedure (`.r`) files, in the WebClient installation at $DLC/`wcadd`, where $DLC represents the Progress installation directory.

NOTE: To make the dialogs look exactly the same as when WebClient invokes them, obtain the `prompt` input parameter from CODEBASE-LOCATOR:END-USER-PROMPT.

NOTE: Before calling an authentication dialog, the application is responsible for obtaining the relevant attribute values from the CODEBASE-LOCATOR handle to pass to these routines. For example, the value of NOT PERSISTENT-CACHE-DISABLED should be passed for the second parameter, `enable-persistent-checkbox`.

Also, if some but not all the required values are available from the CODEBASE-LOCATOR handle, they can be passed in as default values, allowing the user to fill in the remaining values.

NOTE: After calling an authentication dialog, the application is responsible for setting the appropriate CODEBASE-LOCATOR attributes with the information obtained from the end user.

6.4.1 Displaying the AIA Authentication Dialog

To display WebClient's AIA authentication dialog, use the _getAIAAuthentication.p procedure, which has the following signature:

```
DEFINE INPUT        PARAMETER prompt                    AS CHARACTER.
DEFINE INPUT        PARAMETER enable-persistent-checkbox AS LOGICAL.
DEFINE INPUT-OUTPUT PARAMETER url-userid                AS CHARACTER.
DEFINE INPUT-OUTPUT PARAMETER url-password              AS CHARACTER.
DEFINE INPUT-OUTPUT PARAMETER persistent                AS LOGICAL.
DEFINE OUTPUT       PARAMETER choseOK                   AS LOGICAL.
```

Use this procedure if the codebase and business logic share authentication information and there is no authentication at the AppServer itself.

To invoke the _getAIAAuthentication.p procedure, call it using a partially-qualified pathname that starts with the wcadd directory, as in the following code fragment:

```
RUN wcadd/_getAIAAuthentication.p(parameter-list).
```

6.4.2 Displaying the AppServer Authentication Dialog

To display WebClient's AppServer authentication dialog, use the _getAppServerAuthentication.p procedure, which has the following signature:

```
DEFINE INPUT        PARAMETER prompt                    AS CHARACTER.
DEFINE INPUT        PARAMETER enable-persistent-checkbox AS LOGICAL.
DEFINE INPUT-OUTPUT PARAMETER appserver_userid          AS CHARACTER.
DEFINE INPUT-OUTPUT PARAMETER appserver_password        AS CHARACTER.
DEFINE INPUT-OUTPUT PARAMETER appserver_info            AS CHARACTER.
DEFINE INPUT-OUTPUT PARAMETER persistent                AS LOGICAL.
DEFINE OUTPUT       PARAMETER choseOK                   AS LOGICAL.
```

Use the _getAppServerAuthentication.p procedure if the codebase and business logic reside on the same AppServer or share the same authorization information and

- Authentication is required only at the AppServer and not at the AIA Web server (or you are not using AIA)

  or

- Authentication is required at AIA and at the AppServer, but AIA authorization information is set in the security cache and thus available in CODEBASE-LOCATOR:URL-USERID and CODEBASE-LOCATOR:URL-PASSWORD.

  NOTE: In this case, you might instead call _getAIAandAppServerAuthentication.p, passing in the known values and allowing the end user to fill in the missing values.

To invoke the _getAppServerAuthentication.p procedure, call it using a partially-qualified pathname that starts with the `wcadd` directory, as in the following code fragment:

```
RUN wcadd/_getAppServerAuthentication.p(parameter-list).
```

6.4.3 Displaying the AIA and AppServer Authentication Dialog

To display WebClient's AIA and AppServer authentication dialog, use the _getAIAandAppServerAuthentication.p procedure, which has the following signature:

```
DEFINE INPUT        PARAMETER prompt                    AS CHARACTER.
DEFINE INPUT        PARAMETER enable-persistent-checkbox AS LOGICAL.
DEFINE INPUT-OUTPUT PARAMETER url-userid                AS CHARACTER.
DEFINE INPUT-OUTPUT PARAMETER url-password              AS CHARACTER.
DEFINE INPUT-OUTPUT PARAMETER appserver_userid          AS CHARACTER.
DEFINE INPUT-OUTPUT PARAMETER appserver_password        AS CHARACTER.
DEFINE INPUT-OUTPUT PARAMETER appserver_info            AS CHARACTER.
DEFINE INPUT-OUTPUT PARAMETER persistent                AS LOGICAL.
DEFINE OUTPUT       PARAMETER choseOK                   AS LOGICAL.
```

Use the _getAIAandAppServerAuthentication.p procedure if the codebase and the business logic share authorization information, you are using the AIA, and authorization is required by the AIA server and by the AppServer.

To invoke the _getAIAandAppServerAuthentication.p procedure, call it using a partially-qualified pathname that starts with the `wcadd` directory, as in the following code fragment:

```
RUN wcadd/_getAIAandAppServerAuthentication.p(parameter-list).
```

6.5 Using URLs in the PROPATH to Find and Download Image Files Over the Web

There can be URLs in the PROPATH environment variable of a WebClient client. If there are, and the client executes the SEARCH() function or the LOAD-IMAGE() method, the URL pathnames are processed along with the other pathnames included in the PROPATH. This feature lets the application access files, especially image files, over the Web.

6.5.1 Using the SEARCH() Function

This is the syntax of the SEARCH function, as specified in the *Progress Language Reference*:

SYNTAX

```
SEARCH ( opsys-file )
```

The SEARCH() function searches the directories and libraries (including URL pathnames) appearing in the PROPATH environment variable for *opsys-file*. The SEARCH() function returns the full pathname of the file unless it resides in the current working directory. If the SEARCH() function does not find the file for any reason, it returns the unknown value (?).

If the file is found in a directory specified by a URL, SEARCH() returns the full URL pathname of the file, which consists of the URL's PROPATH entry with the filename appended.

If *opsys-file* is a fully qualified URL or a fully qualified pathname, SEARCH() checks for the existence directly, and does not search the directories and URLs in the PROPATH.

NOTE: SEARCH() does not download any files.

6.5.2 Using the LOAD–IMAGE() Method

This is the syntax of the LOAD–IMAGE() method, as specified in the *Progress Language Reference*:

SYNTAX

```
LOAD-IMAGE ( filename
  [ ,
    x-offset ,
    y-offset ,
    width ,
    height
  ]
)
```

The LOAD–IMAGE() method applies to images and buttons. LOAD–IMAGE() reads the image contained in *filename*. LOAD–IMAGE() recognizes PROPATH entries consisting of URLs. If LOAD-IMAGE() finds an image in the directory specified by a URL, LOAD–IMAGE() downloads the image from the Web server and loads it into local memory directly, bypassing the end user's disk.

If the *filename* argument is fully qualified (whether URL or local pathname), LOAD–IMAGE() loads the image file directly, without searching directories or URLs in PROPATH.

If LOAD–IMAGE() cannot load the specified image file, it returns an error indicating the reason for the failure.

NOTE: You can also download images over the web by using the LOAD–IMAGE–DOWN(), LOAD–IMAGE–UP(), LOAD–IMAGE–INSENSITIVE(), and LOAD–ICON() methods. This feature is less useful, however, because these methods are generally called for transient graphical-interface events requiring an immediate response — which a Web download might not be able to satisfy.

6.5.3 Handling Errors

If the application uses URLs in the PROPATH, the application is responsible for handling all errors, which might involve URLs, connections, and authentication.

URL Specification Errors

If the PROPATH contains an ill-formed URL (for example, one with the percent symbol (%), which is not allowed), the following processing occurs:

- If the URL is not a fully-qualified path, SEARCH() and LOAD–IMAGE() do not raise an error, skip the URL, and continue searching with the next entry in the PROPATH.
- If the URL is fully-qualified path, SEARCH() and LOAD-IMAGE() raise an error.

Connection Errors

If the PROPATH contains a correctly-formed URL but the connection to the Web server fails, the following processing occurs:

- If the URL is not a fully-qualified path, SEARCH() and LOAD–IMAGE() do not raise an error and searching continues with the next entry in the PROPATH.
- If the URL is fully-qualified path, SEARCH() and LOAD-IMAGE() raise an error.

Authentication Errors

If an application uses SEARCH() or LOAD–IMAGE(), the PROPATH contains a URL, and authentication on a Web server fails, the following processing occurs:

- If the URL is not a fully-qualified path, SEARCH() and LOAD-IMAGE() do not raise an error and searching continues with the next entry in the PROPATH.
- If the URL is a fully-qualified path:
  - The SEARCH() function does not raise an error but returns the unknown value (?).
  - The LOAD–IMAGE(), LOAD–IMAGE–DOWN(), LOAD–IMAGE–UP(), LOAD–IMAGE–INSENSITIVE(), and LOAD–ICON() methods raise error 9368, where *server-name* is the host name of the Web server:

```
Unable to download file from the web. Authentication failed for
server server-name.
```

You can trap this error, request a user ID and password, and try to download image again, as in the following example:

```
DEFINE BUTTON Button1.
Button1:LOAD-IMAGE
   ("http://userid:password@www.progress.com/myimage.jpg")
NO-ERROR.
IF ERROR-STATUS:GET-NUMBER(1) = 9368 THEN DO:
   /* Display dialog. */
   /* Update URL string. */
   /* Retry load-image. */
END.
```

7

Deploying an Application

Progress WebClient provides great flexibility for deploying and updating your application across the Web or a company intranet or extranet. Deployment of an application requires generating the necessary files used to install, run, and update an application, configuring your distributed environment, and placing files on the appropriate server for delivery to end user machines.

This chapter provides information about using the WebClient Application Assembler to help you assemble your application for deployment and updates. This chapter also documents considerations you must make when deploying files on servers for your application, installing WebClient, and installing and updating your application.

The following topics are documented in this chapter:

- Defining your application using the WebClient Application Assembler
- Running the WebClient Application Assembler
- Files generated by the WebClient Application Assembler
- Hosting the application on a server
- Installing WebClient
- Bootstrapping a WebClient application
- WebClient application updates

7.1 Defining Your Application Using the WebClient Application Assembler

You use the WebClient Application Assembler to define how your application is installed and updated. The WebClient Application Assembler offers flexibility for deploying and managing your application. You can use the IntelliStream features to automatically download and update the front-end components of your application. Using IntelliStream, you can have an application installed and updated on an as-needed basis. You can also install and update your application using an external installer. If you want, you can also provide your end user with the option of updating an application or continuing to use an older but acceptable version of the installed application.

A WebClient application definition is contained in a WebClient project file (`.wcp` file) that represents the latest version of an application. After using the WebClient Application Assembler to define the WebClient application, you generate a set of deployment files for this latest version of your application. These generated deployment files are placed in a version-specific directory under an output directory you specify using the WebClient Application Assembler.

NOTE: The Progress WebClient Installation does not include any code from the Progress Application Development Environment (ADE), for example the adecomm code, ADM and ADM2 code, or any of the following Progress ActiveX controls: CSComboBox, CSSpin, PSTimer. If your application uses any of this code, you must package it into your application installation with the rest of your application code. For more information about adding the Progress ActiveX controls, see the "Options Tab" section.

Before you begin to assemble your application, you must decide whether you want to use the IntelliStream application install feature or an external installer for your WebClient application. For more information about the different options available, see Chapter 3, "Designing Your End User's Experience."

7.1.1 IntelliStream

When you use IntelliStream, the WebClient Application Assembler generates all the deployment files necessary to install and run your application. In addition, when you make changes to your application, use the WebClient Application Assembler to easily generate the files you need to update your application.

You take advantage of the IntelliStream features by defining application components. To define a component, your compiled application code must be available to your development machine. In addition, you must package the compiled application code to reflect how you want to deploy the application. For example, you must add your `.r` code to the `.pl` files you want, and all the files must reside in the same directory structure that you intend to use on your end users machine. Therefore, you must be able to access these files using the WebClient Application Assembler's Add Files to Component dialog box. For more information about developing your application and compiling your code, see Chapter 6, "Developing the Application."

Application Components

Components are essential to taking advantage of the IntelliStream feature. Components are collections of one or more files grouped together based on related functionality and are put into a compressed `.CAB` file for deployment. Components are used to deliver the application files and should meet the following requirements:

- Code organized into a component should be functionally related because the component is automatically installed as a unit to an end user's machine. Components can be installed as follows:
  - Before the application first runs (At Startup)
  - When end users run your application and they need specific functionality (As Needed)
  - As a choice to users where they can decide to install functionality before the application runs (Ask User at Startup).

- You can have only one At Startup component. The At Startup component is downloaded before the application runs the first time. This is usually the main code used to start the application and code needed by all users. You identify this unique component by choosing the At Startup option when you define the component in the WebClient Application Assembler's Component Definition dialog box.

- Components should be self contained so that all procedures that call each other are together. If you have a procedure that is called from many parts of your application, you might want to include this procedure in the At Startup component so it is available to all other components.

For more information about components, see, the "Running the WebClient Application Assembler" section, the WebClient Application Assembler help, Chapter 2, "Designing the Application," and Chapter 6, "Developing the Application".

System Tasks

System files, such as `.dll` files, or application-specific files that require special processing, such as updates to the registry, must be defined as System Tasks using the Options tab in the WebClient Application Assembler. Files specified using the Options tab, are automatically added to the At Startup component.

Progress 4GL Install Procedure

You can augment the IntelliStream install capabilities with a 4GL install procedure. This procedure is specified using the Options tab in the WebClient Application Assembler. This procedure and any code that it requires must be put in the At Startup component. WebClient runs this procedure at the end of its installation tasks. If you use a 4GL install procedure, you can also provide an 4GL install uninstall procedure that runs at the end of the uninstall tasks. For more information about using Progress 4GL installers, see the "Options Tab" section.

Uninstall

IntelliStream includes an uninstaller that removes your application and its registry entries from your end user's machine.

7.1.2 External Installer

If you choose to use an external installer that you create yourself, the external installer must complete all the installation tasks required to install your application and must also install all registry keys required by WebClient. You must also provide an uninstaller for your application.

WebClient Registry Keys and Shortcuts

When WebClient is installed, it creates registry entries based on the input from the user, under the following registry key:

```
HKEY_LOCAL_MACHINE\Software\PSC\WebClient
```

These registry key values specify path settings for accessing WebClient functionality and user-supplied options for running WebClient. You must be aware of these registry entries when you write your application installation. The path settings include two values that you can use to set up shortcuts for your application and a default setting for your application installation directory.

Table 7–1 lists the registry key value names as they appear in the registry and provides a description of the information that appears under Data in the registry.

Table 7–1: Registry Entries Made by WebClient

Registry Key Value Name	Data
ProwciniPath	The complete path, including filename, of the WebClient Initializer executable.
ProwcamPath	The complete path, including filename, of the WebClient Application Manager executable.
ApplicationsPath	Default path on the end user's machine that you can use as a root directory to install your WebClient application. The WebClient installation allows the end user to specify this location. Using this registry value in your application installer eliminates the need to prompt the users for information about where they want the application installed. As a result, this creates a more silent application installation. However, you do not want to use this registry value if there is a possibility that there is not enough space on a drive to install the application.

You can provide a shortcut for the user to launch the application directly from Windows after the application is initially installed. This shortcut executes the WebClient Initializer, passing to it a reference to the application configuration file for the application. You must invoke the application this way to take advantage of the automatic update capabilities of WebClient. When you set up the shortcut in the installer, use the following command syntax to execute WebClient Initializer:

SYNTAX

```
ProwciniPath { "<VendorName>/<ApplicationName>" -s }
```

ProwciniPath

The complete path, including filename, to the WebClient Initializer executable. This pathname is stored by the WebClient installer as the `ProwciniPath` registry key value.

<VendorName>/<ApplicationName>

A unique identifier for an application. The values for *VendorName* and *ApplicationName* match the values you specify in the General tab of the WebClient Application Assembler when you define your application. WebClient Initializer uses these values to indirectly locate the application configuration file.

CAUTION: When you change the location of your application configuration file, your users shortcut will no longer work. Therefore, tell your users to go to the same Web page they used the first time they ran the application to automatically update the location of the application configuration file associated with this identifier. After the user updates the location of the application configuration file, the shortcut will work.

The WebClient installer also installs a shortcut for running the WebClient Application Manager. You can create a separate shortcut that runs the WebClient Application Manager with your application highlighted. To set up this application-dedicated shortcut, use the following command syntax:

SYNTAX

```
ProwcamPath "<VendorName>/<ApplicationName>"
```

ProwcamPath

The complete path, including filename, to the WebClient Application Manager executable. This pathname is stored by the WebClient installer as the `ProwcamPath` registry key value.

<VendorName>/<ApplicationName>

A unique identifier for an application. The values for *VendorName* and *ApplicationName* match the values you specify in the General tab of the WebClient Application Assembler when you define your application. WebClient Application Manager uses these values to access information about the WebClient Application.

Application Installer Registry Key Entries for Vendor and Application

The installer is responsible for setting the Application Directory. The installer either gets the Application Directory information from the ApplicationsPath key set by WebClient, or it overrides this value by prompting users to indicate where they want the application installed.

If the installer runs successfully, it must store the ApplicationInstallVersion so WebClient initializer knows that it has run and does not need to run again.

To register this information, your installation program must create the following registry key and registry key values for this key:

```
HKEY_LOCAL_MACHINE\Software\<VendorName>\<ApplicationName>\
```

Where <*VendorName*> and <*ApplicationName*> match the values you enter in the General tab of the WebClient Application Assembler when you define your application.

Table 7–2 lists the registry key value names as they appear in the registry and provides a description of the information that appears under Data in the registry.

Table 7–2: Vendor and Application Name Registry Key Values

Registry Key Value Name	Data
`ApplicationDirectory`	Directory where the user installed the application. This value can be derived from ApplicationsPath.
`ApplicationInstallVersion`	Application install version identifier. You must set this value so that WebClient Initializer can determine whether or not your external installer ran successfully. This version must match the version entered in the Options Tab of the WebClient Application Assembler for the WebEnabled Install. For more information, see the "Options Tab" section.

Application Installer Registry Key Entries for the Configuration File

The ProwcappLocator registry keys are required if the installer sets up a shortcut for the application. This information should match what you enter in the WebClient Application Assembler's Configuration File Locator Definition dialog box. For more information, see the "General Tab" section. Your installation program must create the following registry key and registry key values defining the server for the application configuration file under this key:

```
HKEY_LOCAL_MACHINE\Software\<VendorName>\<ApplicationName>\ProwcappLocator
```

Table 7–3 lists the registry key values as they appear in the registry and provides a description of the information that appears under Data in the registry.

Table 7–3: Configuration File Registry Key Entries

Registry Key Value Name	Data
Type	Indicates the type of server on which the configuration file is located. The type must be InternetServer.
URL	Well formed URL used to identify where the configuration file is located. The URL must begin with **HTTP://**, **HTTPS://**, or **File://**.
Filename	Filename of the application CAB file containing the application configuration file (`.prowcapc`).
EndUserDescription	Information displayed to your end users when WebClient requests authentication information. For more information see Chapter 5, "Designing Security."
Prompt	Indicates whether WebClient should prompt the user for authentication information for the server before downloading the application configuration file. This value can be either Yes or No. If Yes, WebClient prompts the user for authentication information.
DisablePersistentCache	Indicates whether or not a users can save authentication information in a persistent cache on their machines. This value can be Yes or No. If Yes, the information cannot be saved. If No, the information can be saved.

Application Installer Registry Key Entries for WebClient Applications

Your installation program must create a registry key value that is used by the WebClient Application Manager under the following key:

```
HKEY_LOCAL_MACHINE\Software\WebClient\Applications\
```

Table 7–4 lists the registry key value name as it appear in the registry and provides a description of the information that appears under Data in the registry.

Table 7–4: WebClient Applications Registry Key Value

Registry Key Value Name	Data
<Vendor Name>/<Application Name>	" "

NOTE: There must be a space between the two quotes.

7.1.3 Using a Combination of IntelliStream and an External Installer

You can use a combination of IntelliStream and an external installer for your application. Progress Software recommends that if you use a combination, you should use an external installer for the initial installation and IntelliStream for updates. If you do use a combination, your installer needs to provide the following registry key entries for the components and the install version of your application.

Application Installer Registry Key Entries for Components

If you use an external installer for the initial installation of your components, but you want to use IntelliStream to update those components, your installer must store the name of each component it installs in addition to the other tasks required by your external installer. For more information see the "External Installer" section. Therefore, your installation program must create the following registry key and registry key values for each component under this key:

```
HKEY_LOCAL_MACHINE\Software\<VendorName>\ApplicationName>\Components
```

Component Name is the identifier you specified for the component in the WebClient Application Assembler Component Definition dialog box. Table 7–5 lists the registry key value name as it appears in the registry and provides a description of the information that appears under Data in the registry.

Table 7–5: Component Registry Key Entry

Registry Key Value Name	Data
<ComponentName>	" "

NOTE: There must be a space between the two quotes.

Providing an Uninstall

You must provide an uninstall for your application when you use a combination of IntelliStream and an external installer. Your uninstall must remove all parts of the application installed and all registry entries made by the external installer. However, before performing these tasks, the uninstall must first run WebClient Initializer to remove any updates and registry entries made by IntelliStream. To call WebClient Initializer, use the following command syntax:

SYNTAX

```
ProwciniPath { "<VendorName>/<ApplicationName>"-u }
```

ProwciniPath

The complete path, including filename, to the WebClient Initializer executable. This pathname is stored by the WebClient installer as the `ProwciniPath` registry key value.

<VendorName>/<ApplicationName>

A unique identifier for an application. The values for *VendorName* and *ApplicationName* match the values you specify in the General tab of the WebClient Application Assembler when you define your application. WebClient Initializer uses these values to locate information about the WebClient application to perform the application uninstall.

7.2 Running the WebClient Application Assembler

The WebClient Application Assembler is a Progress PRO*Tool and is part of your Progress development environment. Before you run the WebClient Application Assembler, you should read the design chapters in this guide and make your decisions about how you want to deliver and install the application on your end users' machines. When you run the WebClient Application Assembler, you are presented with a number of screens in which you provide information about your application install based on the design decisions you make.

Input Files Required for the WebClient Application Assembler

Before you run the WebClient Application Assembler to define your application and its components, You must have a root directory for your application and it must contain:

- All files that comprise the application (`.r` files, `.pl` files, image files, and so on).
- All files that require special install processing (System Tasks) for your application. This includes third-party files such as ActiveX controls, system `.dll` files, image files for shortcuts, and an initialization file.

In addition to the above items, the following must be true:

- All files placed in the components must reside under the Application Root Directory: the directory you specify in the General Tab of the WebClient Application Assembler.
- The application's directory structure under the Application Root Directory on the development machine must be the same as the directory structure you want to use on the end users' machine under the directory where the application is installed.

Follow these steps to run the WebClient Application Assembler:

1 ♦ Choose **PRO*Tools** from the Tools menu.

The PRO*Tools menu bar opens.

2 ♦ From the menu bar, choose WebClient Application Assembler.

The WebClient Assembler, shown in Figure 7–1, opens.

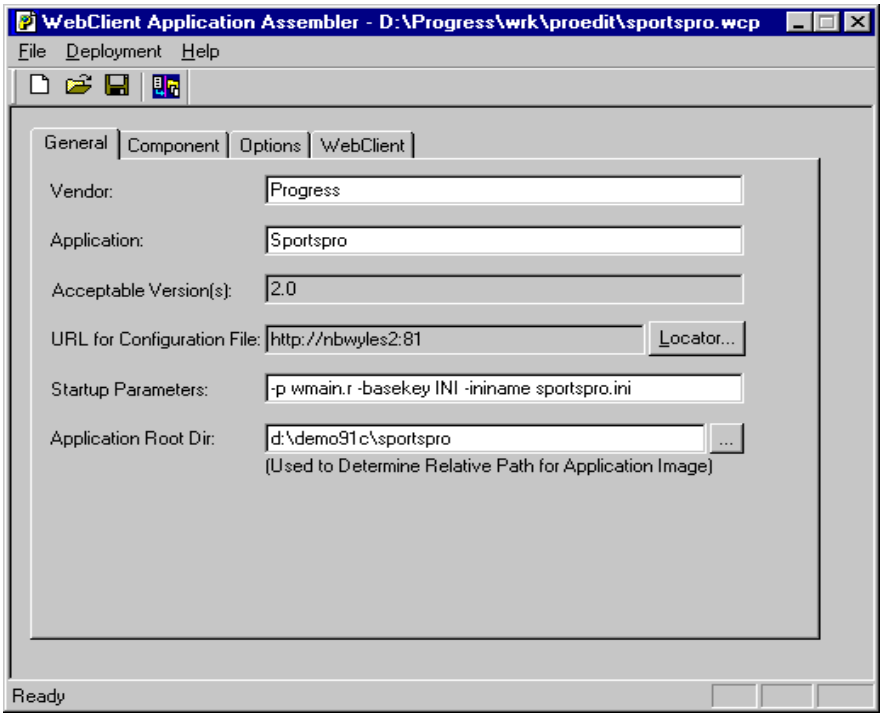

Figure 7–1: WebClient Application Assembler

Using the WebClient Application Assembler, you can perform the following tasks:

- Create a new project file
- Modify an existing project file
- Generate all files necessary for installing and updating a WebClient application

The WebClient Application Assembler includes the following tabs that you use to provide information about your application, and to generate your deployment files, including the application configuration file:

- General
- Component
- Options
- WebClient

This section describes the functionality of the WebClient Application Assembler. For detailed information about the options in the tabs and dialog boxes, see the online Help system.

7.2.1 General Tab

You use the General tab, shown in Figure 7–2, to provide basic information about the application, including startup information. Every value, except for the Application Root Directory, applies to every application regardless of whether you use IntelliStream or an external installer to install your application.

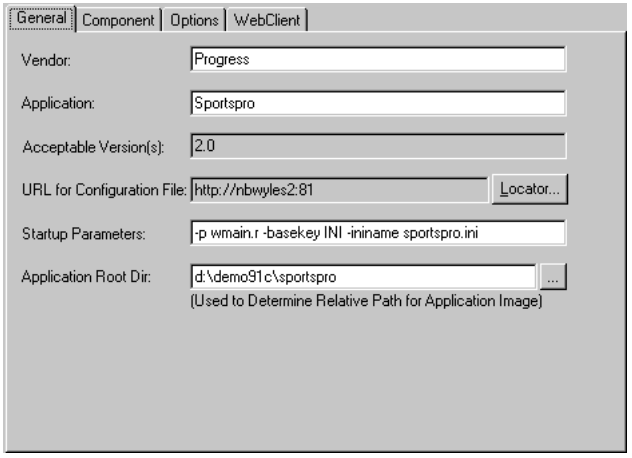

Figure 7–2: WebClient Application Assembler General Tab

From the General tab, you use the Locator button to open the Configuration File Locator Definition dialog box, shown in Figure 7–3. You use this dialog box to specify the InternetServer for the application configuration file and to define authentication options for that server.

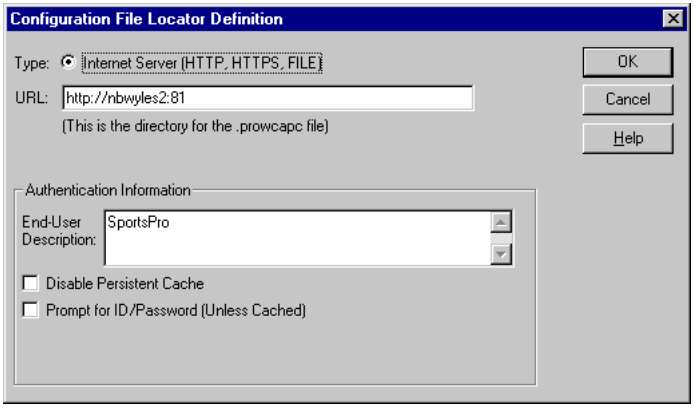

Figure 7–3: Configuration File Locator Definition Dialog Box

7.2.2 Component Tab

You use the Component tab, shown in Figure 7–4, to add, remove, or change components for your application. You only use this tab if you use IntelliStream to install or apply updates for your application.

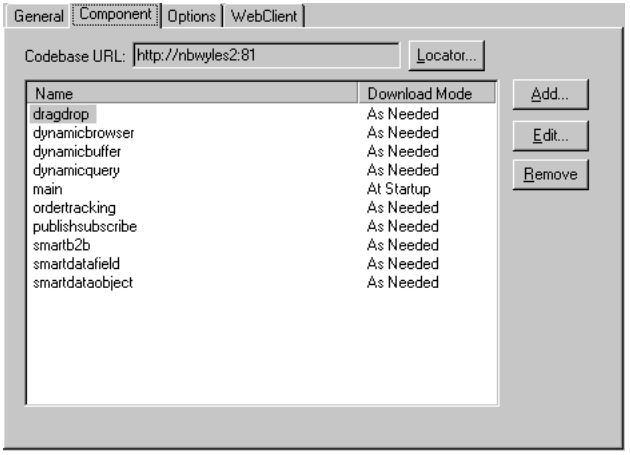

Figure 7–4: WebClient Application Assembler Component Tab

From the Component tab, you open the Codebase Locator Definition dialog box, shown in Figure 7–5, to specify the server of your application components and component updates (`.CAB`) files and the Component Definition dialog box, shown in Figure 7–6, to define your component files and specify the following component types: At Startup, Ask User at Startup, As Needed.

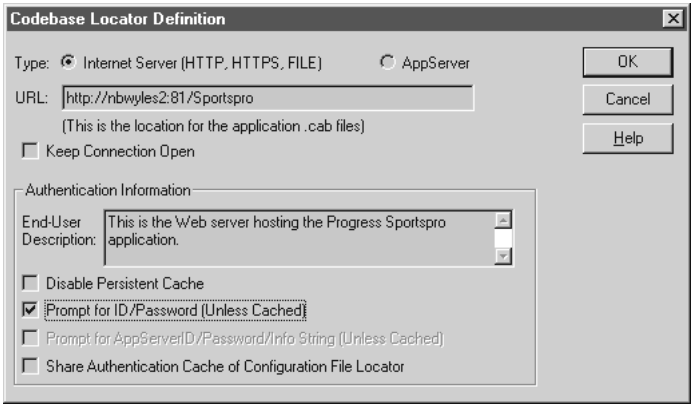

Figure 7–5: Codebase Locator Definition Dialog Box

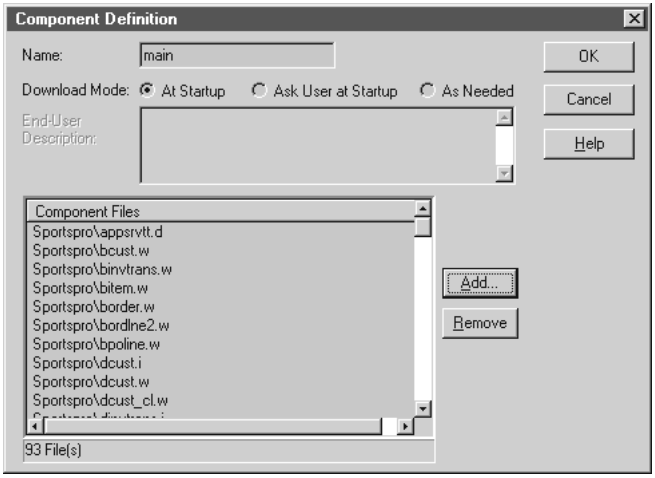

Figure 7–6: Component Definition Dialog Box

7.2.3 Options Tab

You use the Options tab, shown in Figure 7–7, to specify options for the application installation. You use this tab to indicate whether you are using an external installer, to identify any System Tasks you want IntelliStream to perform, and to specify a 4GL install procedure that you want to use for the installation.

From the Options Tab, you choose the System Tasks Definition button to open the System Tasks Definition dialog box.

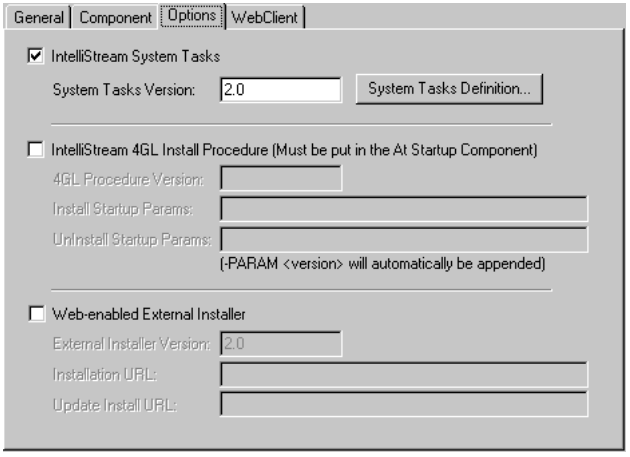

Figure 7–7: WebClient Application Assembler Options Tab

System Tasks Definitions

You use the System Tasks Definition dialog box to define the following system tasks:

- Run ini2reg.exe
- Add shortcuts on the desktop
- Install system or common files
- Register application-specific files, such as the following ActiveX controls provided by Progress: CSCombo, CSSpin, and PSTimer

All files specified to perform system tasks are automatically added to your At Startup component.

Progress 4GL Install Procedure

You can use a 4GLinstall procedure to perform installation tasks not available with IntelliStream System Tasks. The 4GL install procedure runs as the last step of an IntelliStream install to complete any application-specific install tasks that IntelliStream did not handle. You must place this procedure, and any procedures that it calls, in the At Startup component.

If you use a 4GL install procedure, WebClient passes "*<CurrentVersion>, <NewVersion>"* as the value of the `-param` parameter to the procedure. As a result, when Progress runs the 4GL install procedure, the `-param` parameter is added to the startup parameters specified in the Install Startup Params field in the Options tab, shown in Figure 7–7.

For example, if the last 4GL install procedure was version 4 of the installer, and the next installer is version 5, the `-param` value is now “4,5”.

When the 4GL installer runs for the first time, there is no version for the current installer, as a result, `-param` has a value of *",<NewVersion>"*. For example, if the 4GL installer is being installed for the first time, and the version for this installer is V1, the `-param` parameter value is ",V1".

NOTE: If you use a 4GLinstall procedure to perform installation tasks, you can also provide an uninstall procedure.

Web-enabled External Installer

Use these options to provide information about your external installer. For more detailed information about this option, see the “External Installer” section.

NOTE: If you use an external installer, you must provide an uninstall program. You do not need to identify the uninstall program using the WebClient Application Assembler. Your installer needs to make the uninstall program available to your users.

7.2.4 WebClient Tab

You use the WebClient tab, shown in Figure 7–8, to provide information about installing WebClient for your application.

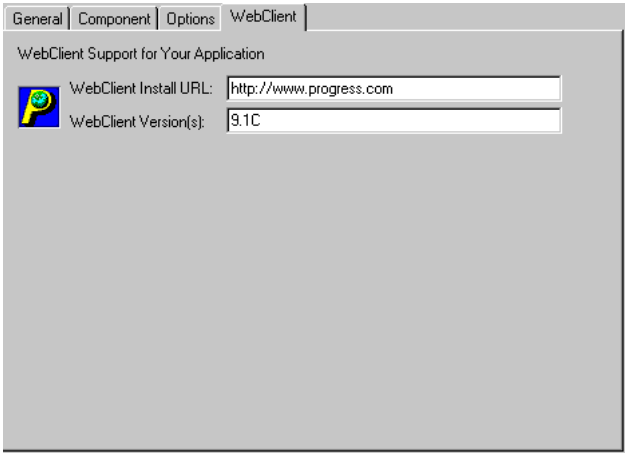

Figure 7–8: WebClient Application Assembler WebClient Tab

7.3 Files Generated by the WebClient Application Assembler

After you have defined the installation requirements for your application, and you are ready to deploy your application, use the WebClient Application Assembler Generate dialog box to create the deployment files for a version of your application. To access the Generate dialog box, shown in Figure 7–9, choose Generate... from the Deployment drop-down menu or choose the Generate button on the toolbar.

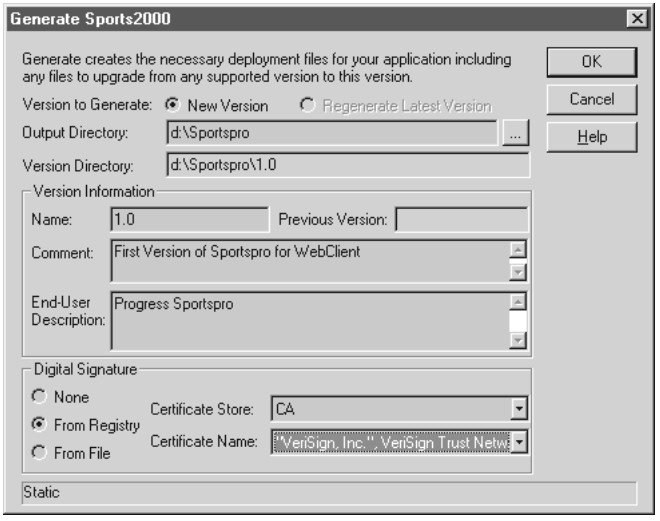

Figure 7–9: WebClient Application Assembler Generate Dialog Box

Whenever you use the Generate dialog box to create a version of your application, the WebClient Application Assembler creates a directory to store the deployment files for the current version of the application. The directory name matches the name you enter in the Version Information Name field in the Generate dialog box and is a subdirectory of the Output Directory. The name for this version directory appears in the Version Directory field of the Generate dialog box.

When you regenerate a version of the application, the WebClient Application Assembler first deletes all the files in the version directory and then regenerates the deployment files.

CAUTION: You must not regenerate a version of an application that you have already deployed.

When you generate a version of your application, the WebClient Application Assembler validates and updates the information about your application and creates the following files:

- **Application configuration file** — Generated application definition file that contains deployment information for your WebClient application. The configuration filename is derived from the name of the project file and has an extension of `.prowcapp`. The WebClient Application Assembler places this file in a `.CAB` file that has an extension of `.prowcapc`. For example, if your `.wcp` filename is `AutoInsurance.wcp`, the WebClient Application Assembler places a file named `AutoInsurance.prowcapp` into a `.CAB` file named `AutoInsurance.prowcapc`.

If you select the Digitally Sign Files option in the Generate dialog box when you generate your deployment files, the `.prowcapc` file and all the component files are signed with a digital certificate.

- **WebClient version information file (`.wcv`)** — Generated file that contains version information for all the files within the application including `.r` files within each `.pl` file. A `.wcv` file is generated for each version and saved in the version directory.

  NOTE: This file is not deployed.

- **Application component files (`.CAB`)** — Generated files that contain the files in an application component. The WebClient Application Assembler generates one `.CAB` file for each component and saves it in the version directory. The root of the `.CAB` filename is the component name you specify in the Component dialog box.

- **Application components update files (`.CAB`)** — Generated files that contain the files necessary to update the application components. The WebClient Application Assembler generates one `.CAB` file for each component that needs updating from the previous version and is saved in the version directory.

  Each `.CAB` file contains the changed files and a `.wcm` file that provides instructions for applying the changes to update the application. Each `.CAB` file has a unique name based on the component name, the previous version, and the new version.

NOTE: If you use an external installer, you must prepare this install image yourself.

7.4 Hosting the Application on a Server

After you use the WebClient Application Assembler to generate your deployment files, you must place these files on your deployment server so that WebClient can access the files.

You have a number of options when deciding on the type of server you want to use to host your application. You can use an Internet-based server: a Web server, a file server, or you can use a Progress AppServer. Each server offers different advantages and supports different features. When deciding on which server to use for hosting your application you must consider the following:

- Where you want to host your application configuration file
- Where you want to host your application code (`.CAB`) files
- Optionally, where you want to host your external installer

For more information about design considerations to help you make these decisions, see Chapter 4, "Designing the Deployment Configuration."

This section also provides information about:

- Hosting the configuration file
- Hosting the application code
- Hosting an external installer

CAUTION: For all server types, when copying files from your development machine to another machine, you must use a binary copy for all files.

7.4.1 Hosting the Configuration File

When you use the WebClient Application Assembler to define your application, you use the Configuration File Locator Definition dialog box to specify a URL that points to the directory on a server where the configuration file resides. Place the application configuration file in the directory specified by the URL in the WebClient Application Assembler's Configuration File Locator Definition dialog box.

NOTE: The application configuration file cannot be hosted on an AppServer.

The first time the configuration file is downloaded to the end user's machine, its location is specified in the `bootstrap.htm` file. For more information about this `bootstrap.htm` file, see the "Bootstrapping a WebClient Application" section. When downloaded, the configuration file is cached on the user's machine.

Although the application configuration file is placed in the version directory when generated, you should place the application configuration file in a version-independent directory when you deploy the file. You must always deploy the application configuration file to this same directory, regardless of the version number.

CAUTION: If you move the configuration file, you must change all references to the location of this file. This includes the references in the `bootstrap.htm` and `webclient.htm` files, and the URL you specified in the Configuration File Locator Definition dialog box of the WebClient Application Assembler.You must then use the WebClient Application Assembler to generate a new version of the application configuration file. Also, if you provided a shortcut for running the application, You must tell your users to go to the installation Web page they used initially to run the bootstrap process to automatically update the shortcut.

7.4.2 Hosting Your Application Component Files

The Codebase Locator specifies the location of the component (`.CAB`) and component update files for your application. When hosting an application using IntelliStream, you can host the component files on a Web server, AppServer, or file server.

In addition to specifying the server type using the WebClient Application Assembler Codebase Locator Definition dialog box, you also specify a URL that enables WebClient to access the component and component update files. For more information about specifying a Codebase Locator, see Chapter 4, "Designing the Deployment Configuration."

You should copy the version directory and its contents, including all its subdirectories, to the location that you specified in the Codebase Locator Definition dialog box. The application version directory on the server machine must have the same name as the version directory in the WebClient Application Assembler Generate dialog box

When hosting your application, you can use:

- **An Appserver** — The version directory for the application must be relative to the AppServers' working directory.
- **A Web server —** The version directory for the application must be relative to the root directory of your Web server.
- **A File server** — The version directory for the application should be a subdirectory or the directory specified in the file URL.

7.4.3 Hosting Your External Installer

If you use an external installer for installing or updating your application, you must do the following to deploy the external installer:

- If you use an external installer to install your application, you must copy the installer to the location you specified in the Install URL field of the WebClient Application Assembler's Options tab. See the "Options Tab" section for more information.
- If you use an external installer to update your application, you must copy the installer to the location you specified in the Update URL field of the WebClient Application Assembler's Options tab. See the "Options Tab" section for more information.

7.4.4 Hosting on a Web Server

If you deploy your application component files or external installer on a Web server, there are a number of other tasks that you must perform and considerations you must make. The following section provides information about:

- Case sensitivity on UNIX Web servers
- Executable directories
- Authentication options on Microsoft Internet Information Server (IIS)
- Configuring MIME types for your Web server

Case Sensitivity on UNIX Web Servers

Web servers that use the UNIX operating system, such as Apache, are case sensitive. Therefore, the letter case of installation filenames on the Web server and those in your application configuration file must match or a run-time error will occur.

Executable Directories

Do not place a Web installer in an executable directory on your Web server. Otherwise, when a user's browser accesses the installer executable, the installer attempts to execute on the Web server instead of downloading and executing on the user's system. This results in a server run-time error.

You can prevent this on an iPlanet Web Server or Netscape Enterprise Server by setting the Web server to download all executable files referenced from a browser instead of executing them. For more information, visit the iPlanet Web site to read the article on setting your Web server to let users download executable files. Otherwise, see your Web server documentation for information on how to plan the content of your directories (or to create virtual directories) so they do not allow local execution of files referenced by the user's browser.

Authentication Options on Microsoft Internet Information Server (IIS)

If you are configuring Internet Information Server, select only the Basic (Clear Text) authentication option in the WWW Services Properties for the Web server. If the Windows NT Challenge/Response authentication option is selected, end users who are not running Internet Explorer receive an authentication dialog box that they cannot satisfy. This prevents them from starting an installation. As a result, end users can never run their application over the Web.

NOTE: Basic authentication uses a well known encryption algorithm for user names and passwords that provides minimal security.

Configuring MIME Types for Your Web Server

When the user attempts to access the application, the Web page that you provide references the configuration file for the application. Before your user's browser can open the configuration file using WebClient Initializer, it must recognize a MIME type for the configuration file. To accomplish this, you must configure your Web server to associate the configuration file extension with its MIME type.

The method used for configuring your Web server depends on the type of Web server you use. For each Web server, however, you must specify the extension and MIME type for the configuration file types, as follows:

- **MIME types**— `application/progress-wcappcab`
- **File type extension** — `prowcapc`

The following sections provide instructions for configuring the MIME type on the Progress-supported Web servers.

NOTE: If an Internet Service Provider (ISP) hosts your Web installation, you must provide it with this configuration file MIME type information required to configure the Web server properly.

Apache Web Server

To register the MIME types for all directories on the server, do one of the following:

- Add the following line to the `srm.conf` file:

```
addtype application/progress-wcappcab prowcapc
```

- Add the following line to the `mime.types` file:

```
application/progress-wcappcab prowcapc
```

If you are using a dedicated Web server, rehash/recycle to register this information.

iPlanet Web Server/Netscape Enterprise Server

On iPlanet/Netscape, follow these steps to register MIME types:

1 ♦ Open the Administration Server.

2 ♦ Select a server and choose the Manage button.

3 ♦ Choose Preferences→ Mime Types.

4 ♦ Type **application/progress-wcappcab** for Content-Type.

5 ♦ Type **prowcapc** for File Suffix.

6 ♦ Choose the New Type button to register the new MIME type on the Web server.

7 ♦ Complete the Administration Server steps to save and apply your changes so the Web server recognizes the new MIME type.

Microsoft Internet Information Server (IIS)

The IIS server stores its MIME types in the Windows registry. You register the MIME type for the application configuration file using the **iis_wc.reg** registry file.

You can find this file in the following location on the Progress installation CD.:

```
webinstall\webclient\web_image
```

where *Webinstall* is the directory where you install your WebClient install images.

To merge the MIME type entries from this file into the registry and configure the Web server, follow these steps:

1 ♦ Copy the **iis_wc.reg** file to your Web server.

2 ♦ To register the MIME type, do one of the following on the Web server:

- Select the **iis_wc.reg** in the Windows Explorer, click the right mouse button, and select Merge. This imports the contents of the file into the registry.
- Double-click the file in the Windows Explorer. This imports the contents of the file into the registry.

3 ♦ Restart your Web server to register these changes.

7.5 Installing WebClient

You must make an installation available to install WebClient on the user's machine. Progress provides a default InstallShield image that you can customize to install WebClient. You can also refer the user to or provide a link to a page on the Progress Web site that the user can use to run a WebClient installation.

Your Progress installation CD includes the following WebClient install images:

- A stand-alone InstallShield image that you can use to create an installation CD-ROM
- A One-Click Install image that you can use to host a Web-enabled installation

For more information about copying theses install images to your system, see the "Accessing the WebClient Installation Files" section.

For a Web-enabled install, you can set up a WebClient installation Web page to be one of the following:

- A Progress-supplied Web page (`webclient.htm` file). You can customize this file to start your application installation after WebClient is installed.
- An application-specific Web page of your own design. The logic for this page should run the WebClient One-Click Install directly, rather than bringing up the Progress default WebClient One-Click Install Web page (which requires the user to click another button). To do this, you can copy the logic from the `webclient.htm` file to your own Web page.

Though you cannot customize the CD-ROM image, Progress Software strongly recommends that you customize the One-Click Install. For more information, about customizing the WebClient installation, see the "Customizing the WebClient Installation" section.

NOTE: Whatever Web page you provide for the user to access the WebClient installer, you must specify the URL for this page in both the `bootstrap.htm` file and in the WebClient Install URL field of the WebClient Application Assembler's WebClient tab.

7.5.1 Accessing the WebClient Installation Files

The two prebuilt WebClient InstallShield images are ready-to-deploy WebClient installers that you can use to install WebClient on users' machines. One image is used for a CD-ROM install and one image is used for a Web-enabled install.

To access these install images, insert your Progress installation CD in your CD-ROM drive and open the root level. From there, you can find the images as follows:

- CD-ROM WebClient installer image in the following directory:

  ```
  webinstall\webclient\win_image
  ```

  To create a CD-ROM for installing WebClient, copy the `win_image` directory contents to the CD that you want to distribute.

- One-Click Install WebClient installer image in the following directory:

  ```
  webinstall\webclient\web_image
  ```

  To provide a Web installer for installing WebClient over the Web, copy the directory contents to an appropriate location on your Web site.

  Although you can deploy this installer without customization, Progress Software recommends that you customize it to provide installation of your WebClient application along with WebClient. For more information, see the "Customizing the WebClient Installation" section.

  NOTE: If you provide a prebuilt WebClient installer without customization, you must provide your end users with instruction to install your WebClient application as a separate installation step.

7.5.2 Customizing the WebClient Installation

To customize a Web-enabled WebClient installer, you modify certain lines in the default One-Click Install Web page to run WebClient Initializer or to invoke your application installer directly if you are using an external installer. Running WebClient Initializer leads your user through the next installation step to automatically install your WebClient application.

CAUTION: Any modifications that you make to the WebClient One-Click Install **must not** interfere with the original operation of the installer. Your modifications can only add functionality to the operations that it performs by default or change the appearance of the page. Any other changes can lead to a WebClient installation with unpredictable results.

You can find a copy of the default WebClient Web page (`webclient.htm`) in the following location on your Progress installation CD:

```
webinstall\webclient\web_image\webclient.htm
```

To implement one of the supported customizations, you must modify the following `webclient.htm` fragment. The portions of the fragment you must change appear in bold:

```
function startInstall(){
    // TODO: To run the WebClient Initializer after the WebClient install completes, you
    // must provide the location (including the file name) of your prowcapc file. To do this,
    //      remove the "//" at the beginning of the last line of this comment (to uncomment it).
    // Then customize that line by replacing <URL...> with the location and name of your
    // prowcapc file as in this example:
    //
    //       ether.SetProperty("ProwcappURL",
"http://server1/webclient/app_installer/yourapp.prowcapc");
    //
    // ether.SetProperty("ProwcappURL", "<URL to your .prowcapc file>");

    // OR:   If you are using a One-Click application install (implemented using InstallShield
    // V6 or later) you can run it directly after the WebClient install completes by
    //      providing the location (directory only) of your installer image plus the location
    //       (including the file name) of your prowcapc file. To do this, remove the "//"
    //       at the beginning of the last 2 lines of this comment (to uncomment them).
    //       Then customize those lines by replacing <URL...> with the directory
    //       containing your application installer image and the location and name of your
    //       prowcapc file as in this example:
    //
    //   ether.SetProperty("ApplicationURL", "http://server1/webclient/app_installer");
    //   ether.SetProperty("ProwcappURL",
"http://server1/webclient/app_installer/yourapp.prowcapc");
    //
    // ether.SetProperty("ApplicationURL", "<URL to your One-Click app installer
directory>");
    // ether.SetProperty("ProwcappURL", "<URL to your .prowcapc file>");

 ether.LegacyMode = true;
 ether.Play();
}
```

To run WebClient Initializer after the WebClient install, you might customize the `startInstall()` function to look like this:

```
function startInstall(){
   ether.SetProperty("ProwcappURL",
      "http://NSSWeb:81/webclient/BakeWare_installer/BakeWareV10.prowcapc");
   ether.LegacyMode = true;
   ether.Play();
}
```

This customization invokes the Initializer and passes it the specified `BakeWareV10.prowcapc` file, which will perform the task necessary to install the application.

To directly run your application One-Click Install, if you have one, from the WebClient install, you might customize the `startInstall()` function to look like this:

```
function startInstall(){
   ether.SetProperty("ApplicationURL",
      "http://NSSWeb:81/webclient/BakeWare_installer");
   ether.SetProperty("ProwcappURL",
      "http://NSSWeb:81/webclient/BakeWare_installer/BakeWareV10.prowcapc");
   ether.LegacyMode = true;
   ether.Play();
}
```

This customization first executes your application One-Click Install from the specified `BakeWare_installer` directory on your Web server at host `NSSWeb`. It also passes the location of the application configuration file (the specified `BakeWareV10.prowcapc` file) to the application installer. The application installer can then invoke WebClient Initializer to launch the installed application after completing the application installation.

7.6 Bootstrapping a WebClient Application

Whether you use IntelliStream or an external installer to install your application, your WebClient application is always started by WebClient Initializer. For information about installation options, see Chapter 3, “Designing Your End User’s Experience”.

WebClient Initializer is a program that controls the installation, update, and execution of a WebClient application that can also include installing a different version of WebClient itself.

WebClient Initializer uses information in an application configuration file to control how it starts a WebClient application. The information that you provide in this application configuration file allows WebClient Initializer to identify and verify the status of both WebClient and the application on the user's system, and to start the application.

Starting the Application

You can provide two basic options for the user to start a WebClient application:

- **URL** — This option allows the user to start the application from a Web browser. The specified URL identifies a `bootstrap.htm` file provided by Progress that you must customize to access the WebClient One-Click install image and your application configuration file. This `bootstrap.htm` file starts the process of installing and executing both WebClient and the application that it runs. You can make this URL available to the user in any way appropriate for your Web site. For example, you can have a user enter the URL directly in the locator of their browser, or you can provide a link to this URL on your application Web page.

NOTE: If WebClient is installed from a CD, you should still use the `bootstrap.htm` file to install the application.

- **Shortcut** — This option allows the user to start the application directly from the Windows desktop. If you use IntelliStream, this shortcut can be created automatically using the System Tasks Option in the WebClient Application Assembler. If you use an external installer, your installer must create this shortcut.

How the Bootstrapping Process Works

The prescribed way to bootstrap your WebClient application is to use the `bootstrap.htm` file that Progress provides and that you must customize. The `bootstrap.htm` file is used to initiate the installation process.

For information about customizing this file see the "Customizing the Bootstrap.htm File" section.

This `bootstrap.htm` has the following logical paths:

- If WebClient is not installed, your end user is taken to a Web page to install WebClient.
- If WebClient is installed, WebClient Initializer runs and performs the following tasks:
  - Checks that an acceptable version of WebClient is installed on the user's system and gives the user an opportunity to update the version of the WebClient installation, if necessary.
  - Checks whether the application is installed on the user's system and gives the user an opportunity to install it, if necessary.
  - Checks whether any application updates are required and gives the user an opportunity to update the application, if necessary.
  - Starts the application

Whenever WebClient Initializer runs, it takes your users to the next step of the bootstrapping process. Therefore, any process to which WebClient initializer gives control to perform an installation step (WebClient install and external application install) needs to run WebClient Initializer again for the bootstrapping process to continue and ultimately run the application.

Figure 7–10 illustrates the bootstrapping process and shows the bootstrap process flow depending on WebClient Initializer's verification at the different decision points in the process. This process is transparent to your end user, and depends on whether you used IntelliStream or an external installer and how you customized your WebClient installation to install and start your application.

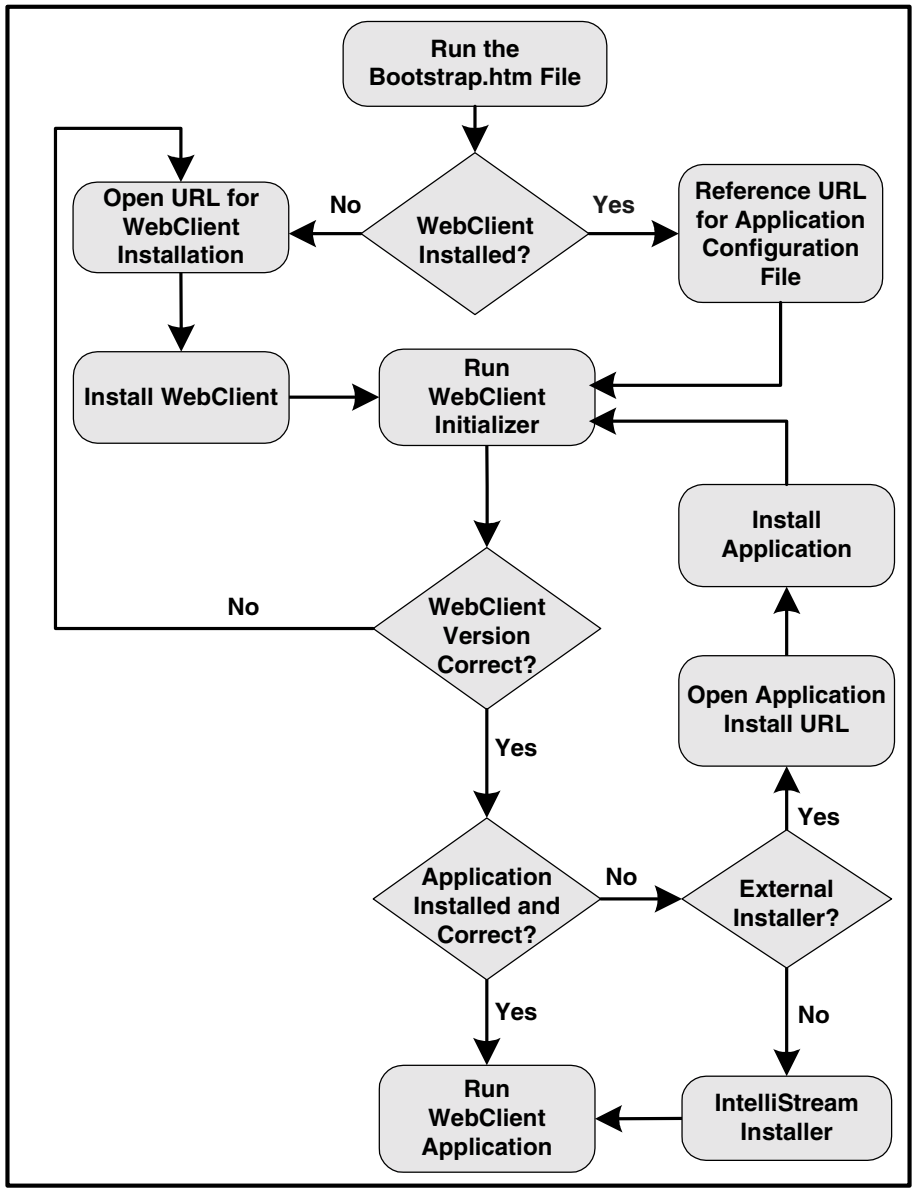

Figure 7–10: Bootstrapping Process

NOTE: During this bootstrapping process, if any URL requires authentication on your Web server, the end user receives a prompt for the appropriate user ID and password.

Bootstrapping with No WebClient Installed

If WebClient is not found on the user's machine, the `bootstrap.htm` file opens a WebClient installation page based on a URL that you specify. This WebClient installation URL is determined by you and must be customized in the bootstrap HTML file. For more information on the WebClient installation page, see the "Installing WebClient" section.

Bootstrapping with Some Version of the WebClient Installed

If any version of WebClient is found on the user's machine, the `bootstrap.htm` file references a URL to the application's configuration file, which runs WebClient Initializer. This configuration file URL is determined by you and must be customized in the `bootstrap.htm` file.

Customizing the Bootstrap.htm File

For any scenario using the `bootstrap.htm` file, you must customize the file provided by Progress. To customize the `bootstrap.htm` file, you must substitute your own URLs for the two URL settings shown in bold:

- **`document.locations.href` —** This setting specifies the URL of the `.CAB` file containing the application configuration file for the application
- **`window.location` —** This setting specifies the URL of the Web page used to access the WebClient installer over the Web. You can use your own customized version of WebClient or the default WebClient installer provided on the Progress Web site.

See the sample `bootstrap.htm` file that follows:

(1 of 2)

```
<html>

<script language="JavaScript">
 installed = false;
 netscape = (navigator.appName.indexOf("Netscape") != -1);

 // See if WebClient is installed
 if (netscape)
 {
   var mimeTypes = navigator.mimeTypes["application/progress-wcappcab"];
   if (mimeTypes)
     installed = true;
 }
</script>

<script language="VBScript">
 ' This is ignored by Netscape

 On Error Resume Next

 set ctrl = CreateObject("ProWCVer.ComObj.1")
 if isObject(ctrl) then
   installed = true
 end if
</script>

<script language="JavaScript">
 if (installed)
 {
   // Reference a prowcapp file to invoke prowcini.exe
   // NOTE: MODIFY THIS URL TO POINT TO YOUR .prowcapc FILE.
   document.location.href =
""http://NSSWeb:81/webclient/BakeWare_install/BakeWareV10.prowcapc";
 }
 else
 {
   // Start install process for Web Client
   // NOTE: MODIFY THIS URL TO POINT TO THE WebClient Install page
   window.location = "http://NSSWeb:81/webclient/webclient.htm";
 }

</script>

<body>
```

(2 of 2)

```
<!--// NOTE: THIS TEXT CAN BE MODIFIED //-->
<p align="center"><font face="Verdana, Arial, Helvetica" size="3">This
Application is Powered by Progress</tr>
<p align="center"><font face="Verdana, Arial, Helvetica" size="3">Thank you
for using our application!</tr>

<!--// NOTE: THIS URL CAN BE CHANGED TO POINT TO AN APPLICATION SPECIFIC JPG
    OR OTHER APPLICATION SPECIFIC HTML CAN BE PLACED HERE.//-->
<p align="center"><img SRC="graphics/WCSplash.jpg" height=250 width=350>

</body>
</html>
```

Matching Letter Case in URL Pathnames

If you are using a UNIX Web server, URL pathnames that you provide must match the letter case of these pathnames on the Web server. If the letter case of these pathnames and filenames do not match, your end users will receive a message that the files cannot be found when they attempt to install and run WebClient and your application.

Checking for an Acceptable Version of WebClient

The first verification WebClient Initializer does is to determine that an acceptable version of WebClient is installed on your user's system. The Initializer compares the installed versions of WebClient with the acceptable versions listed in the application configuration file. If there is an acceptable but older version, the Initializer gives the user an opportunity to install the preferred version or run with an existing version. If there is no older but acceptable version, the Initializer gives the user an opportunity to install the preferred version.

For any case where the user chooses to install the preferred version of WebClient, the Initializer opens a Web browser on the user's system to the URL of the WebClient installation specified in the application configuration file and exits, allowing the user to continue with the installation over the Web.

If the user chooses to go with an acceptable version already installed, the Initializer proceeds with the next validation. Otherwise, the Initializer exits without proceeding.

Whatever WebClient version the user chooses to run, the user can activate a Don't ask me this again toggle box to prevent notification of another update until you specify a different preferred version in the configuration file.

If WebClient Initializer finds that the application specified in the configuration file is not installed, one of the following happens:

- If you use IntelliStream, the Initializer starts WebClient to install and start the application.
- If you use an external installer, the Initializer opens your user's Web browser to the URL of the application install page and exits.

Checking for Required Application Updates

If WebClient Initializer finds that the installed version of the application is out of date, it gives the user an opportunity to update it. For more information, see the "WebClient Application Updates" section.

7.7 WebClient Application Updates

When you make changes to your application, all you need to do is use the WebClient Application Assembler Generate dialog box to generate a new version of the application. This creates a new configuration file and possibly new component files and component update files and places them in a new version subdirectory.

On the end user's machine, WebClient Initializer compares the installed version of the application with the acceptable version in the application configuration file and then prompts the end user as follows:

- If the installed version is acceptable but older than the current version, the Initializer gives the user an opportunity to update to the current (preferred) version or run with the existing version.
- If the installed version is not acceptable, the Initializer gives the user an opportunity to update to the preferred version.

If you use:

- IntelliStream, and the end user chooses to update, WebClient Initializer starts WebClient to perform the IntelliStream update and run the application.

  Depending on how you implemented your security, the end user might be prompted for authentication information and verification to download the modified files. For more information, see Chapter 5, "Designing Security," and "Application Components," in this chapter.

NOTE: A major benefit of IntelliStream is that only the individual `.r` or other file that have been modified are downloaded during and update. Progress Software strongly recommends that you use the **Generate MD-5** option when you compile your `.r` code. For more information, see Chapter 1, "Overview."

- An external installer to update the application, you can use the same Web page and logic that you use for the initial installation of the application, or you can provide a separate application-specific Web page and logic.

  For any case where the user chooses to update the application to the preferred version, the Initializer opens a Web browser on the user's system using the application update URL specified in the application configuration file and exits. At this point your user can continue with the update over the Web.

8

Your End User's Experience

Before your end user installs WebClient and installs and runs your WebClient application, you must provide them with the information they need to perform the installation and to run the application. After WebClient and the application are installed, you need to provide information about the Progress WebClient Application Manager tool so that they can use this tool under your supervision to view and change information about the WebClient and application installation. This chapter provides you with information about:

- Preparing documentation for your end users
- WebClient Application Manager

8.1 Preparing Documentation for Your End Users

You must provide documentation for your users that includes following information:

- The WebClient installer and the application installer can potentially make changes to system DLL files, therefore, in order to install and run a WebClient application over the Web, one of your end users must have Administrator privileges.
- The end users machine must run either Netscape 4.x (or later) or Microsoft Internet Explorer 4.x (or later) to install WebClient and run the application.
- You should inform users that during WebClient installation they must provide the proxy host and port of any firewall on their system. For more information, see "WebClient Application Manager."
- You should warn users that during the WebClient and application installation they might be told to reboot their machines after the WebClient and application are installed.
- To run your application, users must have Web access. Even though the application is installed on a user's machine, WebClient Initializer checks the application configuration file on your Web site each time the application is run to verify whether there is a new version of WebClient or the application.

  If their browser displays an alert box similar to the one shown in Figure 8–1, users must choose the **Open it** option to install or start the application.

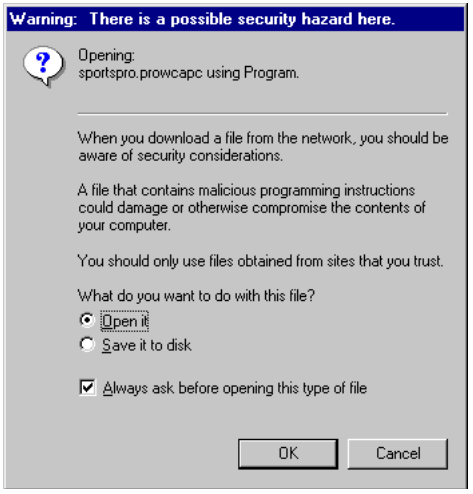

Figure 8–1: Sample Browser File Download Alert Box

Users can clear the Always ask before opening this type of file check box if they do not want to see this alert box the next time they launch the application from their browser.

NOTE: Clearing this check box turns off notification for **all** WebClient applications that the user launches from their browser.

- If Netscape is running on a user's machine when WebClient is installed, and the user subsequently wants to use Netscape to start the application, the user must first stop and restart Netscape, and then start the application. Otherwise, when the file with the `.prowcapc` extension is automatically downloaded to the user's machine, Netscape does not know the associated application (WebClient Initializer) to open the file and displays a dialog box prompting the user to specify the application. If users see this dialog box, they must shut down and restart Netscape to invoke the application.

- When users first install WebClient from your Web site, the browser might require them to download a setup player for the installer. When they download the player, a dialog box prompts them to Grant a certificate that allows the player to be installed. You should advise your users that they must always choose to Grant the certificate.

- You must tell users how to run the WebClient Application Manager to access the WebClient Initializer log file or make any post-installation changes to their application configuration. For more information see the "WebClient Application Manager" section.

- If you use IntelliStream to install your application, you must tell your end users that if they uninstall the WebClient application, they must first uninstall the application and then uninstall WebClient.

- If your application performs HTTPS connections to a Web server that uses Digital Certificates signed by a private CA, you must provide your end users with documentation about your installation package for Digital Certificates. Your documentation should direct the end user to first download and install the WebClient and then immediately install the Root Digital Certificates. Your documentation should also provide your end users with detailed information about how the would update the Root Digital Certificates for the WebClient. Because each public or private PKI vendor has different procedures for installing and using Digital Certificates, you need to provide explicit directions for each PKI vendor you use. For more information, see Chapter 5, "Designing Security."

8.2 WebClient Application Manager

WebClient Application Manager allows your users to configure and manage their use of a WebClient application and to provide information that you can use for troubleshooting. Your end users should only run WebClient Application Manager under your supervision and usually for the following reasons:

- To specify post-installation WebClient startup parameters
- To specify proxy server host and port for the user's firewall
- To access information such as the WebClient Initializer log file and the system registry settings to help you to debug any installation problems.

WebClient Application Manager is installed as part of the WebClient installation. Although your end user might have multiple versions of WebClient installed on their machine, there is always only one version of WebClient Application Manager. This is always the WebClient Application Manager installed with the most current version of WebClient. However, although the WebClient Application Manager is part of most current WebClient Installation, it can be used to display information about all WebClient applications installed on a user's machine.

8.2.1 Using the WebClient Application Manager

The WebClient Application Manager gives end users a way to view and edit information about the WebClient applications installed on their system. The WebClient Application Manager Main window has the following tabs and buttons:

- **WebClient tab** — Lists WebClient versions installed on the user's system and allows the user to modify general options for WebClient execution. See the "WebClient Tab" section.
- **Applications tab** — Lists WebClient applications installed on the user's system by vendor name and application name, and allows the user to review, set, or change execution options for a selected application using the Edit dialog box. For more information see the "Applications Tab" section.
- **Close** — Closes the WebClient Application Manager.
- **Edit** — Opens the Edit dialog box for the selected application. Users can access the Edit button only from the Applications tab.

For more information about the Main window, see the "Main Window" section.

8.2.2 Modes of Execution

Depending on how the user starts WebClient Application Manager, the program starts in one of the following modes:

- **WebClient** — This mode is available from a shortcut created by the WebClient installation. The WebClient installer creates this shortcut by default on the Start menu under the Progress WebClient program folder. When the user uses this shortcut, the Main window opens with the WebClient tab displayed.

- **Application** — You can make this mode available to the end user through a shortcut that you set up during the application installation. When the user uses this shortcut, the Main window opens with the Applications tab and current application selected.

  For more information about setting up the shortcut for this mode, see the "External Installer" section and the"System Tasks" section in Chapter 7, "Deploying an Application."

8.2.3 Main Window

The Main window displays information about WebClient in the WebClient tab and provides a list of available WebClient Applications in the Applications tab.

WebClient Tab

If the user runs WebClient Application Manager in WebClient mode, the Main window opens with the WebClient tab selected, as shown in Figure 8–2. This tab shows a list of all WebClient versions installed on the system along with an indication of the patch level for each one.

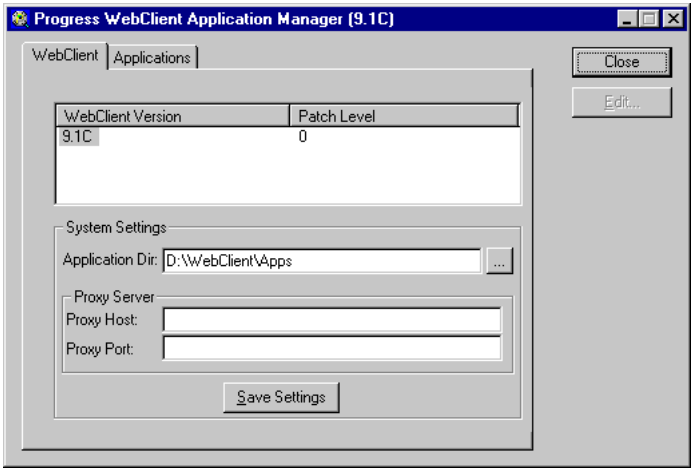

Figure 8–2: WebClient Tab of the WebClient Application Manager

The WebClient tab includes a System Settings group box that has editable fields that display values for the application directory, and the host and port of a proxy server if the user's system must connect to your Web server through an HTTP-based proxy server (firewall). If the user provided the proxy host and port during WebClient installation, these fields display them as the initial values. The user can modify them as follows:

- **Application Directory** — Allows users to change the directory on their system where a WebClient application is installed by default. Users can use the browse button to change this directory. You would have your users change this directory when there is not enough space on the drive the user initially specified for the application installation.

- **Proxy Host** — Allows the user to set or change the name or IP address of the host for the user's firewall. This option appends the –proxyhost startup parameter with the associated host information to the list of startup parameters used to execute WebClient.

- **Proxy Port** — Allows the user to set or change the port for the user's firewall. This option appends the –proxyport startup parameter with the associated port information to the list of startup parameters used to execute WebClient.

To save any changes made on this tab, the user must choose the Save Settings button.

Applications Tab

The Applications tab, shown in Figure 8–3, displays a list of all WebClient applications currently installed on the user's system. Applications are identified using the vendor name and application name.

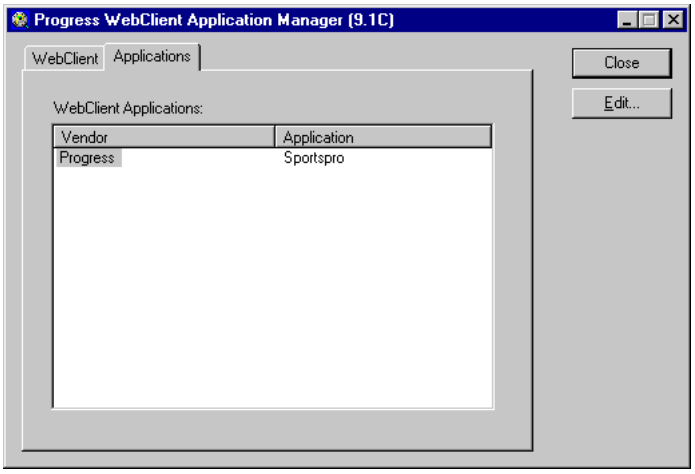

Figure 8–3: Applications Tab of the WebClient Application Manager

To make changes to an application's options or to view the WebClient application log file, select the application in the list and choose the Edit button to open the Edit dialog box, shown in Figure 8–4.

8.2.4 Edit Dialog Box

The Edit dialog box, shown in Figure 8–4, displays information about WebClient applications and allows users to change certain options for the application selected in the Applications tab. The information displayed reflects the information about the last time the application ran.

Applications on the end user's machine might be version 9.1B, for example, or a later version that used the WebClient Application Assembler. The Main window displays information depending on the application selected in the Applications tab.

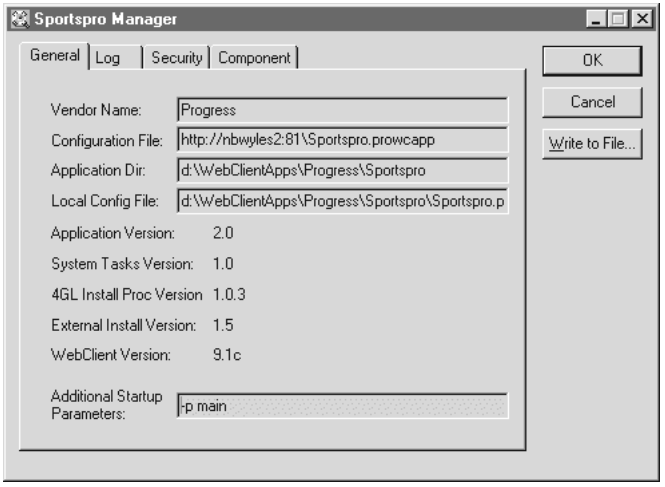

Figure 8–4: Application Manager Edit Dialog Box

The Edit dialog box title reflects the name of the selected application, for example Sportspro Manager, and includes a tabbed folder with multiple tabs. The tabs that appear in the Edit dialog box vary depending on whether or not you use the WebClient Application Assembler to define the application installation. For all applications, the following tabs appear in the Edit dialog box:

- **General** — Lists general information about the selected application, and allows the user to set or change certain execution options. See the "General Tab" section.
- **Log** — Displays the contents of the application log file generated by WebClient and allows the user to delete the contents of the file. See the "Log Tab" section.

For application installations that were defined using the WebClient Application Assembler, the following additional tabs appear in the Edit dialog box:

- **Security** — Displays security information about the selected application. For more information, see the "Security Tab" section.
- **Component** — Displays information about the application components that are installed on your end users system. For more information, see the "Component Tab" section.

From the Edit dialog box, users can choose the following buttons:

- **OK** — Saves any option changes and closes the dialog box
- **Cancel** — Closes the dialog box without saving option changes
- **Write to File** — Copies all the registry values associated with the selected application to a file you specify. This information is used to help you with solving problems that occur when the application runs.

General Tab

The General tab displays mostly read-only-information about the application. However, you can use this tab to have your users add startup parameters for the application. The information displayed reflects the status of the application the last time it was run on this system.

The fields that appear in the General tab vary depending on whether or not you use the WebClient Application Assembler to define the application installation. For example, the General tab shown in Figure 8–5 shows information about a version 9.1C application whose installation was defined using the WebClient Application Assembler, and the General tab shown in Figure 8–6 shows information about a version 9.1B application whose installation was not defined using the WebClient Application Assembler.

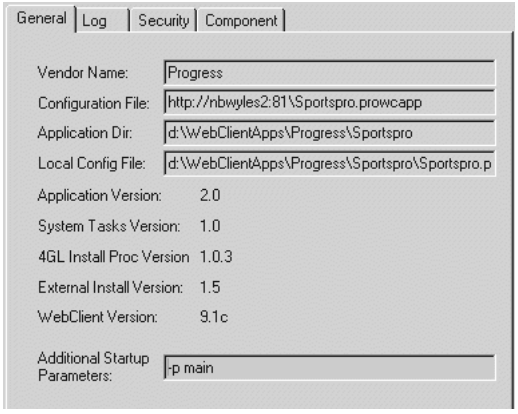

Figure 8–5: General Tab Displaying Information About a 9.1C Application

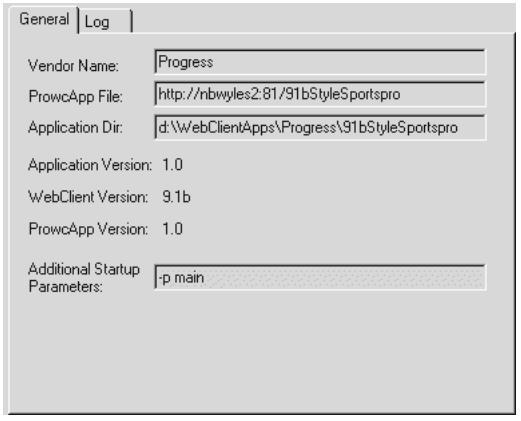

Figure 8–6: General Tab Displaying Information About a 9.1B Application

For all applications, the following fields appear in the General tab:

- **Vendor name** — Name of the application provider.
- **Application Dir** — Local directory where the application is installed and uses as its execution working directory.
- **Application Version** — Current version of the application.
- **WebClient Version** — Version of WebClient used to run the application.
- **Additional Startup Parameters** — Editable field that allows the user to enter any well-formed list of Progress startup parameters using the syntax that you provide them. For more information on the available startup parameters, see the *Progress Startup Command and Parameter Reference*.

  One reason your users might use these parameters is to improve performance specific to this user. WebClient Initializer appends all startup parameters specified in this field to the end of the list of startup parameters already specified.

If the application installation was not defined using the WebClient Application Assembler, the following additional fields appear in the General tab:

- **ProwcApp File** — URL for the configuration file used to install and start the application.
- **ProwcApp Version** — Version number used for the benefit of the ISV.

If the application installation was defined using the WebClient Application Assembler, the following additional fields appear in the General tab:

- **Configuration File** — URL for the configuration file used to install and start the application.
- **Local Config File —** Pathname of the configuration file that is cached on the end user's machine.
- **System Tasks Version —** Current version of system tasks as defined in the WebClient Application Assembler
- **4GL Install Proc Version —** Current version of the 4GL install procedure as defined in the WebClient Application Assembler
- **External Install Version —** Current version of the external installer as defined in the WebClient Application Assembler

CAUTION: WebClient Application Manager does not validate the data that the user enters.

Log Tab

The Log tab, shown in Figure 8–7, displays the contents of the application log file generated by WebClient.

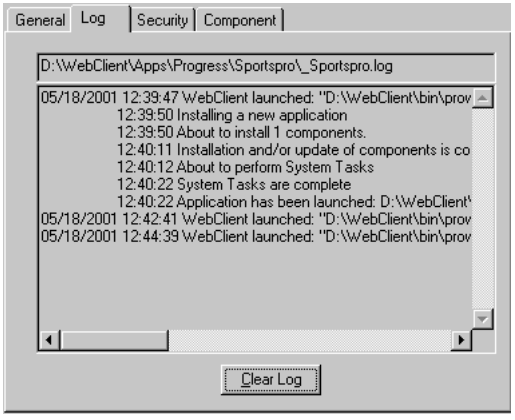

Figure 8–7: Log Tab of the WebClient Application Manager

WebClient writes status messages to a log file for you to troubleshoot problems. The log file is created in the Application Directory you specified.

The path to the log file appears above the file contents box so that the end user can locate the file if you or Progress Technical Support ask for it. The user can delete the application log file by choosing the Clear Log button.

CAUTION: The effect of choosing the Clear Log button is permanent. Choosing the Cancel button **does not** undo deletion of the application log file.

Security Tab

The Security tab, shown in Figure 8–8, displays authentication information about the selected application. This tab appears only when you use the WebClient Application Assembler to define the application installation.

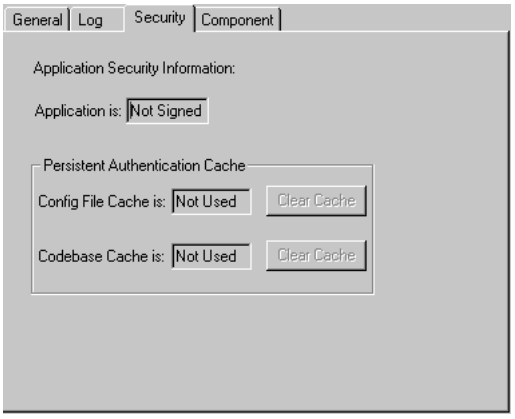

Figure 8–8: Security Tab

Application Security Information — Indicates whether or not the application is signed. The possible values are: **Signed** or **Not Signed**.

Persistent Authentication Cache — Indicates whether or not the Config File Cache and the Codebase Cache are being used. The possible values are:

- **Used** — Authentication information is cached.
- **Not Used** — Authentication information is not cached.

You can use the **Clear Cache** button to clear the registry entries for the associated security cache. This button is not available when the cache is not used.

CAUTION: Clearing the cache is permanent and cannot be undone.

Component Tab

The Component tab, shown in Figure 8–9, provides a list of application components that have been downloaded to the end users machine. This tab appears only when you use the WebClient Application Assembler to define the application installation.

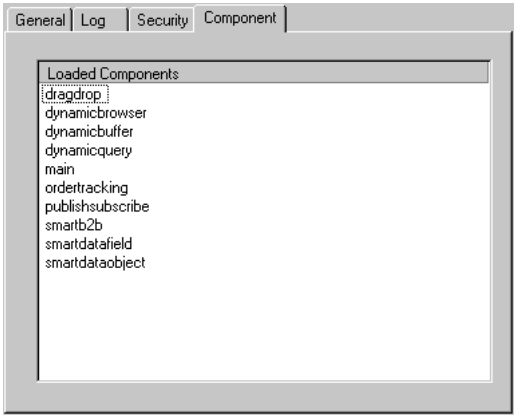

Figure 8–9: Component Tab

A

Deploying the Sample Application With IntelliStream

This appendix tells you how deploy the Progress sample application, SportsPro, with WebClient with IntelliStream.

NOTE: There is no difference between SportsPro in the current version of Progress and Sports2000 in previous versions.

When you run the sample application with IntelliStream technology, only a portion of the application downloads initially, while the rest downloads only as needed.

These instructions use the following symbols:

- `%DLC%` refers to the Progress installation directory on a particular machine. For example, the instructions might mention `%DLC%` on the end user-machine.
- `%ASWORKDIR%` refers to the work directory of a Progress AppServer.
- `%APPROOTDIR%` refers to the root directory of a WebClient application.
- `%OUTDIR%` refers to the directory in which the WebClient application assembler places the files it generates.
- `%MYWEBSERVER%` represents your Web server's domain name or IP address.

NOTE: A copy of the WebClient files that support the sample application reside in the following directory:
`%DLC%\src\samples\webclient\dynappdel`.
A copy of these instructions resides in the following file:
`%DLC%\src\samples\webclient\dynappdel\instructions.txt`.

These instructions cover:

1. Preparing the Web server
2. Setting up the application on the AppServer machine
3. Setting up the cabinet file containing the application configuration file on the Web server machine
4. Installing and running the application on the client machine

A.1 Preparing the Web Server

Preparing the Web server involves:

- Setting up the Java Servlet Engine and the AppServer Internet Adapter
- Setting up MIME types for the application configuration file

NOTE: The application configuration file always has a file extension of `.prowcapp` and always resides in a Microsoft Cabinet file with a file extension of `.prowcapc`. This is true whether the application is signed or unsigned.

A.1.1 Setting Up the Java Servlet Engine and the AppServer Internet Adapter

For instructions on setting up the Java Servlet Engine (JSE) and the AppServer Internet Adapter (AIA) on the Web server, see the "AppServer Internet Adapter" chapter of the *Progress Version 9 Product Update Bulletin* and the Progress Knowledge Base entries on setting up the JSE to work with the AIA. To access these, search the Knowledge Base on "JSE" and "AIA."

A.1.2 Setting Up MIME Types for the Application Configuration File

On the Web server, set up MIME types for the application configuration file. The technique for doing so depends on your particular Web server. For information on setting up MIME types for a particular Web server, see the "Configuring MIME Types for Your Web Server" section of Chapter 7, "Deploying an Application." Also, see the documentation for your Web server.

A.1.3 Setting Up the WebClient Install and Bootstrap Mechanism

Follow these steps to set up the WebClient install and bootstrap mechanism:

1 ♦ Insert the Progress CDROM into the CDROM drive.

2 ♦ Recursively copy the `webinstall\webclient\web_image` directory to the `web_image` subdirectory of your Web server's document root directory.

3 ♦ Edit the `bootstrap.htm` file (which was copied in Step 2) so it references your WebClient install image and your application configuration file as follows:

In the `bootstrap.htm` file, replace the `document.location.href` and `window.location` URLs to match the location of your configuration and `webclient.htm` files, respectively.

For example:

```
document.location.href = "http://pctest/web_image/sportspro.prowcapc";

window.location = "http://pctest/web_image/webclient.htm";
```

4 ♦ Edit the `webclient.htm` file (which was copied in Step 2) as follows:

a) Uncomment the lines that invoke the WebClient initializer. (Keep commented the lines that invoke the One-Click application install directly.)

b) Change *<URL to your .prowcapc file>* to `http://%MYWEBSERVER%/web_image/sportspro.prowcapc`.

For example:

```
ether.SetProperty("ProwcappURL",
"http://pctest/web_image/sportpro.prowcapc");
```

A.2 Setting Up the Application on the AppServer Machine

Follow these steps to set up the application on the AppServer machine:

NOTE: This example assumes that the AppServer machine is the same as the development machine on which you prepare the front-end and back-end application code. For this reason, this machine must be a Windows machine.

1 ♦ Verify that a Progress **development** product has been installed on the AppServer machine and that the person doing the installation, when presented with the Setup Type window, chose the **complete installation**, as opposed to the typical installation or the custom installation.

2 ♦ Verify that a Progress AppServer has been installed and configured on the AppServer machine.

3 ♦ Create the WebClient application root directory, the symbol for which is `%APPROOTDIR%`. The application root directory can reside anywhere, but note its path.

4 ♦ Recursively copy the source-code files of the Sports2000 application from the `%DLC%\src\sports2000\gui` directory `to %ASWORKDIR%.`

NOTE: The `gui` directory contains both back-end and front-end application files. `%ASWORKDIR%` is where the entire application will be compiled, though only the back-end r-code (`.r`) files will be run from this directory.

5 ♦ Create a copy of the Sports2000 database on the AppServer machine. The copy can reside anywhere, but be sure to note its path.

6 ♦ Start a Progress session. Within the Progress session:

a) Connect to the Sports2000 database, entering its path, which you noted in Step 5.

b) Use the PROPATH Editor to add the AppServer work directory to the PROPATH.

c) In the Application Compiler, go to Options→ Compiler, change “Minimize R-code size” to YES, change Generate MD-5 to YES, and click OK. In the main window, click Modify. In the File Specification window, enter the AppServer’s working directory, click OK, and compile the application.

7 ♦ Recursively copy the following directories as specified below, creating the destination subdirectory as necessary:

Recursively Copy This Directory	To This Location
%ASWORKDIR%\gui	%APPROOTDIR%\appl
%DLC%\gui\adm2	%APPROOTDIR%\gui\adm2
%DLC%\jms	%APPROOTDIR%\jms
%DLC%\src\sports2000\images	%APPROOTDIR%\appl\images

8 ♦ Copy the following files as specified below, creating the destination subdirectory as necessary:

Copy This File	To This Location
%DLC%\src\samples\webclient\dynappdel\sportspro.ini	%APPROOTDIR%
%DLC%\src\samples\webclient\dynappdel\appsrvtt.d	%APPROOTDIR%
%DLC%\src\samples\webclient\dynappdel\sportspro.wcp	%APPROOTDIR%
%DLC%\gui\adecomm.pl	%APPROOTDIR%\gui
%DLC%\gui\adeicon.pl	%APPROOTDIR%\gui
%DLC%\gui\adexml.pl	%APPROOTDIR%\gui
%DLC%\gui\adeicon\progress.ico	%APPROOTDIR%
%DLC%\bin\system\csspin32.ocx	%APPROOTDIR%

9 ♦ Use a text editor to modify the `appsrvtt.d` file, which you copied in Step 8, as follows:

a) In `appsrvtt.d`, search for the following string:

```
http://yourservername/aia/Aia?AppService=sports2000partition
```

b) In the string, change *yourservername* to the domain name of the Web server running the AIA. For example, if the domain name of the Web server running the AIA is http://testhost, modify the string to appear as follows:

```
http://testhost/aia/Aia?AppService=sports2000partition
```

c) Save `appsrvtt.d`.

10 ♦ Use the WebClient Application Assembler as follows:

a) From the main menu, select File→ Open, and open `%APPROOTDIR%\sportspro.wcp`.

b) On the General tab, change the path of the application root directory to its actual path on the AppServer machine. The default value is `d:\demo91c\sportspro`.

c) On the same tab, click on the Locator button. The Configuration File Locator Definition window appears. In this window, change the URL to the host name of the server hosting the configuration file, which defaults to `http://`*`yourhostname`*`.`

d) In the Codebase Locator Definition window (main menu→ Deployment→ Locators→ Codebase), change the URL to the host name of the server hosting the codebase, which defaults to http://*yourhostname*.

e) On the Options tab, check the System Tasks toggle box and click the System Tasks Definition button. The System Tasks Definition window appears.

On the Shortcut tab, check the Application Shortcut toggle box and click the ellipsis (...) button next to the Icon field. The Open window appears.

Manipulate the Open window to select the `progress.ico` file, which you copied to `%APPROOTDIR%` in Step 7.

On the Files tab, select the file `d:\demo91c\sportspro\csspin32.ocx` and click the Remove button. Then, add the file `%APPROOTDIR\csspin32.ocx`. Finally, confirm that the Application Specific toggle box is checked.

f) Save `sportspro.wcp`.

g) In the Generate window (main menu→ Deployment→ Generate), then choose Regenerate Latest Version. In the Output Directory field, enter the name of the directory where you want the generated files to be placed — that is, the value of `%OUTDIR%`. Finally, click OK.

The Application Assembler creates the application configuration file and the cabinet files and places them into the output directory you specified.

11 ♦ Use the proserve command or the Progress Explorer to start a database server for the Sports2000 database running on the AppServer machine. For example, if the Sportspro2000 database resides at %APPROOTDIR%\databases, use a command similar to the following:

```
%DLC%\bin\proserve %APPROOTDIR%\databases\sports20000.db
```

12 ♦ Use the Progress Explorer to configure the default asbroker1 AppServer instance as follows:

a) In the General Options for Broker tab:

Change the broker working directory to the AppServer work directory (%ASWORKDIR%).

In the Appservice Name list, add the service name sports2000partition.

b) In the General Options for Server tab:

Add the database connection parameter to Server Startup Parameters. For example if the Sports2000 database resides at %ASWORKDIR%\databases, enter the following string:

```
-db %ASWORKDIR%\databases\sports2000
```

13 ♦ Use the Progress Explorer or the asbman command to start the asbroker1 AppServer instance. For example (using the asbman command):

```
%DLC%\bin\asbman -name asbroker1 -start
```

A.3 Setting Up the Application to Launch from the Web Server

Follow these steps to set up the application to launch from your Web server machine:

1 ♦ Copy `sportspro.prowcapc` (the cabinet file containing the configuration file to be downloaded) and the cabinet (`.cab`) files (containing the application files to be downloaded) from `%OUTDIR%` on the AppServer machine to the Web server root directory as follows:

   a) Create a directory called `2.0` off the Web server document root directory.

   b) Copy the cabinet files to the `2.0` directory.

   c) Copy `sportspro.prowcapc` to the Web server document root directory.

2 ♦ Configure (if not already configured) and start an instance of the AppServer Internet Adapter to run as a Java servlet in the Java Servlet Engine and to access the AppServer served by asbroker1.

A.4 Installing and Running the Application on the Client Machine

Follow these steps to install and launch the application on the client machine:

1 ♦ Use a Web browser (Internet Explorer or Netscape 4.1 or newer) to go to the following URL, where *YourDomainName* represents the domain name of your Web server:

```
http://YourDomainName/web_image/bootstrap.htm
```

2 ♦ To install WebClient and the Sportspro application, respond to the prompts. The application will be installed.

NOTE: The sample application's SmartB2B and SonicMQ buttons invoke options not yet implemented.

B

Deploying the Sample Application Without IntelliStream

This appendix tells you how to deploy the Progress sample application, SportsPro, with WebClient without IntelliStream.

NOTE: There is no difference between the SportsPro application in the current version of Progress and the Sports2000 application in previous versions.

These instructions use the following symbols:

- `%DLC%` represents the Progress installation directory on a particular machine. For example, the instructions might mention `%DLC%` on the end-user machine.
- `%ASWORKDIR%` represents a Progress AppServer's work directory.
- `%MYWEBSERVER%` represents your Web server's domain name or IP address.

These instructions cover:

1. Preparing the Web server
2. Setting up the application on the AppServer machine
3. Setting up the application on the Web server machine
4. Installing and running the application on the client machine

B.1 Preparing the Web Server

Preparing the Web server involves:

- Setting up the Java Servlet Engine and the AppServer Internet Adapter
- Setting up MIME types for the application configuration file

B.1.1 Setting Up the Java Servlet Engine and the AppServer Internet Adapter

For instructions on setting up the Java Servlet Engine (JSE) and the AppServer Internet Adapter (AIA) on the Web server, see the "AppServer Internet Adapter" chapter of the *Progress Version 9 Product Update Bulletin* and the Progress Knowledge Base entries on setting up the JSE to work with the AIA. To access these, search the Knowledge Base on "JSE" and "AIA."

B.1.2 Setting Up MIME Types for the Application Configuration File

On the Web server, set up MIME types for the application configuration file. The technique for doing so depends on your particular Web server. For information on setting up MIME types for a particular Web server, see the "Configuring MIME Types for Your Web Server" section of Chapter 7, "Deploying an Application." Also, see the documentation for your Web server.

B.2 Setting Up the Application on the AppServer Machine

Follow these steps to set up the application on the AppServer machine:

1 ♦ Verify that a Progress **development** product has been installed on the AppServer machine and that the person doing the installation, when presented with the Setup Type window, chose the **complete installation**, as opposed to the typical installation or the custom installation.

2 ♦ Recursively copy the back-end files of the Sports2000 application from the `%DLC%\src\sports2000\gui` directory to the AppServer work directory.

NOTE: The `gui` directory contains both back-end and front-end application files.

3 ♦ Create a copy of the sports2000 database on the AppServer machine. The copy can reside anywhere, but note its full path — for example:

```
d:\wrk91b\sports2000.db
```

4 ♦ Start a database server for the sports2000 database running on the AppServer machine. Use the Progress Explorer or a proserve command similar to the following:

```
%DLC%\bin\proserve d:\wrk91b\sports2000.db
```

5 ♦ Compile the sports2000 back-end application files you copied to the AppServer working directory in Step 2, as follows:

a) Start a Progress session.

b) Connect to the sports2000 database.

c) Use the PRO*Tools PROPATH tool to add the AppServer working directory to the PROPATH.

d) Start the Application Compiler, select Options→ Compiler, (optionally) set Minimize R-code Size to Yes, and click OK.

e) In the Application Compiler's main window, click Modify, enter the path of the AppServer working directory, click OK, and compile the application.

6 ♦ Use the Progress Explorer to configure the default asbroker1 AppServer instance, as follows:

a) In the General Options for Broker tab:

Change the broker working directory to the AppServer work directory.

In the Appservice Name list, add the service name sports2000partition.

b) In the General Options for Server tab:

Add the database connection parameter to Server Startup Parameters. For example, if the Sports2000 database resides at `d:\wrk91b`, add the following parameter:

-db `d:\wrk91b`\sports2000

7 ♦ Use the Progress Explorer to configure and start the controlling NameServer instance for the asbroker1 AppServer.

8 ♦ Use the Progress Explorer or the asbman command to start the asbroker1 AppServer instance. For example (using the asbman command):

```
%DLC%\bin\asbman -name asbroker1 -start
```

B.3 Setting Up the Application on the Web Server Machine

Follow these steps to set up the application to launch from your Web server machine:

1 ♦ Recursively copy all files from the CDROM's `webinstall\sports2000\web_image` directory to the `sports2000` subdirectory of the Web server document root directory.

2 ♦ Copy all files from the CDROM's `webinstall\webclient\web_image` directory to the Web server document root directory's `webclient` subdirectory.

3 ♦ In the Web server document root directory's `Webclient` subdirectory, edit the `webclient.htm` file to automatically install the application automatically just after installing WebClient. To do this:

a) Uncomment the lines that invoke the One-Click application install directly. (Keep commented the lines that invoke the WebClient initializer.)

b) Change *<URL to your One-Click application installer directory>* to `http://%MYWEBSERVER%/sports2000`.

c) Change *<URL to your .prowcapc file>* to `http://%MYWEBSERVER%/sports2000/sports2000.prowcapc`.

For example:

```
ether.SetProperty("ApplicationURL", "http://pctest/sports2000");

ether.SetProperty("ProwcappURL",
"http://pctest/sports2000/sports2000.prowcapc");
```

These changes let you install WebClient and the Sports2000 application on the client machine.

4 ♦ In the Web server document root directory's sports2000 subdirectory, make the following changes:

a) Edit the bootstrap.htm file, replacing the document.location.href and window.location URLs to match the location of your configuration and webclient.htm files, respectively.

For example:

```
document.location.href =
"http://pctest/sports2000/sports2000.prowcapc";

window.location = "http://pctest/webclient/webclient.htm";
```

b) Edit the sports2000.htm file, uncommenting the line containing the prowcapc entry, and including the domain name or IP address of your Web server and the name of your application configuration file.

For example:

```
ether.SetProperty("ProwcappURL",
"http://pctest/sports2000/sports2000.prowcapc");
```

5 ♦ Use the WebClient Application Assembler PRO*Tool to generate a customized sports2000.prowcapp file. To do this:

a) Copy webinstall\sports2000\sports2000.wcp from the CDROM to any directory.

b) Open the file with the Application Assembler.

c) Modify the following URLS: the URL of your configuration file specified in the Configuration File Locator Window, the URL of the WebClient install on the WebClient tab, and the URL for your application install specified on the Options tab.

d) From the Application Assembler's main menu, choose Deployment→ Generate. Enter an output directory and a version name (for example, v1) and click OK to generate a new configuration file.

e) Copy the configuration file to the Web server document root directory's `sports2000` subdirectory.

6 ♦ Configure (if not already configured) and start an instance of the AppServer Internet Adapter to run as a Java servlet in the Java Servlet Engine and to access the AppServer served by asbroker1.

For instructions on setting up the Java Servlet Engine (JSE) and the AppServer Internet Adapter (AIA) on the Web server, see the "AppServer Internet Adapter" chapter of the *Progress Version 9 Product Update Bulletin* and the Progress Knowledge Base entries on setting up the JSE to work with the AIA. To access these, search the Knowledge Base on "JSE" and "AIA."

B.4 Installing and Running the Application on the Client Machine

Follow these steps to install and launch the application on the client machine:

NOTE: Normally Step 1 would not be necessary. But since it was impossible to predict the name of your server, the `appservtt.d` file, which contains the connection URL and which is included in the installation materials, has been set to *`YourServername`*.

1 ♦ Edit the hosts (`winnt\system32\drivers\etc\hosts`) file on the client machine to add an entry containing the IP address and host name of your server. The entry might appear as follows:

```
173.16.114.111      YourServername
```

2 ♦ Open a Web browser (Internet Explorer or Netscape Version 4 or newer) and go to one of the following URLs:

```
http://%MYWEBSERVER%/sports2000/bootstrap.htm
```

or

```
http://%MYWEBSERVER/sports2000/sports2000.prowcapc
```

For example:

```
http://pctest/sports2000/bootstrap.htm.
```

3 ♦ Install WebClient and the Sports2000 application by following the instructions and accepting the defaults.

NOTE: The sample application's SmartB2B and SonicMQ buttons invoke options not yet implemented.

Glossary

A

Application

Software that performs some real-world function (accounting, database management, word processing, etc.) as opposed to software that performs some function associated with the computer itself.

Application Component

A collection of one or more application (`.r`) files, procedure library (`.pl`) files, image files, and so on, grouped together based on related functionality. The collection of components makes up an application.

Application Configuration File

Configuration file created using the WebClient Application Assembler. This configuration file provides WebClient with the installation information needed to install, update, and run an application.

Application Configuration File Locator

A description containing information (including location information) on a particular WebClient application configuration file to be downloaded.

Application Homepage

The Web page that an ISV provides to allow an end user to install and launch a WebClient application.

Application Installer

An installer that the ISV configures to install a WebClient application on a user's machine, typically over the Web.

Application Server

The process running on an AppServer that executes 4GL procedures in response to remote procedure requests. An AppServer can have any number of Application Server processes to service remote procedure requests.

Application Service

An entire business function provided by an AppServer. An Application Service has a specific name, and that name is an arbitrary alias for the AppServer that provides a specific function such as accounting or inventory.

AppServer

Progress Software's name for its Application Server software.

ASP

Industry acronym for Application Service Provider, a term for a company that rents rather than sells a software license.

Assembler

See WebClient Application Assembler.

B

Bootstrap HTML File

An HTML file containing information that initiates the installation and execution of both WebClient and WebClient applications over the Web.

Bootstrapping Process

The process initiated when an end user launches a Progress WebClient application over the Web. This process is managed by a bootstrap HTML file provided by Progress together with the Progress WebClient Initializer. You customize this process for your particular application.

C

Codebase

Component files and all other resource files that make up an application. The codebase files are deployed on a server. The location of these resource files is specified using the Codebase Locator.

Codebase Locator

A description containing information (including location information) on WebClient application files to be downloaded.

Code Page

A table that maps each character on it to a unique numeric value.

D

Deployment

The process of distributing and installing the files and programs necessary to install and run a WebClient application over the Web.

Digital Certificate

An attachment to an electronic message that uses a public key encryption system to verify that clients and servers are who they are representing themselves to be. Digital certificates are issued by trusted third parties known as certification authorities (CAs).

Digital Signature

A digital code attached to an electronically transmitted message that attests to the file's authenticity (that the file is from the person or organization claiming to have sent it) and integrity (that it has not been tampered with).

E

External Installer

An application install program, such as InstallShield or similar type of installer technology, used to install your application. An external installer should be responsible for completing all application installation tasks required. An external installer should also make an uninstall program available to users.

H

HTML

Acronym for HyperText Markup Language, the earliest language used on the World Wide Web, which is now being superseded, to some extent, by XML, the eXtensible Markup Language.

I

IntelliStream

Progress technology that allows the application deployer to specify that at application installation, WebClient should download only those files required to initially run the application. Remaining files would be downloaded when the end user needed to use specific functionality provided by the application. This technology also allows for application updates at the file level (including individual procedure files within procedure libraries).

IntelliStream includes WebClient Application Assembler. You use WebClient Application Assembler to define how to install and update your WebClient application.

IntelliStream also lets the application deployer digitally sign each file to be downloaded. If this option is chosen, the WebClient on the end user's machine verifies the digital signature of each file downloaded.

ISV

Industry abbreviation for Independent Software Vendor.

M

Microsoft Authenticode Technology

The technology WebClient uses to create and verify digital signatures.

Microsoft Cabinet (.cab) File

A file in which WebClient stores compressed and signed application files that users download to their machines.

Migration

Term used metaphorically for the process of orderly change, for example from one code version to another.

MIME Handler

A file association mechanism that associates applications with MIME types.

MIME Type

A type associated with a file extension using the Multipurpose Internet Mail Extension standard. This type can have an associated application specified that starts up in response to being recognized by a Web browser or Internet mailer.

N

NameServer

A process that directs a client connection request to an AppServer that supports a specified business function. A client indicates which AppServer instance it wants to connect to by specifying an Application Service name that identifies the required business function.

O

Out-hosting

A service provided by a third-party who hosts Web sites produced by and owned by others. A WebClient application might be hosted by an application deplores or by an out-hoster.

P

Platform

An industry term referring to the type of system (for example, Windows NT on Intel) on which some software runs.

Proxy Host

The host name or IP address of a machine that hosts the proxy server between a LAN and the Internet.

Proxy Port

The port on which a proxy server listens for messages between a LAN and the Internet.

Proxy Server

A server that provides a firewall to filter both incoming and outgoing messages between a LAN and the Internet.

S

Secure AppServer

Provides support for sending and receiving encrypted data. A Secure AppServer includes the AIA/S component and allows WebClients and 4GL Clients that have client-side security installed to send encrypted data over the Internet, or an intranet or extranet.

Secure Socket Layer (SSL)

Industry-standard protocol that supports sending encrypted information across the Internet. SSL runs above TCP/IP and below the HTTP protocol. The SSL protocol provides for both client authentication by the server as well as server authentication by the client.

Security Cache

A cache of authentication items (such as user IDs and passwords) entered by the end user for accessing servers. The cache is maintained by WebClient and accessible to the application. There are separate security caches for the application configuration file locator and for the codebase locator, although, under certain circumstances, the application configuration file locator cache can be accessed while accessing the codebase locator cache.

Self-extracting Installation

A software installation contained in a single self-extracting executable file that a user can copy and run on their own machine to install the software.

Shortcut Mode

A mode of running the Progress WebClient Initializer where the initializer runs, downloads, and opens a WebClient application configuration file whose location is stored in the registry by the installer for a particular WebClient application.

Single Sign-on

A feature that allows an end user to enter a particular user ID and password once, even though the application might need it multiple times. WebClient provides single sign-on by caching authentication items entered by the end-user, then letting the application access the cached items.

W

WebClient

A Progress client specially designed for installing and running applications over the World Wide Web or a company intranet. WebClient does not support direct database access but does execute compiled 4GL (r-code) and perform other functions similar to a standard GUI 4GL client running in Windows.

WebClient Application

A Progress application separated into two parts: 1) a back-end part that runs on an AppServer, and 2) a front-end part that runs in a Progress WebClient and that is downloadable over the World Wide Web.

WebClient Application Assembler

A PRO*Tool in the Progress ADE that allows the ISVs to assemble a WebClient application and to create and maintain the files needed to deploy and manage a WebClient application.

WebClient Application Configuration File

File generated by the WebClient Application Assembler. This file contains deployment information about a single WebClient application.

WebClient Application Manager

A program installed with WebClient. WebClient Application Manager allows your user to configure and manage their use of WebClient application, and helps users provide you with information needed to solve application installation problems.

WebClient

Program that is responsible for running an application on a user's machine.

WebClient Initializer

A WebClient executable that initializes WebClient to run a WebClient application. The initializer validates an end user's system configuration, ensures that the appropriate WebClient and application versions are installed, and invokes the WebClient to run the application.

WebClient Install Page

A Progress provided Web page used to install a Progress WebClient over the Web.

Web-deployed Application

Any Progress application that is deployed across the World Wide Web from a Web server to a client machine, typically using a Progress WebClient.

Web-enabled 4GL Client

A Progress client running a 4GL application that is distributed across the World Wide Web using the Progress AppServer. The client connects to the AppServer through a Web server using the AppServer Internet Adapter and HTTP or HTTPS.

Web-enabled Application

A Progress multi-tier application that runs using a Web-enabled 4GL client and Web-enabled AppServer.

Web-enabled AppServer

A Progress AppServer that handles client connections through a Web server using the AppServer Internet Adapter and HTTP or HTTPS.

Index

A

Application
  component files 7–21
  components 7–3
  components update files 7–21

Application configuration file 7–20

Application mode 8–5

Application name
  registry keys 7–7, 7–10

Applications
  registry keys 7–10

Applications, signed and unsigned 5–9

AppServer protocol 4–6

As Needed components 1–9

Ask at Startup components 1–9

Assymetric ciphers 5–3

ASYNC requests 6–7

Audience ix

Authentication
  IIS Web server 7–24

Authentication dialogs 5–14

Authenticity (of a downloaded file) 5–2

B

Bold typeface
  as typographical convention xi

Bootstrap HTML file
  defined 7–31

Bootstrap mechanism A–3

bootstrap.htm file 7–34
  customizing 7–34
  starting WebClient applications 7–31

Bootstrapping
  with no WebClient installed 7–34
  with version of WebClient installed 7–34

C

Case Sensitivity
  UNIX Web servers 7–24

CD-ROM
  installing WebClient 3–2
  installing WebClient applications 3–3

CD-ROM deployment
  creating 7–28

Codebase locator 1–8

CODEBASE-LOCATOR handle 6–2

compared to standard 4GL client 1–2

Components
  application 7–3
  download mode 1–9
  registry keys 7–11

Computational feasibility 5–3

Configuration file locator 1–12

Customizing
  bootstrap.htm 7–34
  WebClient install 3–2
  WebClient installer 7–27, 7–28

Customizing bootstrap.htm file 7–34

D

Defining
  system tasks 7–18

Deploying WebClient applications
  authentication on Microsoft IIS 7–24
  case sensitivity
    installation filenames 7–24
    URL pathnames 7–36
  end user documentation 8–2
  executable directories 7–24

Digital signatures 1–10

Documentation for end users 8–2

Download mode
  for components 1–9

E

End user
  providing documentation 3–7

Error messages
  displaying descriptions xix

Example procedures xvi

Executable directories
  Web installer 7–24

External installer
  installation overview 3–5
  installation tasks 3–5
  registry keys for uninstall 7–11
  uninstalling WebClient applications 3–6
  WebClient application updates 3–5
  WebClient applications 3–3
  Web-enabled installer 7–18

F

Features
  IntelliStream 3–4

File protocol 4–6

Firewalls
  setting after WebClient install 8–6

H

Help
  xix

HTTP protocol 4–5

I

IIS Web server
authentication 7–24

Input files
WebClient Application Assembler 7–12

Installation overview
external installer 3–5
IntelIiStream 3–4

Installation recommendations
WebClient applications 3–3

Installation tasks
external installer 3–5

Installer
WebClient 7–27

Installing
WebClient applications 3–3

InstallShield One-Click Install
customizing 7–28

Integrity (of a downloaded file) 5–2

IntelliStream
application components 7–3
features 3–4
installation overview 3–4
installing WebClient applications 3–3
uninstalling WebClient applications 3–6
WebClient application updates 3–5

Italic typeface
as typographical convention xi

K

KEEP-CONNECTION-OPEN attribute 6–3

Keystrokes xii

L

Launching applications
shortcuts 7–31
URL 7–31

LOAD–ICON method support for URLs 6–13

LOAD–IMAGE method support for URLs 6–13

LOAD–IMAGE support for URLs
authentication 6–12

LOAD-IMAGE() method 6–12

LOAD–IMAGE–DOWN method support for URLs 6–13

LOAD–IMAGE–INSENSITIVE method support for URLs 6–13

LOAD–IMAGE–UP method support for URLs 6–13

M

Manual
organization of x
syntax notation xii

Matching letter case
UNIX Web server 7–36

Messages
displaying descriptions xix

Microsoft AuthenticodeTM Technology 1–5

Microsoft cabinet (.cab) files 1–5

MIME type settings
Apache Web Server 7–25
iPlanet/Netscape Web Servers 7–26
Microsoft IIS 7–26

MIME types A–3, B–2
  configuring for Web servers 7–25

Modes
  WebClient Application Manager 8–5

Monospaced typeface
  as typographical convention xi

O

One-Click Install image
  WebClient 3–2

Overview
  uninstalling WebClient applications 3–6
  WebClient Application Assembler 1–7
  WebClient application updates 3–5
  WebClient install 3–2

P

Procedures
  examples of xvi

Progress 4GL install procedure
  WebClient applications 3–3

Project file
  WebClient 7–2

PROPATH support for URLs 6–12
  authentication 6–12

Providing documentation
  for end users 3–7

ProwcappLocator
  registry keys 7–8

Proxy Host (–proxyhost) startup parameter 8–6

Proxy Port (–proxyport) startup parameter 8–6

Proxy servers
  setting after WebClient install 8–6

–proxyhost startup parameter 8–6

–proxyport startup parameter 8–6

Public Key Infrastructure (PKI) 5–4

Public-key certificates 5–4

R

Registry keys
  application name 7–7, 7–10
  applications 7–10
  components 7–11
  external installer 7–11
  ProwcappLocator 7–8
  vendor name 7–7
  WebClient 7–5
  WebClient installer 7–5

Running applications
  Web access 8–2

S

SEARCH function support for URLs 6–12
  authentication 6–12

Setting MIME type
  Apache Web Server 7–25
  iPlanet/Netscape Web Servers 7–26
  Microsoft IIS 7–26

Shortcut
  starting WebClient applications 3–6

Shortcuts
  launching applications 7–31
  starting WebClient applications 7–31

Single sign-on 1–13, 5–13, 6–8

SmartObjects 2–2

Sports2000 A–1, B–1

Sports2000 application A–1

SportsPro A–1, B–1

Sportspro application A–1

Starting
  WebClient applications 3–6, 7–31
  WebClient applications using a shortcut 3–6
  WebClient applications using a URL 3–6

Starting WebClient applications
  using a shortcut 7–31

Startup parameters
  Proxy Host (–proxyhost) 8–6
  Proxy Port (–proxyport) 8–6

Syntax notation xii

System Tasks 7–4

System tasks 1–10, 1–13

System Tasks Definition
  defining 7–18

T

TEMP–TABLE parameters 2–2

Typographical conventions xi

U

Uninstall
  registry keys 7–11

Uninstalling
  WebClient applications 3–6

Uninstalling WebClient applications
  external installer 3–6
  IntelliStream 3–6
  overview 3–6

UNIX Web server
  matching letter case 7–36

UNIX Web servers
  case sensitivity 7–24

URL
  starting WebClient applications 3–6, 7–31

URL support
  authentication 6–14
  case sensitivity 7–36
  launching applications 7–31
  LOAD–IMAGE method 6–13
  PROPATH 6–12
  SEARCH function 6–12

URLs in the PROPATH 6–12

V

Vendor name
  registry keys 7–7

W

Web access
  running applications 8–2
  WebClient applications 8–2

Web authentication
  4GL support 6–14

Web deployment
  creating 7–28

Web install
  WebClient applications 3–3

Web installer
  executable directories 7–24

Web servers
  configuring MIM types 7–25

WebClient 1–2
  CD-ROM install 3–2
  compared to 4GL client 1–2
  installer 7–27
  One-Click install image 3–2
  project file 7–2
  registry keys 7–5
  use senarios 1–3

WebClient Application Assembler 1–7
  generating files 7–3
  input files 7–12
  managing applications 7–2
  overview 1–7

WebClient Application Configuration file 1–5

WebClient Application Manager 1–5
  defined 8–4
  edit dialog box 8–7
    General tab 8–9
    Log tab 8–11
  functions 8–4
  main dialog box 8–5
    Applications tab 8–7
    WebClient tab 8–5
  modes of execution 8–5
  setting a firewall 8–6
  shortcut syntax 7–7

WebClient application updates
  external installer 3–5
  IntelliStream 3–5
  overview 3–5

WebClient applications
  CD-ROM install 3–3
  external installer 3–3
  installation recommendations 3–3
  installing 3–3
  installing with IntelliStream 3–3
  launch mechanism
    overview 7–30
  launch options 7–31
  Progress 4GL install procedure 3–3
  starting 3–6, 7–31
  starting with bootstrap.htm file 7–31
  uninstalling 3–6
  URL for starting 7–31
  Web install 3–3

WebClient executable 1–5

WebClient Initialize 1–2

WebClient Initializer 1–5
  defined 7–30
  shortcut syntax 7–6
  validating acceptable versions
    application 7–37
    Progress WebClient 7–36

WebClient install
  customizing 3–2
  overview 3–2

WebClient installer
  customizing 7–28, 7–29
  registry keys created 7–5
  running application installer
    example 7–30
  running WebClient Initializer
    example 7–30

WebClient mode 8–5

WebClient project (.wcp) file 1–5

WebClient version information file 7–21

webclient.htm 7–29
  customizations 7–29
  customizing a WebClient installer 7–29

Web-enabled installer
  external installer 7–18