Progress®



Develop OpenEdge WebClient Applications

OpenEdge®

Table of Contents

Copyright....7
Preface....9

Develop WebClient applications....11
WebClient installation types....14
Requirements for using WebClient....17
WebClient architecture....17
WebClient Application Assembler....17
WebClient Deployment Packager....18
IntelliStream and WebClient....18
Using non-IntelliStream technology....19
Using IntelliStream with non-IntelliStream technologies....20
Run the application....20
Application requirements....21
WebClient platforms....22
Define the WebClient application's platform....23
Running 32-bit WebClient applications on 64-bit Windows....24
Deploy a sample application using WebClient and IntelliStream....24

Design Your End User's Experience....31
Installation process....32
Microsoft Windows installations....32
WebClient application installation....33
IntelliStream....33
External WebClient application installer....34
Updates for your WebClient application....34
Uninstalling your application....35
Starting WebClient applications....35
Provide documentation for your WebClient application....35

Host your WebClient Deployment....37
Hosting application configuration files....37
Hosting WebClient codebases....38

Design for Security....41
Digitally signing files....41
Making Microsoft signing tools available to OpenEdge....42
How WebClient uses digital signatures....42

Protecting servers with user IDs and passwords....44
Single sign-on and security caching....46
Using WebClient logon dialogs in your applications....47
Choosing secure communication protocols....47

Develop Your Application....49

Using the CODEBASE-LOCATOR handle and its attributes....49
Checking application registry access....52
Managing application server connections....52
Implementing single sign-on....53
Using WebClient authentication dialogs 53
Using URLs in PROPATH in WebClient for image file downloads....55
PROPATH error handling....56
Compiling the WebClient application....57

Your End Users' Experience....59

Preparing documentation for end users....59
WebClient Application Manager....61
Using the WebClient Application Manager....61
WebClient tab....62
Applications tab....63
Edit dialog....63

Deploy a WebClient Application....69

WebClient Application Assembler prerequisites....70
Installation types....70
Administrator installations....71
Personal installations....72
Choosing an installation method....73
IntelliStream....73
External installer....74
Using a combination of IntelliStream and an external installer....78
Defining applications in WebClient Application Assembler....80
General tab....81
Component tab....84
Options tab....85
WebClient tab....88
Files generated by the WebClient Application Assembler....88
Customizing deployment configurations 89
Distribution of the Deployment Packager....90
Changes allowed through the Deployment Packager....90
Defining a custom deployment configuration....91
Validating the custom deployment configuration....92

Generating the .prowcapc file....93
Hosting the application on a server....94
Hosting the configuration file....94
Hosting your application component files....94
Hosting your external installer....95
Hosting on a Web server....95
Installing WebClient....97
Accessing the WebClient installation files....97
Customizing the WebClient installation....98
Bootstrapping a WebClient application....101
Customizing the bootstrap.htm file....104
Checking for an acceptable version of WebClient....106
WebClient application updates....106
Updating procedure libraries....107
Suppressing the WebClient Application Update dialog....107
WebClient and OpenEdge GUI for .NET....109
.NET Framework....109
.NET assemblies....111
Other files 111

Index....113

Copyright

© 2021 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

These materials and all Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. The information in these materials is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear therein. The references in these materials to specific platforms supported are subject to change.

DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Defrag This, Deliver More Than Expected, DevReach (and design), Icenium, Inspec, Ipswitch, iMacros, Kendo UI, Kinvey, MessageWay, MOVEit, NativeChat, NativeScript, OpenEdge, Powered by Chef, Powered by Progress, Progress, Progress Software Developers Network, SequeLink, Sitefinity (and Design), Sitefinity, Sitefinity (and design), SpeedScript, Stylus Studio, Stylized Design (Arrow/3D Box logo), Styleized Design (C Chef logo), Stylized Design of Samurai, TeamPulse, Telerik, Telerik (and design), Test Studio, WebSpeed, WhatsConfigured, WhatsConnected, WhatsUp, and WS_FTP are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries.

Analytics360, AppServer, BusinessEdge, Chef Automate, Chef Compliance, Chef Desktop, Chef Habitat, Chef WorkStation, Corticon.js, Corticon Rules, Data Access, DataDirect Autonomous REST Connector, DataDirect Spy, DevCraft, Fiddler, Fiddler Everywhere, FiddlerCap, FiddlerCore, FiddlerScript, Hybrid Data Pipeline, iMail, JustAssembly, JustDecompile, JustMock, KendoReact, NativeScript Sidekick, OpenAccess, PASOE, Pro2, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, Push Jobs, SafeSpaceVR, Sitefinity Cloud, Sitefinity CMS, Sitefinity Digital Experience Cloud, Sitefinity Feather, Sitefinity Insight, Sitefinity Thunder, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, Supermarket, SupportLink, Unite UX, and WebClient are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

July 2021

Product version: Progress OpenEdge 12.4

Preface

Purpose

This manual describes the architecture and options for enabling a standard OpenEdge application for deployment over the Web using the OpenEdgeWebClient. 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.

Audience

The audience for this manual includes:

- OpenEdge developers who 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 WebClient application developed by someone else.

Organization

Develop WebClient applications on page 11

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

Design Your End User's Experience on page 31

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

Host your WebClient Deployment on page 37

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

Design for Security on page 41

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

Develop Your Application on page 49

Describes how to use ABL to implement WebClient application design decisions and requirements.

Deploy a WebClient Application on page 69

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.

Your End Users' Experience on page 59

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.

Deploy a sample application using WebClient and IntelliStream on page 24

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

1

Develop WebClient applications

By using the WebClient, you can take a Web-enabled OpenEdge application, deploy and configure its back-end components as desired, package its front-end components, and host the packages on a server 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.

WebClient vs. standard ABL client

There are several factors that you should consider to determine if the WebClient is right for you. First compare the WebClient to a standard ABL client.

Feature	WebClient	Standard ABL client
Execution	Contains no compiler; runs as a reduced-size executable.	Compiles source and runs as a complete ABL client executable.
Data access	Supports temp-table access, but does not support direct access to the OpenEdge database or DataServers; relies on a connected application server to provide indirect access to all databases and DataServers.	Supports direct access to all data objects, as well as indirect access to databases and DataServers using a connected application server.
Customization	Cannot be customized using OEBuild.	Can be customized using OEBuild.

Feature	WebClient	Standard ABL client
UI	Supports only the graphical user interface (GUI) and batch modes in Windows.	Supports all ABL user-interface modes.
Launch method	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 ABL procedures that the application comprises 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 ABL client on the end user's behalf. The ABL procedures that the application comprises must be installed on the end user's machine prior to run time.

WebClient

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.

ASP

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 like the following:

- **Identifying the application** — End users identify the application on your Web site as a likely business solution through a Web search.
- **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.
- **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, and so on).
- **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 OpenEdge components. End users then run the application, which access the business logic on your back-end server.

IT department

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:

- **Identifying the application** — You search the Web to identify an application to deploy via WebClient.
- **Evaluating the application** — Your end users sample the application through a live demonstration, installed and executed directly on their systems over the Internet.
- **Purchasing the application** — You buy the application.
- **Deploying the application** — You deploy the application on a local machine.
- **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.

WebClient files and software

The following files and software support your WebClient application deployment:

- **WebClient executable (`prowc.exe`)** — The WebClient main executable, responsible for running the application on the user's machine.
- **WebClient Initializer (`prowcini.exe`)** — The program responsible for coordinating the installation and execution of a WebClient application. The Initializer is installed with WebClient.
- **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 Deployment Packager (contained within `prowcappmgr.exe`)** — A utility used to modify the deployment configuration for an application.
- **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 WebClient 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.

Note: In Windows Explorer, if you double-click on a project file, the WebClient Application Assembler starts and opens that project definition.

- **Microsoft Cabinet (`.cab`) files** — Files into which the procedure files, image files, etc., that make up 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.
- **WebClient application** — The application to be deployed through WebClient.

WebClient versus WebClient applications

The WebClient itself is separate from the applications that you deploy with it. When used alone, *WebClient* refers to the WebClient executable and the rest of the infrastructure on which a deployed application runs. *WebClient application* refers to the deployed application itself.

The OpenEdge WebClient enables you to build and distribute updates for web-based OpenEdge applications.

For details, see the following topics:

- WebClient installation types
- Requirements for using WebClient
- WebClient architecture
- Run the application
- Application requirements
- WebClient platforms

- Deploy a sample application using WebClient and IntelliStream

WebClient installation types

There are two types of WebClient installations: *Administrator installations* and *Personal installations*. The two types need different permissions and write files to different locations.

Administrator installations

An Administrator installation creates a single installed copy of a WebClient application on a machine that all end user have access to. To accomplish this, the install writes the WebClient and WebClient application information to areas that are common to all users a machine, such as the `HKEY_LOCAL_MACHINE` (HKLM) registry key and the `C:\Program Files` directory. Using these shared areas requires write permissions that are restricted to Admin users. After the installation, these privileges are still required to handle application updates and to download components as needed. If desired, the Admin can set up batch jobs to handle application updates.

For security reasons, you might not want to grant all users Admin privileges. To use a WebClient application in that case, administrators must install the application and grant limited write privileges to the application users.

Personal installations

An administrator does not control this type of installation. If the end users have access to the WebClient Initializer, they can decide to install a personal copy of the application on their machine.

Only the user who installs the personal instance has access to it. For example, if an Administrator ran the WebClient Application Manager, they would not see any of the personal instances that users had installed on the their machine.

Note: Some IntelliStream System Tasks require Administrator privileges to install. If your application makes use of these tasks, users cannot install personal instances of your WebClient application.

A personal installation creates a separate copy of the WebClient application for each user. The installation uses the `HKEY_CURRENT_USER` (HKCU) registry key and the user's own application data directory, such as `C:\Documents and Settings\`*`user-name`*`\Application Data`.

If the WebClient installer determines that an end user does not have Admin permissions, the installer switches to performing a personal install.

Multiple installations on one machine

Progress recommends that you do not use both installation types of the same WebClient application on a single machine. If several users on the same machine use a WebClient application, a single Administrator installation uses less resources. You should uninstall any personal installations of an application if an Administrator installation exists.

However, circumstances might cause a machine to host a mix of Administrator and personal instances of WebClient applications. As a result, a user might attempt to use a personal instance of a particular application that they installed before the Administrator instance was installed. You might also have several different WebClient applications installed on the same machine. When there are multiple WebClient installations on a single machine, an Administrator installation has precedence over a personal installation.

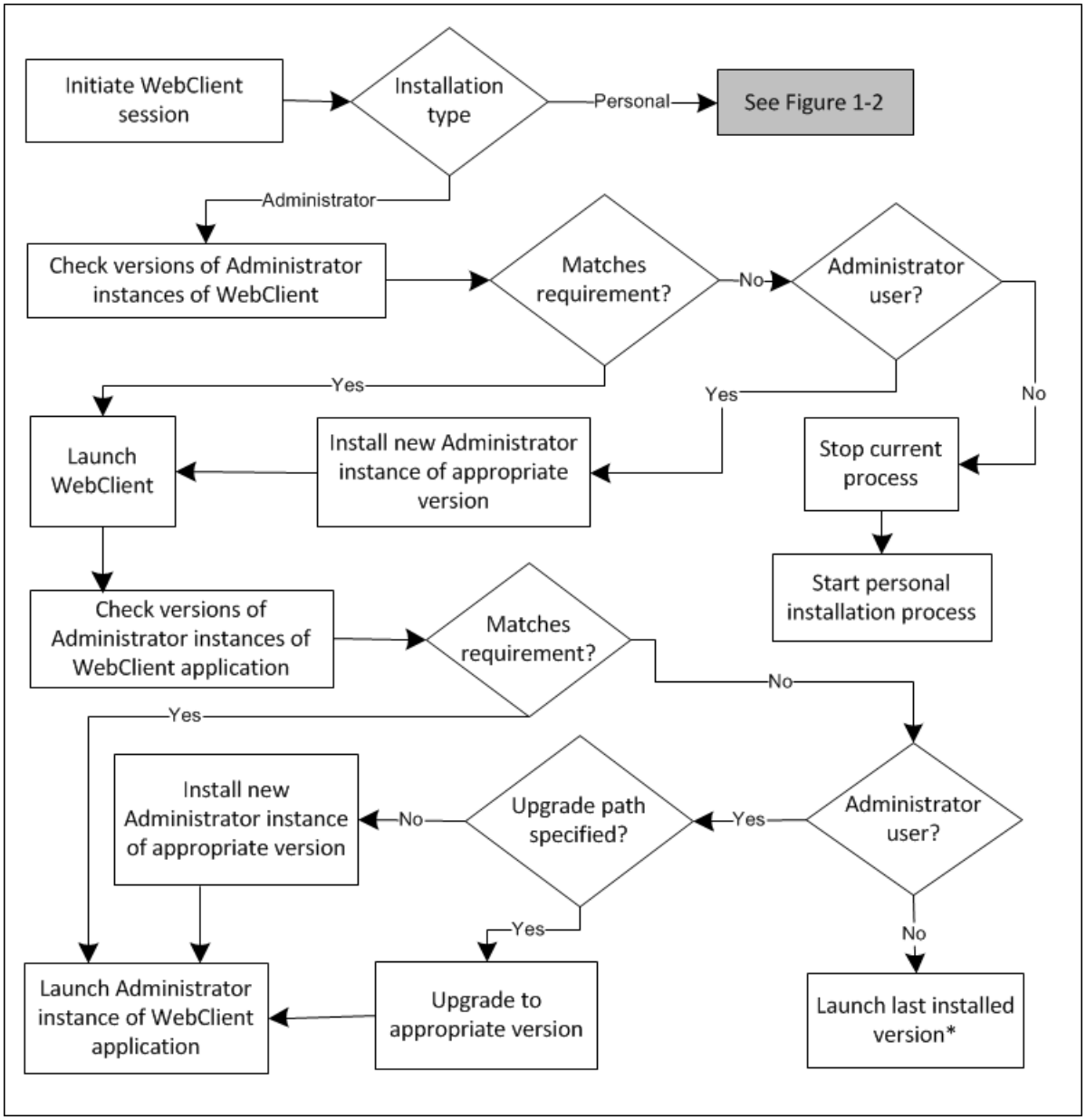
Figures 1 and 2 show the logic process by which WebClient Initializer chooses which instances of the WebClient and the WebClient application to start. The first thing that the Initializer checks is if there is an Administrator instance of WebClient application available. If there is an Administrator instance of the WebClient application, it runs through the process shown in Figure 1. If not, it runs through the process shown in Figure 2.

If there is no appropriate version of the WebClient available, the process does not upgrade an existing instance. Instead, it installs a completely new instance. This preserves the already installed instances for any other WebClient application that might need those particular versions.

When an end user does not have sufficient privileges to update an Administrator instance of a WebClient application, the Initializer launches the last installed version of the application. However, the end user might see error messages when necessary upgrades cannot be made.

If the Initializer does not find an appropriate version of the WebClient application, the result depends on how the application deployer designs the deployment. If the deployer specified an upgrade path for the currently installed version, the Initializer launches the upgrade. If the deployer did not specify an upgrade path, then the Initializer installs a new personal instance of the WebClient application.

Figure 1: Figure 1: Launching an Administrator instance of a WebClient application

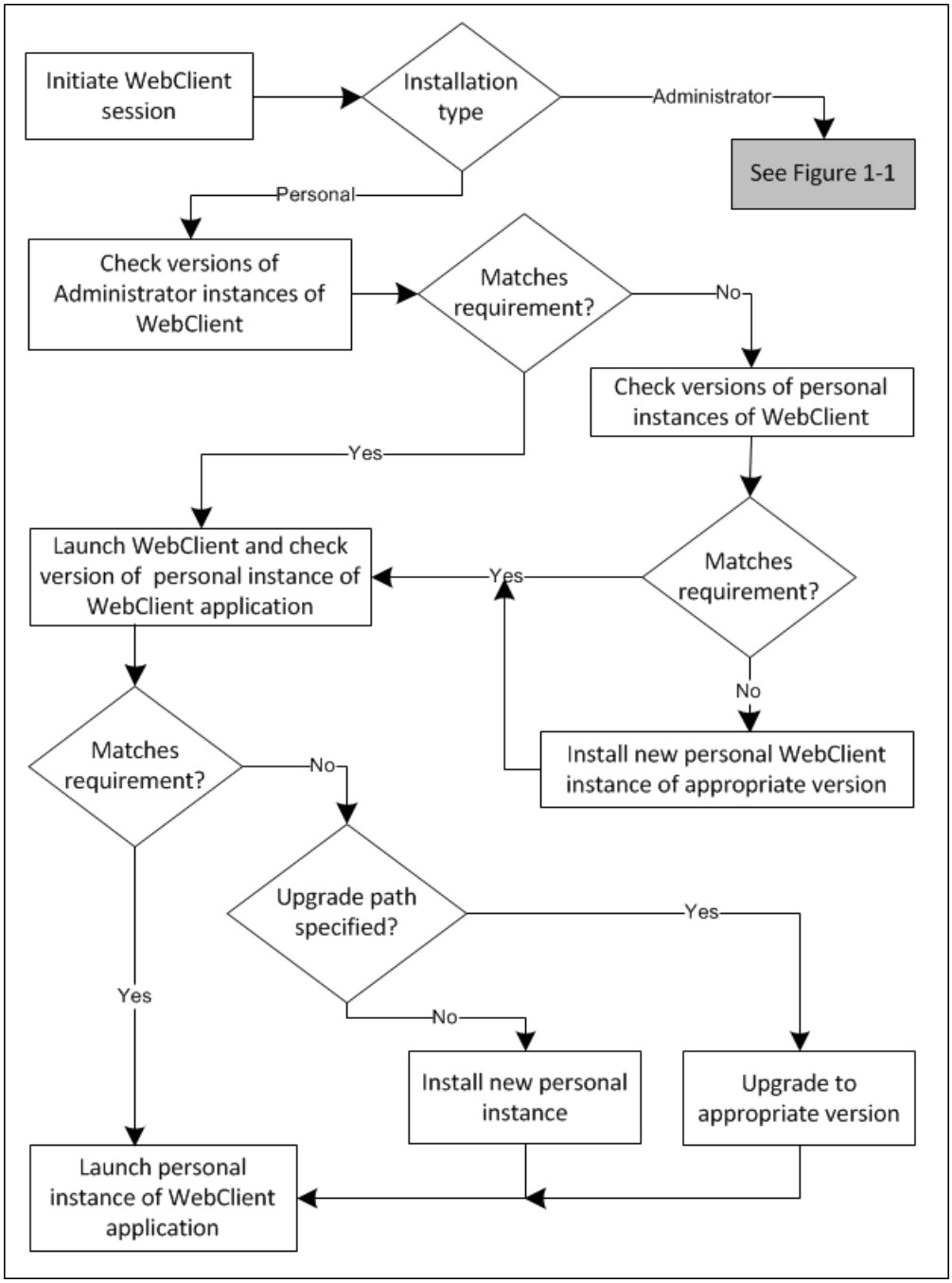


Note: The Figure 1 assumes that you have already performed the initial installation of the WebClient application. The asterisk (*) indicates that the user might see errors when they cannot make necessary upgrades.

When the Initializer does not find an appropriate version of the WebClient application, it checks whether the deployer has specified an upgrade path for the version of the existing instance. If so, it upgrades the existing instance. Otherwise, it installs a new instance of the WebClient application. Administrators might need to uninstall superseded instances to free up disk space.

Even for a personal installation of a WebClient application, the Initializer first checks if there is an Administrator instance of WebClient from some other installation with an acceptable version. If it finds no appropriate Administrator instance of WebClient, it switches to looking for personal instances of WebClient.

Figure 2: Figure 2: Launching a personal instance of a WebClient application



Requirements for using WebClient

To successfully install and use a WebClient application, the end user's machine requires the following:

- Microsoft Windows 8.1 or later
- Sufficient disk space for the WebClient and the application files

  Note: During installation, you should shut down non-essential programs and processes to free up DLLs.

- Appropriate write privileges for installation and upgrades

  The standard model assumes that the WebClient automatically checks for and installs upgrades when it runs. Any end user has privileges to upgrade their personal WebClient installations. However, an administrator must grant certain privileges to non-administrators to enable them to upgrade an Administrator WebClient installation. If the end users are not granted those privileges, an administrator would need to arrange batch jobs with proper privileges to keep the WebClient application up-to-date.

WebClient architecture

The WebClient architecture is composed of WebClient Application Assembler, WebClient Deployment Packager, and choosing to use IntelliStream, non-IntelliStream, or a combination of the two technologies.

WebClient Application Assembler

The WebClient 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 WebClient Application Assembler to generate files for deploying and updating the front end of your applications. This tool aids in the following tasks for 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 the installation of your application.

For more information, see Deploy a WebClient Application on page 69

Using the Application Assembler, you can implement installations and updates that use IntelliStream technology, non-IntelliStream technology, or a combination of the two. For more information, see the IntelliStream and WebClient on page 18, the Using non-IntelliStream technology on page 19, and the Using IntelliStream with non-IntelliStream technologies on page 20.

WebClient Deployment Packager

The WebClient Deployment Packager is a tool for creating custom deployment configurations after the application configuration is generated by the WebClient Application Assembler.

The WebClient Deployment Packagercan tailor the deployment information (such as the location of the server code and information about digital signatures for security) to the needs of specific users, the Deployment Packager eliminates the need to regenerate the `.prowcapp` and `.prowcapc` files. A standalone version is provided for distribution to customers or application hosters who want to customize their own deployment configuration. For more information, see Deploy a WebClient Application on page 69

IntelliStream and WebClient

You can use IntelliStream to create dynamic downloads and updates for your application.

By using IntelliStream technology, you can configure the initial download to only provide the files that are needed to start the application. Then, as the end user runs other parts of the application, the WebClient automatically downloads and installs the additional files as needed. IntelliStream also activates every time the end user starts an application and checks for updated application components. If updated componetns are detected, IntelliStream downloads and installs them before the application starts.

Note: Some IntelliStream System Tasks require Administrator privileges to install. If your application makes use of these tasks, users cannot install personal instances of your WebClient application. For more information, see System Tasks and personal installations on page 87.

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 Deploy a WebClient Application on page 69.

WebClient codebase locator

As part of defining the application for WebClient, you describe the server that contains the application components to be downloaded, using the Application Assembler. This description is called the *codebase locator*, which is used by the WebClient to access a server called the *codebase server*. For more information, see Host your WebClient Deployment on page 37

Components and download mode

To download the files for 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, and so on. A single cabinet file comprises an application component.

You assign one of three download modes to each application component:

- **At Startup** — The initial download includes the component.
- **As Needed** — WebClient automatically downloads the component 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 prompts end users to identify which **As Needed** components they want downloaded. WebClient then downloads these components (along with the at-startup components).

For more information, see Deploy a WebClient Application on page 69

Component update files

When you define 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 contains changed application files. 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 application needs updating, WebClient downloads the necessary component update files to update the end user's installation to the newest version. For more information, see Deploy a WebClient Application on page 69

Note: IntelliStream does not support incremental updates to memory-mapped procedure libraries. For more information, see the Updating procedure libraries on page 107.

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 each version of the r-code from a version-information file. For more information on the version-information file, see Deploy a WebClient Application on page 69

Obsolete versions

When the deployer defines a new version of the application, they can also define certain previous versions as *obsolete*. For example, if you know that all end users have upgraded beyond a certain version, you might define any earlier version as obsolete. When WebClient generates a new version, it does not produce component update files for obsolete versions. For more information, see Deploy a WebClient Application on page 69

System tasks and ABL 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 ABL routine you supply to perform additional installation tasks. For more information on these facilities, see Deploy a WebClient Application on page 69

Note: Some IntelliStream System Tasks require Administrator privileges to install. If your application makes use of these tasks, users cannot install personal instances of your WebClient application. For more information, see the System Tasks and personal installations on page 87 System Tasks and personal installations.

Digital signatures

By using WebClient you can digitally sign each deployed file. When the end user downloads a file, the end user's WebClient instance validates the digital signature. For complete information on digital signatures in general and how WebClient uses them in particular, see Design for Security on page 41

Using non-IntelliStream technology

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

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 it automatically starts the application installation.

Using IntelliStream with non-IntelliStream technologies

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

You can use IntelliStream for initial downloads and updates:

- To minimize the time required for downloads
- To avoid having to write an InstallShield script
- To avoid having to purchase InstallShield

Or you can use an external installer for initial downloads and updates if:

- InstallShield is already being used
- Minimizing the download time is not critical
- Or 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.

Run the application

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. This is done through the Configuration File Locator.

The config file locator

When the application deployer uses the Application Assembler, the Application Assembler writes the application definition to a configuration file. The end user downloads the file as part of the initial download, and each time the end user starts the application.

As part of defining the application, the application deployer defines the server from which the end user downloads the application configuration file. WebClient uses this description, called the *configuration file locator*, to access the application configuration file on the configuration file server. For more information, see Host your WebClient Deployment on page 37.

Note: To allow the WebClient to continue execution even if the configuration file server is not available, select **Allow execution to continue when unable to refresh prowcapp file** under the **General** tab of the Web Application Assembler. See Defining applications in WebClient Application Assembler on page 80 for more information.

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, the end user's WebClient instance 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. When this occurs, the end user is offered the option of downloading the newer version. If the end user so requests, WebClient downloads and applies the updates. For more information on acceptable-to-run versions, see Deploy a WebClient Application on page 69

Server sharing

WebClient and the application can share a single server. An application server that contains application components downloaded by the WebClient might also contain business logic accessed by the same application. When the WebClient and the application access the same server, they can share a connection. For more information on server sharing, see the Managing application server connections on page 52.

Single sign-on

Progress 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. Facilitating single sign-on for the end user eliminates the need for them to enter their logon information multiple times. If SSO is enabled, the WebClient caches the end user authentication information and grants the application access to this cache.

If the end user desires, WebClient can maintain the authentication cache across end user sessions. This is called persistent caching. The you can disable persistent caching if necessary, and the end user can clear the persistent cache. For more information, see the Implementing single sign-on on page 53.

Application requirements

Progress recommends that you follow modern best practices when designing an application that you plan to deploy with WebClient.

Access databases only through an application server

A WebClient application must access databases through an application server, as the client cannot directly access a database. All of the database data presented by the application to the end user must originate on an application server. All database data entered by the end user must be sent to an application server for evaluation and storage. To pass relational-database data between an application server and client, you can use temp-table parameters in remote procedures. For more information on temp-table parameters and remote procedures, see Basic ABL developer.

Modularize your code by function

Consider this scenario: 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. At application-deployment time, you might want only certain *components* (groups of application files) to download initially, and other components to download only when they are first called. When you design your application, you can make deploying the application more efficient by the way you assign code to procedure files. Specifically, avoid assigning pieces of code that support dissimilar functions to the same procedure file, since you might want unrelated modules of an application to download at different times.

Note: However, you do not need to place all code that supports similar functions to the same procedure file, 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 specific routines for one type of insurance.

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. However, you can deploy the application as a single large component. For more information, see Deploy a WebClient Application on page 69.

Choose which installation types to support

Administrator installations and personal installations have different requirements. You must consider these requirements while designing your application and its WebClient deployment. An Administrator installation requires the end users have write access to certain areas that are normally restricted to administrators, but it imposes no additional restrictions on your designs. A personal installation does not have that write access, which results in some limitations. For example, a personal installation cannot:

- Install files to certain areas.

  Adding such files through the IntelliStream System Tasks causes install errors for a personal installation. For more information, see the System Tasks and personal installations on page 87.

- Install a .NET Framework.

  This might cause problems if your WebClient application uses the OpenEdge GUI for .NET. For more information, see the .NET Framework on page 109.

- Write to registry keys or files located in areas to which the non-administrator does not have access.

Note: Your application can check the installation type of a connected client through the `SESSION` system handle. For more information, see the Checking application registry access on page 52.

WebClient platforms

WebClient supports running applications as either 32-bit processes or 64-bit processes. We provide separate installations for 32-bit WebClient and 64-bit WebClient.

64-bit Windows can run either 32-bit WebClient or 64-bit WebClient. You can install both 32- and 64- bit WebClient of the same OpenEdge version on the same 64-bit Windows machine at the same time.

A 64-bit process can access more memory than a 32-bit process. However, unlike other ABL clients, WebClient cannot connect to databases, and does not require large amounts of shared memory.

You might also need the third-party functionality that is available only for 64-bit processes, for example, if the application requires a 64-bit DLL or other external component that only works with a 64-bit process.

Deciding the platform for your application

While selecting the platform for your application, consider the following:

- Applications that are written in ABL can be either 32-bit or 64-bit applications, as the r-code is platform-independent.
- Applications that use platform-neutral .NET assemblies can also be either 32-bit or 64-bit applications.
- Applications which make use of OCXs and/or DLLs need to be reviewed for platform-specific operations. In such cases, if you wish to provide a 64-bit application, you should find 64-bit alternatives for the 32-bit OCXs and DLLs, and change calls to them appropriately.

Define the WebClient application's platform

A WebClient application can support 32-bit, 64-bit, or both 32-bit and 64-bit. If you define an application for a particular platform, it indicates that the application must be run with a WebClient version for that platform only.

If you define an application as a 32-bit application, it will run only with 32-bit WebClient, or if you define an application as a 64-bit application, it will run only with 64-bit WebClient.

When installing 32-bit WebClient applications on a 64-bit Windows platform, the applications are by default installed in the same file system and registry where other 32-bit applications are stored. 64-bit WebClient applications are by default in the file system and registry where other 64-bit applications are stored.

Applications that are defined to support both 32-bit and 64-bit can run on either 32-bit or 64-bit versions of WebClient. It lets you support both platforms within a single WebClient application with a single set of deployment files and you do not need separate versions of the same application.

Applications that support both 32-bit and 64-bit have special handling. Consider the following:

- If the application is new, the WebClient platform required must match the Windows platform, for example, if installed on 32-bit Windows, 32-bit WebClient is required and if installed on 64-bit Windows, 64-bit WebClient is required. This allows a vendor to support both platforms with a single application and relieves the end-user of the decision of whether they should run 32-bit WebClient or 64-bit WebClient.
- If you have an existing 32-bit application, you can define the application for both 32-bit and 64-bit. The existing end-users of the application must manually migrate to the new platforms. End-users with 64-bit Windows running older versions of the application will have been running a 32-bit application with 32-bit WebClient. You can define how the end-users can move to 64-bit WebClient in the following ways:
  - Making it mandatory that the end-user to upgrade to 64-bit WebClient
  - Asking the end-user if they wish to use 64-bit WebClient
  - Continuing to use 32-bit WebClient

Note: Location of 32-bit and 64-bit applications is different as per 64-bit Windows standards, if you want to upgrade to 64-bit WebClient, you must uninstall the older version of the 32-bit application and install the application as 64-bit.

The WebClient Initializer (prowcini.exe) detects whether the packaged application supports both 32-bit and 64-bit platforms, or just one of them. The following pseudocode explains its operation:

```
if prowcapc format not supported by prowcini
  upgrade to newer webclient version
else
  if app already installed for the available platform(s)
    if win64 and prowcapc is 32-bit and 64-bit
      if installed app is 32-bit
        if (force uninstall) or (ask uninstall)
          uninstall and reinstall
        else /* continue as 32-bit */
          run/upgrade as normal
      else /* installed app is 64-bit */
        run/upgrade as normal
    else /* app is installed for this platform */
      run/upgrade as normal
  else if app is installed for wrong platform
    uninstall and reinstall
  else if 64-bit app and win32
    error, cannot install
  else /* new app */
    install app
```

Running 32-bit WebClient applications on 64-bit Windows

You can run your 32-bit applications on 64-bit Windows as the 64-bit Windows operating system continues to support 32-bit processes.

However, if you write your own external installer, you should be aware of registry locations and default file systems for 32-bit processes on 64-bit Windows to be able to write the correct key. For more information, see the Microsoft documentation on registry redirection.

By default, 32-bit applications are installed in a different directory than 64-bit applications, for example in North American Windows installations, 64-bit applications are installed in C:\Program Files, 32-bit applications are installed in `C:\Program Files (x86)` directories.

32-bit processes that access the HKEY_LOCAL_MACHINE registry have to access a sub node that is different from that of 64-bit processes. For example, for an Administrator install, the WebClient information is stored in the key `HKEY_LOCAL_MACHINE\Software\PSC\WebClient`.

Both a 32-bit process and a 64-bit process try to access this key, but the 32-bit process gets redirected to `HKEY_LOCAL_MACHINE\Software\Wow6432Node\PSC\WebClient`.

Personal installations use registry keys that are not redirected (for example, `HKEY_CURRENT_USER\Software\PSC\WebClient`), and file systems that are specific to the user (for example, `C:\Users\Username\AppData`).

Deploy a sample application using WebClient and IntelliStream

You can preform a test run of the application deployment process by using this article as a guide to deploy the OpenEdge sample application, Sports2020, with WebClient and IntelliStream.

Note: These procedures assume that you are installing as an Administrator.

Overview

In most business cases, application service providers need to make their application available to their clients for download. To do so, an ASP can use WebClient and IntelliStream to host an application on an application server so that it is available to their clients.

Use this guide as a template for deploying your own application with WebClient and IntelliStream. Keep in mind that there are a few steps that you do not need to complete during your own setup process, such as downloading can configuring the sample application from OpenEdge.

The process to deploy a sample application with WebClient and IntelliStream consists of:

1. Downloading the sample application.
2. Preparing the Web server
3. Setting MIME types for the application configuration file
4. Configuring the application on the application server machine
5. Setting up the application to launch from the Web server
6. Instal and run the application on the client machine

1. Download the sample application

To get started, download the WebClient files that support the Sports2020 sample application with IntelliStream from the Progress Software Download Center. After they are downloaded, you can find them in the Documentation and Samples file located in the `doc_samples` directory. Instructions for using these files are located in the following file:

```
Doc_and_Samples_Install\src\samples\webclient\dynappdel\instructions.txt
```

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

The instructions in this appendix use the following symbols, some of which might be defined as environment variables on your system:

- `%DLC%` (sometimes also indicated as *`OpenEdge_install_dir`*) refers to the OpenEdge installation directory on a particular machine. For example, the instructions might mention `%DLC%` on the end user-machine.
- `%DOCS%` (sometimes also indicated as *`Doc_and_Samples_Install`*) refers to the Documentation and Samples located in the `doc_samples` directory.
- `%ASWORKDIR%` refers to the work directory of an application server.
- `%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.

2. Prepare the Web server

Preparing the Web server involves:

- Setting up MIME types for the application configuration file
- Setting up the WebClient install and bootstrap mechanism

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.

3. Set 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 on page 95. Also, see the documentation for your Web server.

4. Configure the application on the application server machine

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

To set up the application on the application server machine:

1. Verify that an **OpenEdgedevelopment** product is installed on the application server machine. The person doing the installation, when presented with the **Setup Type** window, must choose the **complete installation**, as opposed to the typical installation or the custom installation.
2. Verify that an application server was installed and configured on the application server 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 Sports2020 application from the %DOCS%\src\sports2020\gui directory to %ASWORKDIR%.

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

5. Create a copy of the Sports2020 database on the application server machine. The copy can reside anywhere, but be sure to note its path.
6. Start an OpenEdge session and do the following:
   a. Connect to the Sports2020 database, entering its path
   b. Use the PROPATH Editor to add the application server work directory to the PROPATH.
   c. In the Application Compiler, go to **Options** > **Compiler,** change **Minimize R-code size** to **YES** and click **OK**. In the main window, click **Modify**. In the **File Specification** window, enter the application server's working directory, click **OK**, and compile the application.
7. Recursively copy the directories in the following table as specified, creating the destination subdirectory as necessary:

Recursively copy directory . . .	To location . . .
%ASWORKDIR%\gui	%APPROOTDIR%\appl
%DLC%\gui\adm2	%APPROOTDIR%\gui\adm2
%DLC%\jms	%APPROOTDIR%\jms
%DOCS%\src\sports2020\images	%APPROOTDIR%\appl\images

8. Copy the files in the following table as specified, creating the destination subdirectory as necessary:

Copy file . . .	To location . . .
%DOCS%\src\samples\webclient\dynappdel\sportspro.ini	%APPROOTDIR%
%DOCS%\src\samples\webclient\dynappdel\appsrvtt.d	%APPROOTDIR%
%DOCS%\src\samples\webclient\dynappdel\sportspro.wcp	%APPROOTDIR%
%DLC%\gui\adecomm.pl	%APPROOTDIR%\gui

Copy file . . .	To location . . .
%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 the previous step, as follows:

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

   ```
   http://yourservername/adapterName/adaptername?AppService=sports2020partition
   ```

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

   ```
   http://testhost/adapterName/adaptername?AppService=sports2020partition
   ```

   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 application server 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 **System Tasks Definition**. 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 the previous step.

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

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

11. Use OpenEdge Management or OpenEdge Explorer, or the `proserve` command to start a database server for the `sports2020` database running on the application server machine. For example, if the `sportspro2020` database resides at `%APPROOTDIR%\databases`, use a command like the following:

```
%DLC%\bin\proserve %APPROOTDIR%\databases\sports2020.db
```

12. Use OpenEdge Management or OpenEdge Explorer to configure the default `asbroker1` application server instance as follows:

    a. In the **General Options for Broker** tab, change the broker working directory to the application server work directory (`%ASWORKDIR%`), and add the service name `sports2020partition` to the **Appservice Name** list.

    b. In the **General Options for Server** tab, add the database connection parameter to **Server Startup Parameters**. For example if the `sports2020` database resides at `%ASWORKDIR%\databases`, enter the following string:

```
-db %ASWORKDIR%\databases\sports2020
```

13. Use OpenEdge Management or OpenEdge Explorer, or the `asbman` command to start the `asbroker1` application server instance. For example (using the `asbman` command):

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

5. Set up the application to launch from the Web server

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 application server 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.

6. Instal and run the application on the client machine

To install and launch the application on the client machine:

1. Use a appropriate Web browser 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 is installed.

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

2

Design Your End User's Experience

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

- How end users install WebClient and 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 affect your end users and how you develop and deploy your application.

For details, see the following topics:

- Installation process
- WebClient application installation
- Updates for your WebClient application
- Uninstalling your application
- Starting WebClient applications
- Provide documentation for your WebClient application

Installation process

You must provide a way for your end users to install WebClient on their machines.

You have the following options to provide a WebClient installation:

- Host an installation on a Web server
- Point your users to the WebClient installation page
- Provide your users with a flash drive that contains a WebClient installation program

OpenEdge provides the following WebClient installers:

- An InstallShield program that you can use to create an installation flash drive.

Note: You cannot customize the flash drive installation.

- A One-Click Install program that you can use to host a Web-enabled installation.

Note: Progress strongly recommends customizing the One-Click Install image to automatically install the application after installing WebClient, and to start the application after it is installed.

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

- With a flash drive installation, your end users use a setup program to install WebClient.
- With a Web-enabled installation, your end users go either to a Web page you provide or to the Progress Software Corporation Web site that provides a link for installing WebClient.

For more information about WebClient installers provided with OpenEdge and how to customize and prepare a WebClient installer for deployment, see Deploy a WebClient Application on page 69

Microsoft Windows installations

Installing WebClient and WebClient applications in Microsoft Windows includes additional considerations due to the User Access Control (UAC) feature. With UAC enabled, an Administrator user runs applications as a Standard (unprivileged) user. To use their additional privileges, an Administrator must specifically select the **Run As Administrator** option from a context menu. By running the Web browser as a Standard user for the installation process, an Administrator could install personal instances of WebClient and WebClient applications.

Note: Progress recommends that Administrators do not create personal instances for themselves in this manner.

Another feature that impacts installations of applications that have not been adapted is "virtualization." This feature causes those applications to redirect writes for protected areas of the registry and file system to a private area for each user. This virtual access to those protected areas enables unadapted applications to still function, while maintaining Windows security control over protected ares.

WebClient application installation

When you design your application, you must decide on the installer technology you want to use. You can deliver your application on a flash drive, over the Internet, or over a company intranet.

OpenEdge offers great flexibility for installing your application. You can use:

- The WebClient IntelliStream technology to install front-end components for your application and perform optional system tasks to satisfy basic installation requirements. This can include an ABL 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 flash drive for your application installation.

Note: You do not have to use the same technology to install your WebClient application as you do for installing WebClient.

Progress recommends that you provide a Web-based installation, either IntelliStream or a Web-enabled external installer, as the typical approach. For users without access to high-speed Internet connections, you can provide a flash drive installation. You should provide all updates over the Web.

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

- Use IntelliStream technology to install your application over the Internet or an intranet when you want to take advantage of the benefits that it offers. IntelliStream technology can:
  - Initially install only the required front-end components of your application, and then install other components only when a user needs the functionality
  - Optionally install your application from an application server with integrated security
  - Automatically start the application after installation
- 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 from a flash drive.
- If you need to provide both a flash drive and Web installation for different customers, you can:
  - Use an external install product, such as InstallShield, for flash drive installations, and use IntelliStream for installations over the Internet or an intranet. This design provides the benefits of IntelliStream features to those end users who have fast Internet access.
  - Use an external install product, such as InstallShield, for flash drive installations, and the One-Click technology for installations over the Internet or an intranet.

IntelliStream

You can use IntelliStream to have WebClient install your application automatically with minimal interaction by your end users.

IntelliStream includes an installer, built into WebClient, that installs and updates your application by downloading the required application components or component updates.

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 prompts for security information and information about the components they want to download. For more information, see Design for Security on page 41 and Deploy a WebClient Application on page 69

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 ABL installation 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.

External WebClient application installer

If you decide to use an external installer, such as InstallShield or a similar program, Progress 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 uninstallation program for the application, typically using the same technology as the installation

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

Updates for your WebClient application

You must choose a technology to update your application.

To provide updates, you can use:

- **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 Corporation recommends IntelliStream for updates, regardless of the technology used 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 installation for updates.

Uninstalling your application

Caution: If you use IntelliStream to install or update your application and you plan to uninstall the WebClient, then you must uninstall your application **before** you uninstall the WebClient.

Typically, installer technology includes an uninstallation feature. For example:

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

Since they cannot write to the correct registry keys, users without Administrator privileges cannot uninstall an Administrator installation. Since they cannot see another user's personal installation, no user can uninstall another user's personal installation.

If an uninstall fails, you must perform a manual uninstall of WebClient.

Starting WebClient applications

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

- 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 shortcut that starts the application directly from the Windows desktop after the application is installed.

For more information, see Host your WebClient Deployment on page 37 and Deploy a WebClient Application on page 69

Provide documentation for your WebClient application

Your deployment plan should include composing and providing documentation for your end users.

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 with the WebClient Application Manager that is installed with WebClient. For example, you might need to provide instructions for setting user-specific startup parameters required by your application.

3

Host your WebClient Deployment

When determining how to host your WebClient deployment, you must consider how you will host your configuration files and code bases.

Whether you use IntelliStream technology or another platform to download your configuration files, the WebClient requires that you download the application configuration file from a standard *Internet-based server*, such as a Web server or a file server.

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

When you deploy a WebClient application, you must choose the servers to host your application.

For details, see the following topics:

- Hosting application configuration files
- Hosting WebClient codebases

Hosting application configuration files

Whether you use IntelliStream technology or not, the WebClient requires you to download the application configuration file from a standard *Internet-based server*, such as 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 Web server with the application server, you might host the configuration file on this Web server. Or, if you have already 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.

After selecting a server, you describe the server to WebClient through the WebClient Application Assembler's **Configuration File Locator Definition** window at during application-deployment.

When WebClient downloads the application configuration file from its server, the download uses a communication protocol based on the server's type. If the application configuration file resides on a Web server, downloads uses HTTPS. HTTPS encrypts the download, which increases security.

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

The URL you enter in the Application Assembler's **Configuration File Locator** window tells WebClient which protocol to use.

The WebClient can be configured to execute even if the configuration file server is not available. See Defining applications in WebClient Application Assembler on page 80 for more information.

Hosting WebClient codebases

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

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

After selecting a server, you describe the server to WebClient through the Application Assembler's **Codebase Locator Definition** window at application-deployment.

Hosting the codebase on an internet-based server

Hosting the codebase on an Internet-based server has the advantage of allowing the application configuration file and the codebase to be managed consistently. If the application configuration file resides on a Web server, downloads use HTTPS. HTTPS encrypts the download, which increases security.

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

Hosting the codebase on an application server

Hosting the codebase on an application server has both advantages and disadvantages.

Advantages

- If the codebase can reside on the same application server that hosts the business logic, administrative tasks are simplified.
- If the application server that hosts the business logic already has security (such as user IDs and passwords), placing the codebase on that application server protects the codebase with the same security, at little or no additional cost.

Disadvantages

Hosting the codebase on an application server has the disadvantage that it adds additional load to the application server. You might need to configure an additional application server instance to handle the additional load of hosting the codebase and to avoid performance degradation.

If application components reside on an application server, downloads can use HTTPS or any of the four variations of application server protocol. The URL you enter in the Application Assembler's Codebase Locator window tells WebClient which protocol to use for codebase downloads. To choose among these protocols, consider the following:

- Using HTTPS protocol with an application server requires using the adapter.

- The HTTPS protocol encrypts downloads over the Internet, which increases security.
- The application server protocols do not use the adapter.
- The application server and application server DCS protocols use TLS tunneling to encrypt downloads over an intranet, which increases security.
- The application server and application server DC protocols do not encrypt downloads, but are faster than their TLS-based counterparts.

4

Design for Security

Your application should combine tight security with a clean user interface that does not require end users to enter their log-on information multiple times. There are a number of ways to provide security, including digitally signing files, using user IDs and passwords, caching information, and using secure communication protocols.

Because of the ever-present security threats when using the Web to distribute your applications, you must secure the files you distribute.

For details, see the following topics:

- Digitally signing files
- Protecting servers with user IDs and passwords
- Single sign-on and security caching
- Using WebClient logon dialogs in your applications
- Choosing secure communication protocols

Digitally signing files

When deploying an application, you can digitally sign each cabinet file and have the end user verify each digital signature when the file downloads.

This approach provides end users with the assurance that each file has:

- **Authenticity** — Assurance that you prepared the file
- **Integrity** — Assurance that nobody has tampered with the file

Note: Progress strongly encourages the application deployer to digitally sign each file to be downloaded.

The WebClient Application Assembler uses Microsoft Authenticode technology to sign the application cabinet files digitally. It requires access to the `signtool.exe`, `pvk2pfx.exe`, and optionally `makecert.exe` tools. You need `makecert.exe` if you wish to create a mock public-key certificates for testing. These tools are part of Microsoft Windows SDK.

Making Microsoft signing tools available to OpenEdge

The WebClient Application Assembler for OpenEdge requires you to identify the location of these signing tools so that it can make use of the tools when signing the cabinet files digitally.

The WebClient Application Assembler requires you to enter the path to the signing tools when generating the application. If you make the signing tools available to the WebClient Application Assembler, the WebClient Application Assembler can detect the location of the signing tools and pre-populate this path for you.

To make the signing tools available to OpenEdge:

1. Download and install the latest Microsoft Windows SDK from the Microsoft site. If you do not wish to perform a full install, make sure you install `signtool.exe`, `pvk2pfx.exe` and optionally `makecert.exe`.
2. Set one of the following environment variables for OpenEdge to identify the location of the tools:
   - `"%WindowsSDKDir%` to identify the Windows SDK installation directory. The WebClient Application Assembler looks for the signing tools in `%WindowsSDKDir%\bin`.
   - `%OE_SIGNING_DIR%` if you have more than one Windows SDK installed, or if setting the `%WindowsSDKDir%` environment variable might conflict with other applications.

How WebClient uses digital signatures

If you want to digitally sign each cabinet file for your WebClient application and you want the end user to verify the digital signature of the downloaded files you must assign a series of keys and certificates.

To digitally sign a cabinet file, 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, 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.

Getting a private key, public key, and public-key certificate

Use the following procedure to obtain private and public keys.

To get a private key, a public key, and a 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. Install the software that generates and securely stores public keys and private keys on your system.

   You can typically get the software from Microsoft or you can 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.
5. Wait for the CA to verify your identity.

   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 the same software and the same Web site that you used to apply.

   You can have digital certificates issued by multiple CAs for a single public/private key pair.

   The digital certificates are stored on your system in the same named certificate location as the one used for the initial public/private key generation.

Defining an application as signed

With your private key, a public key, and a public-key certificate, you can digitally sign an application.

Ensure that you have your private key, public key, and public-key certificate

Use the following procidure to digitally sign an application.

1. Open the **Generate** dialog of the WebClient Application Assembler. For more information on this dialog, see Files generated by the WebClient Application Assembler on page 88.
2. Click **Security**.
3. In the **Digital signature for generated files** section, select one the following options:

   - **From registry**: Select if the digital signature information resides in the registry.
   - **From file**: Select if the digital signature information resides in a certificate file, and you have a separate private key file.
   - **From PFX File**: Select if the digital signature information resides in a PFX file.

4. Enter the path for the signing tools in the **Path To Signtool.exe**.

   If you have set the environment variables OpenEdge populates the path in this field. You can also change the pre-populated value.

   When you define an application as signed and you generate the application, the Application Assembler adds a copy of your public-key certificate to each cabinet file and digitally signs them.

How your public-key certificate is delivered 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.

3. 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.
4. Displays the information on the certificate and asks whether the end user trusts it.

   If the end user says no, the process aborts.
5. Optionally stores your public-key certificate in the digital certificate store of Internet Explorer.

Creating test public-key certificates

WebClient includes a batch file, `MakeTestCert.bat`, that makes it easier for you to create mock public-key certificates for testing. For more information, see the comments in the file, which resides at `OpenEdge-Install-Directory\bin`. However, if you want to use `maketestcert.bat` to generate a test certificate, you must make the Microsoft signing tools available to OpenEdge. For more information, see Making Microsoft signing tools available to OpenEdge on page 42.

Protecting servers with user IDs and passwords

You can require user IDs and passwords for each Internet-based server and application server accessed by WebClient or the WebClient application. Although user IDs and passwords are not required, using them strengthens security.

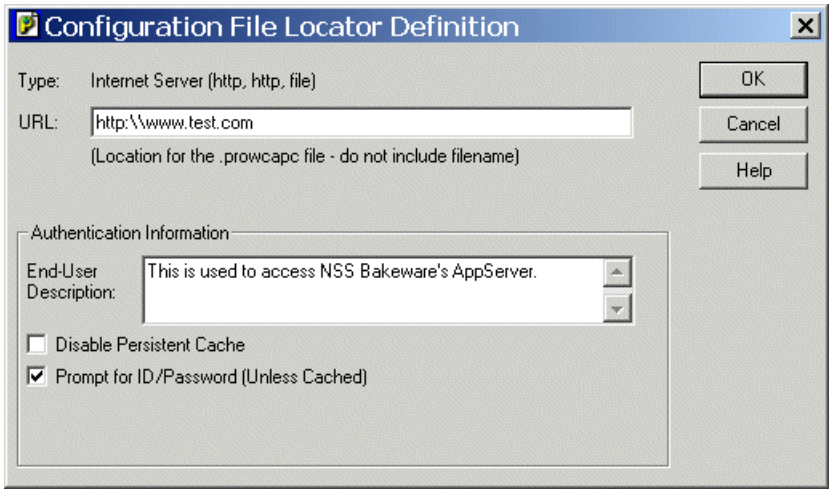
Progress recommends that you seriously consider protecting each server accessed by WebClient or the application with a user ID and password.

Prompting the end user for authentication information

Instead of embedding authentication information in a server URL, you can have WebClient prompt the end user for them when they are needed. How you enable the prompting depends on whether you are configuring an application-configuration-file server or a codebase server.

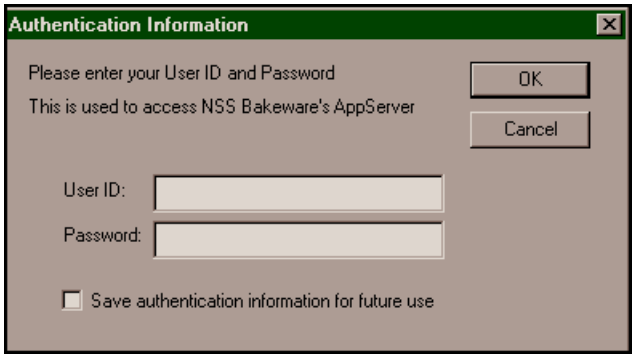
Enabling authentication for an application-configuration-file server

To enable authentication for an application-configuration-file server, go to the Application Assembler's **Configuration File Locator Definition** window. In the **Authentication Information** group box, fill in the **End-User Description** field and check the **Prompt for ID/Password (Unless Cached)** box:



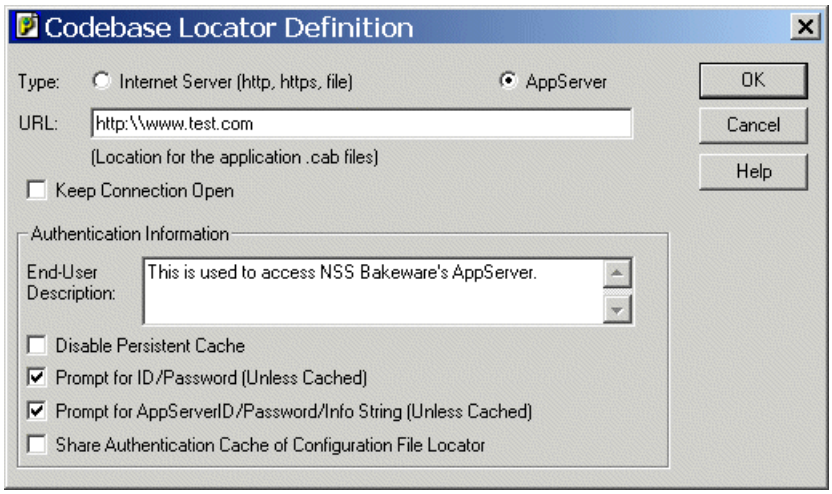
Note: WebClient uses this information only if the end user starts an already-installed WebClient application by using a shortcut. For more information on starting an already-installed application with a shortcut, see Deploy a WebClient Application on page 69

Enabling this option displays the following authentication prompt to the end user:

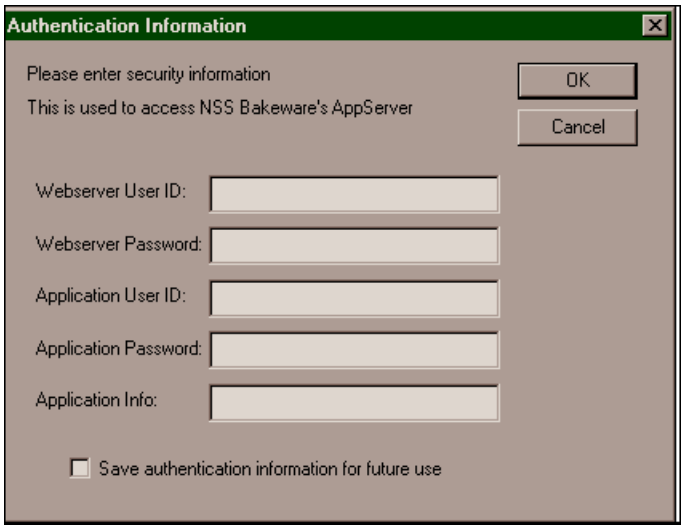


Enabling authentication for a codebase server

To enable authentication for a codebase server, go to the Application Assembler's **Codebase Locator Definition** window. In the **Authentication Information** group box, fill in the **End-User Description** field. Then, check the **Prompt for ID/Password (Unless Cached)** box, and finally, if the codebase server is an application server, check the **Prompt for AppServerID/Password/Info String (Unless Cached)** box:



Enabling this option displays the following authentication prompt to the end user:



Single sign-on and 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 authentication.

When the end user provides authentication information to connect to a server and to download files, WebClient can cache the authentication information and make it available to the application. Similarly, when the end user provides authentication information at an application prompt 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 keeps end users from being 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; for instance, when the same application server handles both downloads of application components and running business logic for the application. For example, if WebClient downloads application components before the application accesses the business logic, the the application can automatically connect to the application server to access the business logic using the same authentication information. Similarly, if the application accesses the business logic before WebClient downloads application components, the application can make authentication information available to WebClient when it connects to the application server to download application components.
- Connecting to multiple servers that use the same authentication information. Suppose the application configuration file and the application components are on different servers, but you set the same authentication information for both servers. You can direct WebClient to make the configuration file's authentication information available to the application as codebase authorization information (but not the other way around).

  Note: Authentication information is always encrypted when stored.

  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. To override this default behavior, the end user must specifically request the *persistent cache*.

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 can choose to persistently cache their authentication information for particular servers. After that, anyone 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.

Using WebClient logon dialogs in your applications

Consider the following scenario. Your application uses the same application server for application components downloads and for accessing business logic, and suppose further that an application needs to access the application server before the WebClient downloads any **As Needed** application components from the application server. An application can determine this because the security cache does not contain authentication information such as user IDs or passwords. As a result, the application must prompt the end user for the logon information. To do so, your application can display the security dialogs that WebClient uses, providing the end user with a consistent interface.

For information on calling the authentication dialogs that WebClient uses from the application, see the Using WebClient authentication dialogs on page 53.

Choosing secure communication protocols

To guard against the inherent security problems associated with network traffic, and to provide a greater level of security for your applications, you can encrypt an application's configuration file and application-component downloads by using a secure communication protocol.

For example, you can use HTTPS to encrypt network traffic with a combination of symmetric and asymmetric ciphers. For more information on symmetric and asymmetric ciphers, see the Digitally signing files on page 41.

Secure protocols that you can use, depending on the server type (Web server, file server, or application server) where the files are hosted, include:

- HTTPS (requires the use of an adapter)
- Application server S (codebase downloads only)
- Application server DCS (codebase downloads only)

For more information on communication protocols for application configuration file and codebase downloads, see Host your WebClient Deployment on page 37.

5

Develop Your Application

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

For details, see the following topics:

- Using the CODEBASE-LOCATOR handle and its attributes
- Checking application registry access
- Managing application server connections
- Implementing single sign-on
- Using WebClient authentication dialogs
- Using URLs in PROPATH in WebClient for image file downloads
- Compiling the WebClient application

Using the CODEBASE-LOCATOR handle and its attributes

Once the application configuration file contains the codebase server description, WebClient makes the description available, to the client at run time, as attributes of the `CODEBASE-LOCATOR` handle.

All the attributes listed are read-only.

Basic attributes

The following table lists the basic attributes of the CODEBASE-LOCATOR handle.

Table 1: Basic attributes of the CODEBASE-LOCATOR handle

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.
LOCATOR-TYPE	Character	AppServer or InternetServer.
NEEDS-APPSERVER-PROMPT	Logical	If TRUE, enables prompting for a user ID, password, and info string for accessing an application server. Applies only if LOCATOR-TYPE is AppServer.
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 adapter for a codebase residing on an application server.
PERSISTENT-CACHE-DISABLED	Logical	If TRUE, the user is not given the option of saving authentication information past the end of the session.
URL	Character	URL of the server.

Security-cache attributes

The following table lists the attributes of the CODEBASE-LOCATOR handle that relate to the security cache.

Table 2: Security-cache attributes of the CODEBASE-LOCATOR handle

Name	Data type	Description
APPSERVER-INFO	Character	Info string to use for connecting to the application server. Unused if LOCATOR-TYPE is InternetServer.
APPSERVER-PASSWORD	Character	Password for connecting to the application server. Unused if LOCATOR-TYPE is InternetServer.
APPSERVER-USERID	Character	User ID for connecting to the application server. Unused if LOCATOR-TYPE is InternetServer.

Name	Data type	Description
KEEP-SECURITY-CACHE	Logical	If TRUE, WebClient saves the security-cache attributes and this attribute, restoring them the next time the end user reruns the application. The default is FALSE. Set to TRUE if the end user sets the **Save Authentication Information for Future Use** toggle box. Can also be set to TRUE by the application, depending on the security dialog used.The application might set this before WebClient does. For more information, see the Implementing single sign-on on page 53. **Note:** The PERSISTENT-CACHE-DISABLED attribute must be FALSE to use this attribute.
URL-PASSWORD	Character	Password for accessing the server. If LOCATOR-TYPE is InternetServer, the password is for accessing a codebase on an Internet-based server. If LOCATOR-TYPE is AppServer, the password is for accessing the adapter for a codebase on an application server. Unused if the URL does not take a password (for example, for a URL of type FILE).
URL-USERID	Character	User ID for accessing the server. If LOCATOR-TYPE is InternetServer, the user ID is for accessing a codebase on an Internet-based server. If LOCATOR-TYPE is AppServer, the user ID is for accessing the adapter for a codebase on an application server. Unused if the URL does not take a user ID (for example, for a URL of type FILE).

Additional attribute

The following table lists an additional attribute of the CODEBASE-LOCATOR handle.

Table 3: Additional attribute of the CODEBASE-LOCATOR handle

Name	Data type	Description
SERVER	Handle	Handle to the application server that 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 application server connection for accessing business logic as for downloading as-needed components. To do this, the application sets the SERVER attribute once the application server 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 application server'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

Checking application registry access

If users can install your application as either Administrator or personal instances, your application might need logic to alter where it reads and writes registry information based, on the type of installation. To enable this, the SESSION system handle has a new attribute, WC-ADMIN-APP. The following code snippet is an example of such logic:

```
DEFINE VARIABLE cAppKey AS CHARACTER NO-UNDO.
IF SESSION:WC-ADMIN-APP THEN
    cAppKey = "HKEY_LOCAL_MACHINE".
ELSE
    cAppKey = "HKEY_CURRENT_USER".
LOAD "MyEnvironment" BASE-KEY cAppKey.
```

Managing application server connections

To download as-needed application components from a codebase residing on an application server, WebClient, by default, creates a new connection to the application server, 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 necessary, the 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 enable this option, in the Application Assembler's **Codebase Locator** window, check the **Keep Connection Open** toggle box.

WebClient can also download as-needed application components over an application server connection that is already open. For example, if your business logic and the codebase reside on the same application server, you could use a connection that is already established to access the business logic. To enable this option, just after you connect to the application server, 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: Progress recommends that an application server connection used for ASYNC requests not be used for downloading application components.

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 application server if both the codebase and the business logic reside on the same server.
- The application deployer checks the **Share Authentication Cache of Configuration File Locator** toggle box on the Application Assembler's **Codebase Locator Definition** dialog. After the end user enters values for **URL-USERID** and **URL-PASSWORD** to download the configuration file, WebClient makes these values (or those that already reside in the persistent cache) available through the CODEBASE-LOCATOR attributes. These can be used to connect to the application server through an adapter if the same Web server hosts the adapter 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 prompting the end user again. For more information on these attributes, see the Using the CODEBASE-LOCATOR handle and its attributes on page 49.

Using WebClient authentication dialogs

Consider the following scenario. Your application accesses an application server and the following are true:

- The same application server contains application components for download and business logic for your application.
- The application detects that the security cache does not contain authentication information.

Note: These conditions occur when WebClient has not yet downloaded any as-needed application components from the application server.

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

These routines reside, as compiled .r files, in the WebClient installation at OpenEdge-Install-Directory\wcadd. If the CODEBASE-LOCATOR handle contains only some of the required values, these values can be passed into these routines as default values, allowing the user to fill in the remaining values.

When using the authentication dialogs, consider the following:

- To make the dialogs look exactly the same as when WebClient invokes them, obtain the prompt input parameter from CODEBASE-LOCATOR:END-USER-PROMPT.
- 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.
- After calling an authentication dialog, the application is responsible for setting the appropriate CODEBASE-LOCATOR attributes with the information obtained from the end user.

Displaying the adapter authentication dialog

To display WebClient's adapter 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 application server itself.

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

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

Displaying the application server authentication dialog

To display WebClient's application server 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 application server, or if they share the same authorization information and authentication is required:

- Only on the application server and not on the adapter Web server (or you are not using the adapter)
- At the adapter and at the application server, but the adapter 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 with a partially qualified pathname that starts with the wcadd directory, as in the following code fragment:

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

Displaying the adapter and application server authentication dialog

To display WebClient's adapter and application server 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 adapter, and authorization is required by the adapter server and by the application server.

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

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

Using URLs in PROPATH in WebClient for image file downloads

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.

Using the SEARCH function

This is the syntax of the SEARCH function:

```
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.

Using the LOAD-IMAGE( ) method

This is the syntax of the LOAD-IMAGE( ) method:

```
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.

PROPATH error handling

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

URL specification errors

If the PROPATH contains an ill-formed URL for example, such as one containing a percent symbol (%) the following processing occurs:

- When 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.
- When the URL is a 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:

- When 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.
- When the URL is a 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:

- When 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.
- When 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 all raise error 9368, where *server-name* is the host name of the Web server, as shown:

    ```
    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 the 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.
    ```

Compiling the WebClient application

When you compile ABL procedures for your WebClient application, the compiler generates an r-code signature value, a fixed-length value that is unique for the exact contents of a given source code file.

If the source code of the procedure changes recompiling the file generates a different r-code signature value. Later, when you deploy a new version of your application using the WebClient Application Assembler, the Assembler compares the existing and new r-code signature values. If the values are different, the Assembler assumes that the new file has changed and marks it as an update of the old file.

6

Your End Users' Experience

After the WebClient and the application are installed, they might need information about the WebClient Application Manager, which enables them to view and change information about the WebClient and application installation.

Before they can do anything, end users need installation information and application instructions.

For details, see the following topics:

- Preparing documentation for end users
- WebClient Application Manager

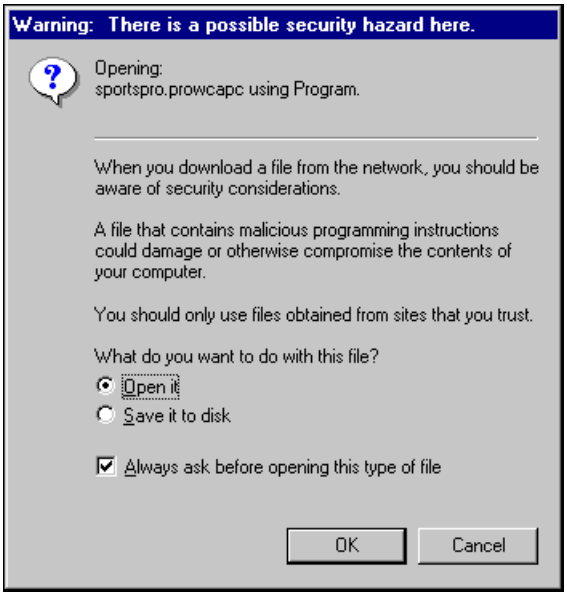
Preparing documentation for end users

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

- You should inform users which installation types you support. If your application installer includes any of the System Tasks that require Administrator privileges, you should specify what they are. If your end users might take the option of granting specific write privileges to non-Administrators, you must detail exactly what privileges are required. For more information on these topics, see the WebClient installation types on page 14.
- If your application uses the OpenEdge GUI for .NET, you should inform users what the appropriate versions of the .NET framework are for running your application. Explain what they need to do, based on the choice you made about including a .NET framework installer. If your WebClient installer launches a .NET Framework installer, tell your end users to accept the EULA and install the framework.For more information, see the .NET Framework on page 109.

- 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 the WebClient Application Manager on page 61.
- 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, users must choose the **Open it** option to install or start the application:



  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.

- 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 prompts them to grant a certificate that allows the player to be installed. 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 on page 61.
- If you use IntelliStream to install your application, tell your end users that, when uninstalling, they must first uninstall your WebClient application and then uninstall the WebClient itself.
- 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 they 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 Design for Security on page 41

WebClient Application Manager

WebClient Application Manager enables end users to configure and manage their use of a WebClient application and to provide information for troubleshooting. It helps you to access and display both 32-bit and 64-bit WebClient versions and WebClient applications, irrespective of the platform of the executable file. For 32-bit WebClient, WebClient Application Manager (prowcam.exe) is a 32-bit executable file, and for 64-bit WebClient, prowcam.exe is a 64-bit executable file.

Only 32-bit WebClient versions and applications can exist on a 32-bit Windows platform. Both 32-bit and 64-bit WebClient versions and applications can exist on a 64-bit Windows platform.

End users should only run the manager with your supervision, and usually for the following reasons:

- To specify post-installation WebClient startup parameters
- To specify the 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

The manager is installed as part of the WebClient installation. Although end users might have multiple versions of WebClient installed on their machine, there is always only one version of the manager for each installation type. You would have a single WebClient Application Manager from all your Administrator instances of WebClient, and another WebClient Application Manager from all your personal instances. The manager for each type of installation is from the most current version of WebClient for that installation type. However, although the manager is part of most current WebClient installation, it can display information about all instances installed on a user's machine.

Using the WebClient Application Manager

The WebClient Application Manager provides end users with a way to view and edit information about the WebClient applications installed on their system. The manager gives access to information about the WebClient instances and instances of WebClient applications. For example:

- **WebClient instances** — Lists the WebClient versions installed on the user's system and enables the user to modify general options for WebClient execution
- **Applications instances** — Lists the WebClient applications installed on the user's system by vendor name and application name, and enables the user to review, set, or change execution options for a selected application

The user can start the manager 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 **WebClient** program folder. When this shortcut is used, the manager appears with the **WebClient** tab selected. For more information, see the WebClient tab on page 62.
- **Application** — You can make this mode available to the end user through a shortcut that you set up during the application installation. When this shortcut is used, the manager appears with the **Applications** tab and current application selected. For more information, see the Applications tab on page 63

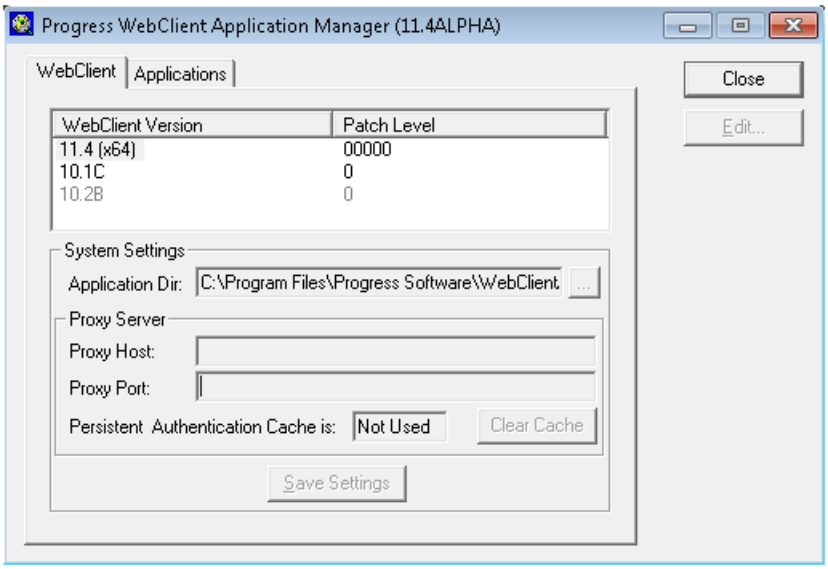
  For more information about setting up the shortcut for this mode, see the External installer on page 74, and the System tasks on page 74.

WebClient tab

If the user runs WebClient Application Manager in WebClient mode, the manager appears with the **WebClient** tab selected. When a user runs the manager, it now shows all Administrator instances of WebClient on the machine, but only that user's personal instances of WebClient.

The manager differentiates between the two installation types by showing personal instances in grey text. On 64-bit Windows platforms, the manager identifies the 64-bit WebClient versions by adding a (x64) suffix to the WebClient version.

In this example, the WebClient Tab displays a 64-bit Administrator install of 11.4 WebClient, a Personal install of 10.2B WebClient and an Administrator install of 10.1C WebClient.



The **WebClient** tab includes the **System Settings** controls, enabling the end user to view and edit the current settings for the application directory, whether or not the persistent authentication cache is enabled, and the host and port of a proxy server if the user's system must connect to your Web server through an HTTPS-based proxy server (firewall). If the user provided the proxy host and port during WebClient installation, these fields display them as the initial values. If the current user does not have write access for the selected instance, the manager does not enable the values for editing.

The user can modify these settings as follows:

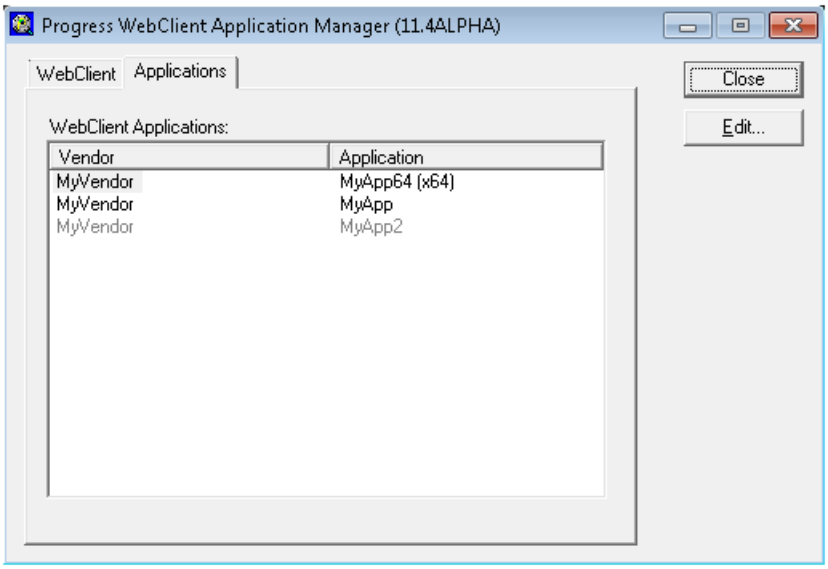
- **Application Dir** — Allows users to change the directory on their system where a WebClient application is installed by default. The users would change this directory when there is not enough space on the drive initially specified by the user for the application installation.
- **Proxy Host** — Allows 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.
- **Persistent Authentication Cache** — Indicates if the proxy server authentication information is cached. You can use the **Clear Cache** button to clear the registry entries for the associated security cache.

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

Applications tab

The **Applications** tab displays a list of all WebClient application instances currently installed on the user's system. The manager identifies the applications by the vendor name and the application name. 64-bit applications also have a (x64) suffix added to the application name.

In the following example, the **Applications** tab displays an Administrator instance of the **MyApp** application, a 64-bit Administrator instance of the **MyApp64** application, and a personal instance of the **MyApp2** applications:



To make changes to an application's options, or to view the WebClient application log file, select the application in the list and click **Edit**.

Edit dialog

The **Edit** dialog 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. If the current user does not have write access for the instance, the manager does not enable the values for editing.

The **Edit** dialog title reflects the name of the selected application. The tabs that appear in the **Edit** dialog vary depending on whether or not you use the WebClient Application Assembler to define the deployment. For all applications, the following tabs appear in the **Edit** dialog:

- **General** — Lists general information about the selected application, and allows the user to set or change certain execution options. See the General tab on page 64.
- **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 on page 66.

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

- **Security** — Displays security information about the selected application. For more information, see the Security tab on page 66.
- **Component** — Displays information about the application components that are installed on the end users system. For more information, see the Component tab on page 67.

Clicking **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 troubleshoot 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 deployment. The following figures show the difference in the information displayed.

Figure 3: General tab without WebClient Application Assembler info

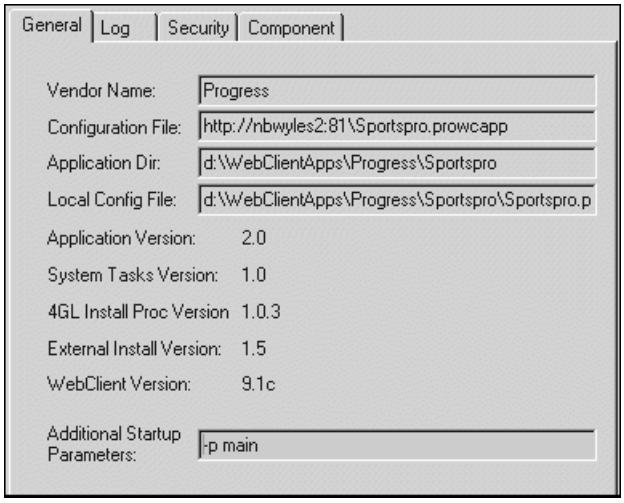
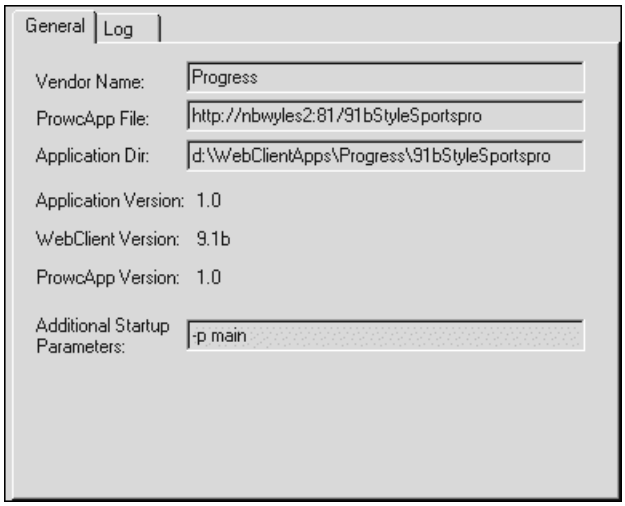


Figure 4: General tab with WebClient Application Assembler info



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, which the application 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 OpenEdge startup parameters using the syntax you provide to them.

  Your users might use these parameters 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 with the WebClient Application Assembler, as shown in Figure 3: General tab without WebClient Application Assembler info on page 64, 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 with the WebClient Application Assembler, as shown in Figure 4: General tab with WebClient Application Assembler info on page 64, the following additional fields appear in the **General** tab:

- **Configuration File** — URL for the configuration file used to install and start the application. This file (`-prowcapp`) specifies the name of the WebClient configuration file (`.prowcapp`) used when deploying an application. The WebClient Application Assembler creates and downloads this configuration file to the end user's machine each time a WebClient application starts. This file provides all the information necessary for the WebClient application. You refer to this file with the following syntax:

  ```
  -prowcapp filename
  ```

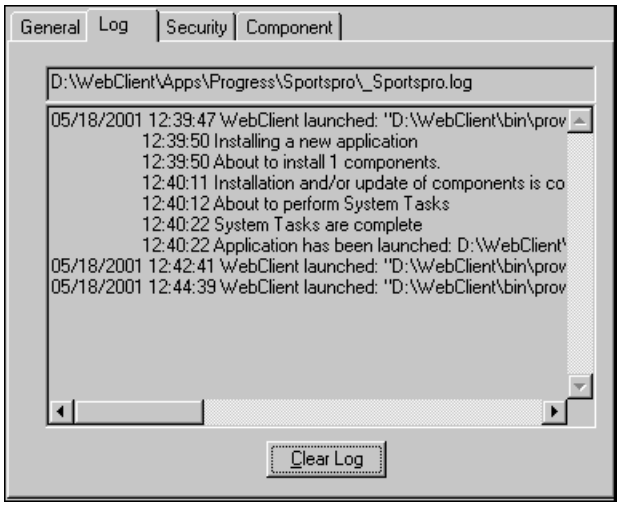
  Where *`filename`* represents the pathname of the application configuration file (`.prowcapp`).

- **Local Config File** — Pathname of the configuration file cached on the end user's machine.
- **System Tasks Version** — Current version of system tasks as defined in the WebClient Application Assembler.
- **ABL Install Proc Version** — Current version of the ABL 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 the user enters.

Log tab

The **Log** tab displays the contents of the application log file generated by WebClient:



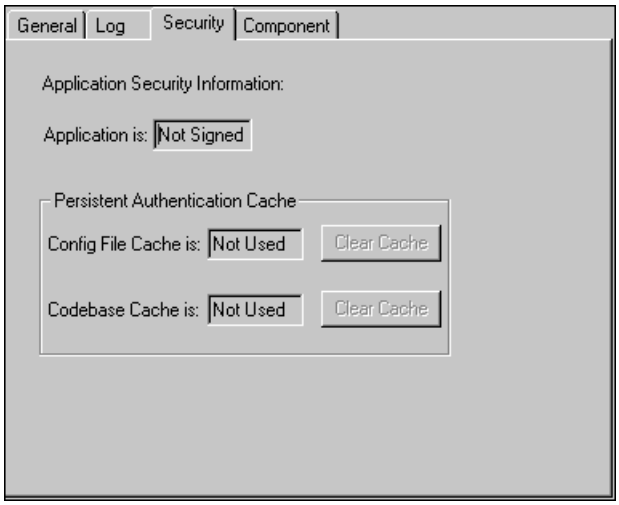
WebClient writes status messages to a log file for you to troubleshoot problems. The log file is created in the **Application Dir** you specified.

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: Clearing the log is **immediate and permanent**. Choosing the **Cancel** button does not undo deletion of the application log file.

Security tab

The **Security** tab displays authentication information about the selected application. This tab appears only when you use the WebClient Application Assembler to define the deployment:



The following fields appear on the **Security** tab:

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

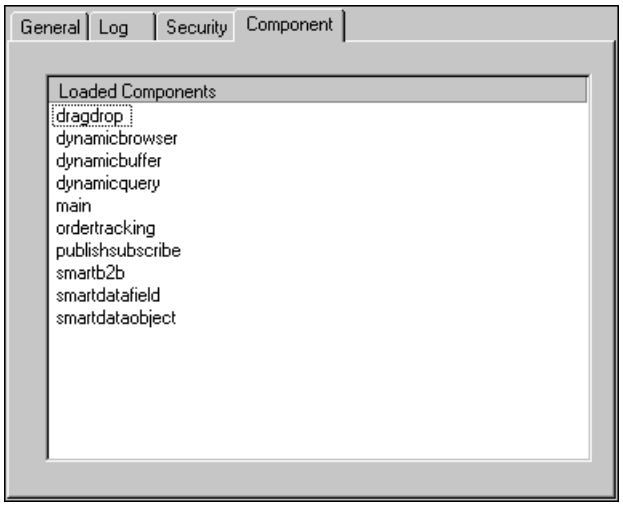
- **Persistent Authentication Cache** — Indicates if the **Config File Cache** and the **Codebase Cache** are being used. The possible values are:
  - **Used** — Authentication information is cached
  - **Not Used** — Authentication information is not cached

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 provides a list of application components that have been downloaded to the end user's machine. This tab appears only when you use the WebClient Application Assembler to define the deployment:



7

Deploy a WebClient Application

With the Application Assembler, you describe the deployment configuration and generate the deployment files. The Deployment Packager enables you to customize a deployment after generating the deployment files.

*Learn about how to deploy a WebClient application by using PRO*Tools: WebClient Application Assembler and WebClient Deployment Packager.*

For details, see the following topics:

- WebClient Application Assembler prerequisites
- Installation types
- Choosing an installation method
- Defining applications in WebClient Application Assembler
- Files generated by the WebClient Application Assembler
- Customizing deployment configurations
- Hosting the application on a server
- Installing WebClient
- Bootstrapping a WebClient application
- WebClient application updates
- WebClient and OpenEdge GUI for .NET

WebClient Application Assembler prerequisites

Before you run the WebClient Application Assembler, consider how you want to deliver and install the application on your end users' machines. These decisions determine the information that you provide in the various WebClient Application Assembler windows.

Required input files

When you run the WebClient Application Assembler, you must have a root directory for your application, and it must contain:

- All files that the application is comprised of, such as `.r` files, `.pl` files, image files, and more.
- All files that require special install processing for your application. This includes third-party files such as system `.dll` files, image files for shortcuts, and an initialization file.
- All files placed in the components must reside under the **Application Root Directory** that you specify in the **General** tab of the WebClient Application Assembler.

  Note:

  If your application uses the OpenEdge GUI for .NET, all assemblies used by the application must be in the assemblies directory under the application root directory.

- 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.

  Note: The WebClient Installation does not include any code from the OpenEdge Application Development Environment (ADE), such as, the `adecomm` code, the ADM code, or any of the controls included with the OpenEdge installation: `CSComboBox`, `CSSpin`, and `PSTimer`. If your application uses any of this code, you must package it into your application installation with the rest of your application code.

Installation types

WebClient installation types on page 14 describes how an end user can install WebClient either as a Administrator instance or a personal instance. Your WebClient deployment might support only Administrator installs, or it might support both Administrator and personal installs. Because the installer determines what type of installation to perform based on the person's privileges, the installer always creates an Administrator instance for an end user with Administrator privileges. Each installation type that you support has requirements for you to consider.

Note: Progress recommends that you do not have both installation types of the same WebClient application on a single machine. If several users on the same machine use a WebClient application, a single Administrator installation uses less resources.

Administrator installations

To start the install process, the WebClient installer first checks if the user can perform an Administrator install.

The installer checks if either of the following conditions is true:

- The user is a member of a group that has Administrator rights.
- The user has write-access to the `HKEY_LOCAL_MACHINE\SOFTWARE` registry key, the `C:\Program Files` directory, and the `C:\Windows\system32` directory.

Granting write privileges

For security reasons, many companies do not want to grant everyone Administrator privileges. To use an Administrator instance, an Administrator should grant all users of your WebClient application write access to certain registry keys and directories. Granting these privileges might be a problem in some deployment environments. The following table lists the necessary write privileges for installing and using an Administrator installation.

Table 4: Administrator installation required permissions

Task	Requires write access to ...
Install an application	`HKLM\SOFTWARE` to create a <*vendor*>\<*app*> subkey `C:\Program Files\Progress Software\WebClient` Write access is only needed when installing both WebClient and a WebClient application.`C:\Program Files\WebClientApps`
Run the application	`HKLM\SOFTWARE\` <*vendor*> \ <*app*> and its subkeys `C:\Program Files\WebClientApps\`<*vendor*>\<*app*> Unless temp files have been redirected to a writeable directory.
Download an "As needed" component	`HKLM\SOFTWARE\` <*vendor*> \ <*app*> `C:\Program Files\WebClientApps\` <*vendor*> \ <*app*> and its subdirectories
Update to a new version	`HKLM\SOFTWARE\` <*vendor*> \ <*app*> `C:\Program Files\Progress Software\WebClient` Write access is only needed if the update includes both WebClient and a WebClient application. `C:\Program Files\WebClientApps\` <*vendor*> \ <*app*> *and its subdirectories*
Uninstall the application	`HKLM\SOFTWARE\` <*vendor*> \ <*app*> and its subkeys `C:\Program Files\Progress Software\WebClient` Write access is only needed if the uninstall includes both WebClient and a WebClient application. `C:\Program Files\WebClientApps\`<*vendor*>\<*app*> and its subdirectories

An Administrator might avoid granting some of these privileges by setting up batch jobs to handle updates for the users. The Administrator must also arrange the download of all "As needed" components before the end users try to access them.

Personal installations

If the end-users' company cannot support Administrator instances, then each end user must install their own personal instance. When the WebClient installer determines that an end user cannot perform an Administrator install, the installer automatically switches to performing a personal install.

The following table lists the default values that the installer supplies during a personal installation. If the user enters a different value, the installer verifies that the user has write privileges for the new choice.

Table 5: Personal installation default values

Directory	Default
WebClient install	`%USERPROFILE%\Application Data\Progress Software\WebClient`
Program shortcuts	`C:\Documents and Settings\<user>\Start Menu\Programs\PSC WebClient <release-number>`
WebClient application install	`%APPDATA%\WebClientApps` **Note:** In some circumstances, the installer might choose a different default: • `%LOCALAPPDATA%\WebClientApps` (non-roaming data) • `%USERPROFILE%\Application Data\WebClientApps`

The lack of Administrator write privileges imposes some limitations on WebClient applications deployed as personal instances. There are common types of files that require Administrator privileges to install, and if you intend to support personal instances of your WebClient application, your application cannot use these types of files. For more information, see System Tasks and personal installations on page 87.

InstallShield setup player add-on for personal installations

The WebClient One-Click Install requires the InstallShield Setup Player to run properly. The InstallShield Setup Player is a control provided by Flexera Software that is installed by Internet Explorer as an Add-on. There are applications besides WebClient that might install the InstallShield Setup Player. Once the Add-on is installed, any application can use it for their Web-based installs.

If the Add-on is not present, a personal WebClient installation fails. You can check whether or not the Add-on is installed by choosing **Tools** > **Manage Add-ons** > **Enable or Disable Add-ons** in your web browser. Look for **InstallShield Setup Player** from **Flexera Software** in the dialog that appears. However, this installation requires Administrator write privileges.

If the Add-on is not present, you can add it before installing WebClient by having an Administrator install the control directly on the end users' machines. The OpenEdge installation includes the following installer for the Add-on:

OpenEdge-install-dir`\webinstall\webclient\web_image\SetupPlayer12.msi`

You can distribute this installer to your end-users. An Administrator can use the following command to silently install and register the Add-on:

```
source-directory\SetupPlayer12.msi /qn
```

The next time that their web browser runs on the machine, the Add-on is available and a personal WebClient installation can succeed.

If you need to uninstall the Add-on, have an Administrator run this command:

```
MsiExec.exe /X{80482183-337F-46D1-AF8B-F2686C9E7498}
```

Choosing an installation method

Before beginning 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.

You can use IntelliStream 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. Alternatively, you can use an external installer to install and update your application. If you want, you can also give your end user the option of updating an application or continuing to use an older but acceptable version of the installed application. For more information about the options available, see Design Your End User's Experience on page 31

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, you can 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 any procedure libraries that you include, and all the files must reside in the same directory structure that you intend to create on your end user's machine. Therefore, you must be able to access these files through the WebClient Application Assembler's **Add Files to Component** dialog. For more information about developing your application and compiling your code, see Develop Your Application on page 49

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 they should meet the following requirements:

- The 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 designated as follows:
  - **At Startup** — Installed before the application first runs
  - **As Needed** — Installed when a user runs your application and needs specific functionality
  - **Ask User at Startup** — Installed before the application runs if the user so requests in response to a prompt

- 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 the code required 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.
- 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 Defining applications in WebClient Application Assembler on page 80, the WebClient Application Assembler help, and Develop Your Application on page 49

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** on the **Options** tab in the WebClient Application Assembler. Files specified on the **Options** tab are automatically added to the **At Startup** component.

ABL install procedure

You can augment the IntelliStream install capabilities with an ABL install procedure. This procedure is specified on 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 an ABL install procedure, you can also provide an ABL uninstall procedure that runs at the end of the uninstall tasks. For more information about using ABL installers, see the Options tab on page 85.

Uninstall

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

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 and the type of installation.

Administrator instances use the following registry key:

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

Note: If you are installing an administrator instance of a 32-bit application on 64-bit Windows, maintain keys in the 32-bit subnode of the registry. If you use a 32-bit process to maintain the registry, it is transparent. If you use a 64-bit process to maintain the registry, the 32-bit subnode is:

```
HKEY_LOCAL_MACHINE\Software\Wow6432Node\PSC\WebClient
```

Personal instances use the following registry key:

```
HKEY_CURRENT_USER\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 application shortcuts and a default setting for your application installation directory. For information on programmatically checking the installation type, see the Checking application registry access on page 52.

The following table 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 6: 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 might not be 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 an application configuration file reference to it. 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:

```
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, the user's shortcut no longer works. Tell your users to start the application from the application Web page to automatically update the location of the application configuration file. After updating the location, the shortcut works again.

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:

```
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 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 the 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:

```
<Basekey>\Software\<VendorName>\<ApplicationName>\
```

Where *<Basekey>* is either the HKEY_LOCAL_MACHINE (HKLM) or the HKEY_CURRENT_USER (HKCU) registry hive depending on the installation type, and *<VendorName>* and *<ApplicationName>* match the values you enter in the **General** tab of the WebClient Application Assembler when you define your application.

The following table 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: 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.

Registry key value name	Data
`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 on page 85.
`Platform`	Platform of the application, if you are installing the application as a personal installation. Both 32-bit and 64-bit application installations are written to the `HKCU\Software\<VendorName>\<ApplicationName>` key or personal installations, and you need the platform value to identify the required platform for the application. The possible values are 32 and 64.

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. Your installation program must create the following registry key and registry key values defining the server for the application configuration file under this key:

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

Where <*Basekey*> is either the HKLM or the HKCU registry hive depending on the installation type, and <*VendorName*> and <*ApplicationName*> match the values you enter in the **General** tab of the WebClient Application Assembler when you define your application.

The following table 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 8: 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 Design for Security on page 41

Registry key value name	Data
`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 used by the WebClient Application Manager under the following key:

```
<Basekey>\Software\WebClient\Applications\
```

Where *<Basekey>* is either the HKLM or the HKCU registry hive depending on the installation type.

The following table 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 9: WebClient applications registry key value

Registry key value name	Data
<VendorName>/*<ApplicationName>*	" "

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

Using a combination of IntelliStream and an external installer

You can use a combination of IntelliStream and an external installer for your application.

Progress 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.

Your installation program must create the following registry key and registry key values for each component under this key:

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

Where <*Basekey*> is either the HKLM or the HKCU registry hive depending on the installation type, and <*VendorName*> and <*ApplicationName*> match the values you enter in the **General** tab of the WebClient Application Assembler when you define your application.

The following table 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 10: Component registry key entry

Registry key value name	Data
<*ComponentName*>	" "

ComponentName is the identifier you specified for the component in the WebClient Application Assembler **Component Definition** dialog.

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:

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

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.

-nonadmin

This option specifies that you are uninstalling a personal instance. You must include this option to uninstall a personal instance.

Defining applications in WebClient Application Assembler

With the WebClient Application Assembler, you create a WebClient project file (`.wcp` file), which is a definition of the latest version of an application.

After building the project file, you generate a set of deployment files for a version of your application. These generated deployment files are placed in a version-specific directory under an output directory that you specify.

You perform the following tasks with the WebClient Application Assembler:

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

---

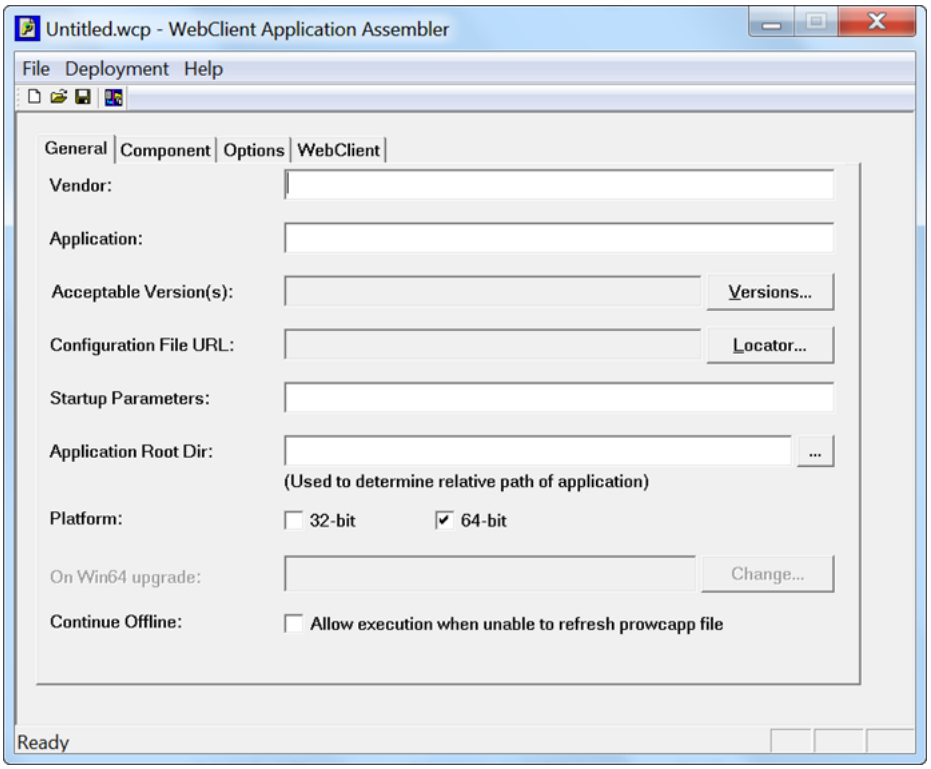
Note: You can use the WebClient Deployment Packager to modify details of the deployment configuration, such as the codebase location, rather than regenerating the deployment files. For more information, see the Customizing deployment configurations on page 89.

To run the WebClient Application Assembler:

1. Choose **Tools** > **PRO*Tools**. The **PRO*Tools** menu bar appears.
2. Choose **WebClient Application Assembler**

   The **WebClient Application Assembler** appears:

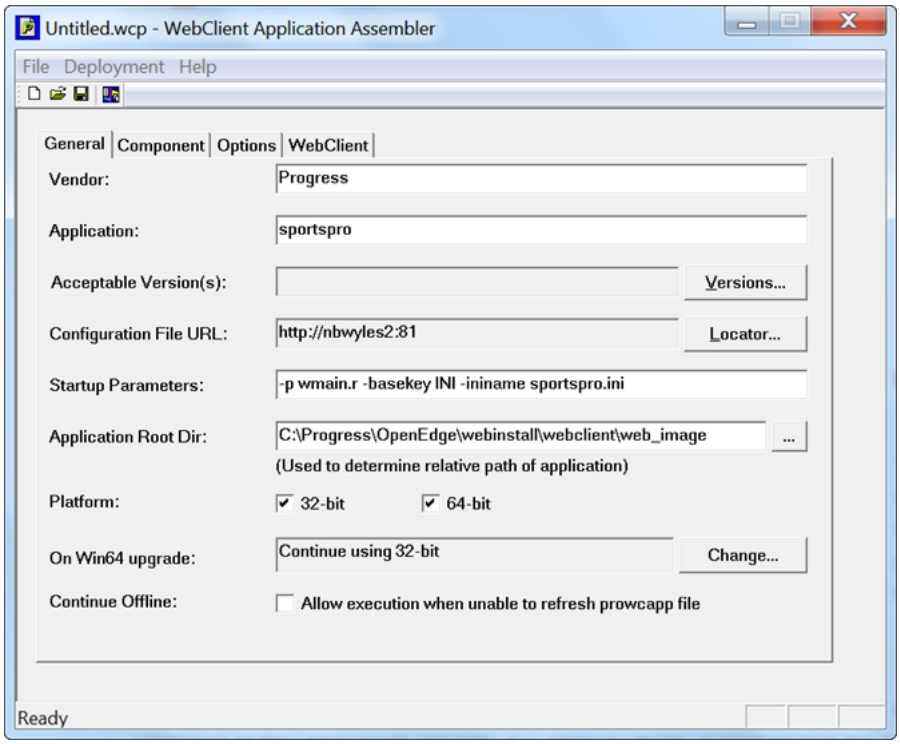


Note: Alternatively, you can run OpenEdge-Install-Directory\bin\prowcappmgr.exe from a Proenv command prompt.

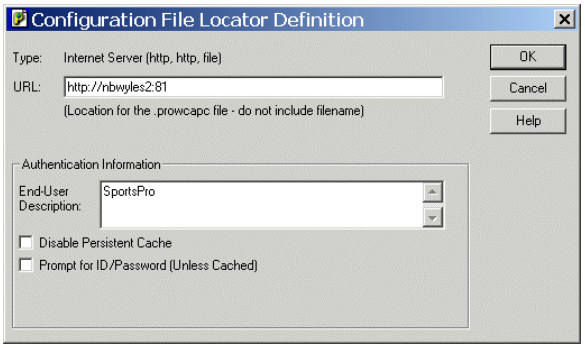
The remainder of this section describes the WebClient Application Assembler functionality.

General tab

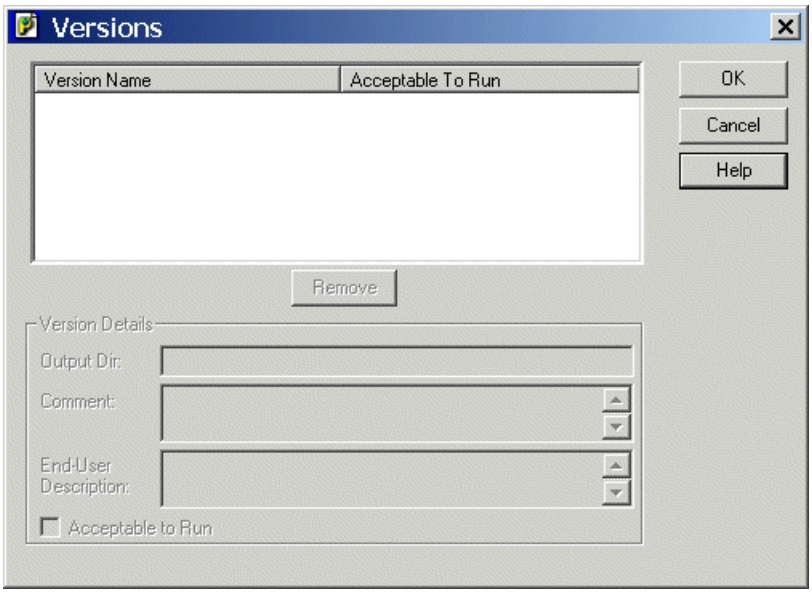
The **General** tab, shown here, provides basic information about the application, including startup information:



- Click **Locator** to open the **Configuration File Locator Definition** dialog. With this dialog, you specify the Internet server for the application configuration file and you define the server's authentication options:

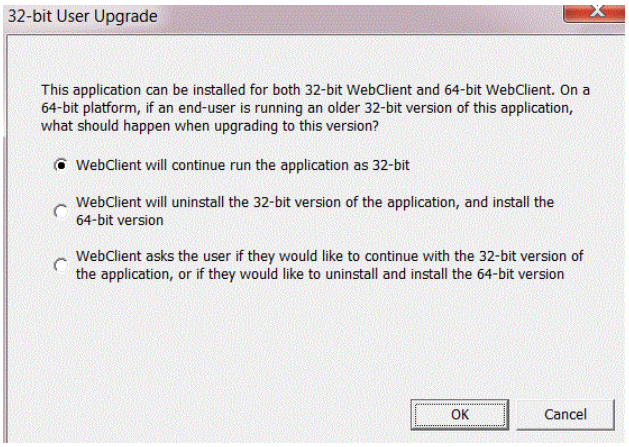


- Click **Versions** to open the **Versions** dialog. With this dialog, you specify which versions of your application are acceptable to run:



- Click **Change** to open the **32-bit User Upgrade** dialog box. With this dialog box, you define the action that WebClient should take when an end-user on a 64-bit Windows machine has an older 32-bit version of the application installed.

Note: Change is enabled only when both the 32-bit and 64-bit Platform check boxes are selected.

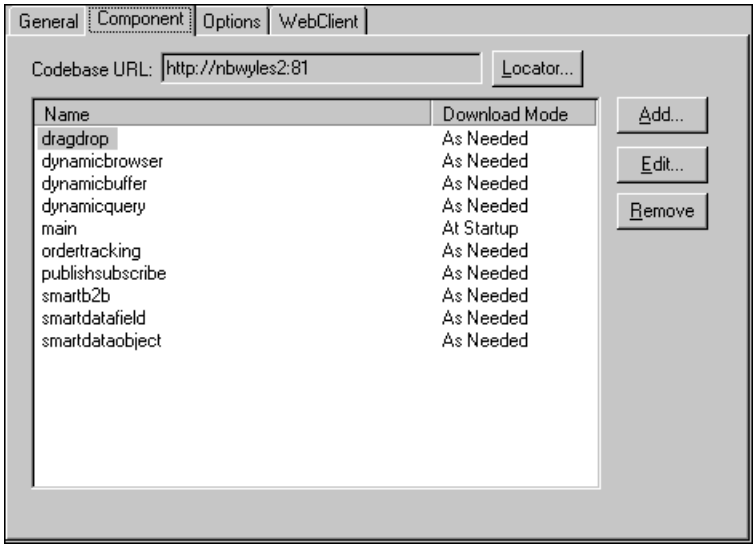


- Each time the end user starts the WebClient application, the WebClient Initializer checks the Web server that is hosting the application configuration file (*`application-name`*`.prowcapp`) to determine if an application update is available. If the Web server is not available, the WebClient application terminates. To allow the WebClient application to continue execution, even if it is not able to contact the hosting Web server, select **Allow execution to continue when unable to refresh prowcapp file**.

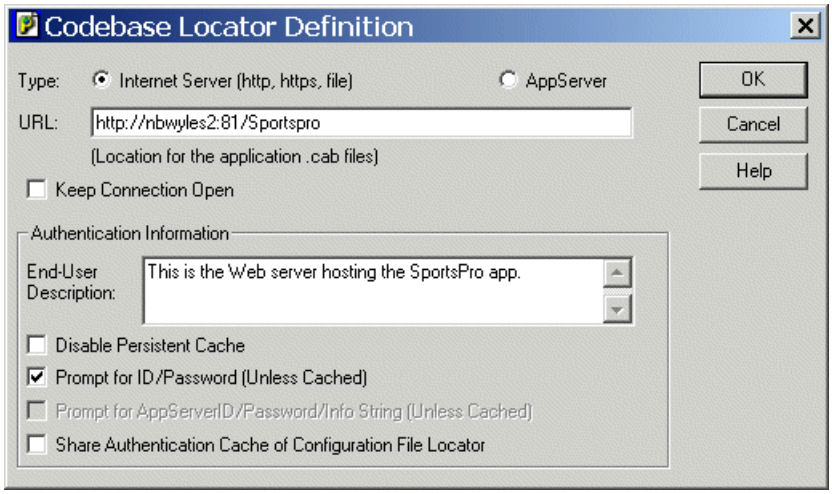
  This option can be useful in areas where network connectivity is poor or sporadic. The downside is that if there is a new version of the application available and the server hosting the application configuration file is down unexpectedly, then clients will attempt to continue running with the old version, even if the old version is now incompatible. Application vendors need to consider this and manage it within the application.

Component tab

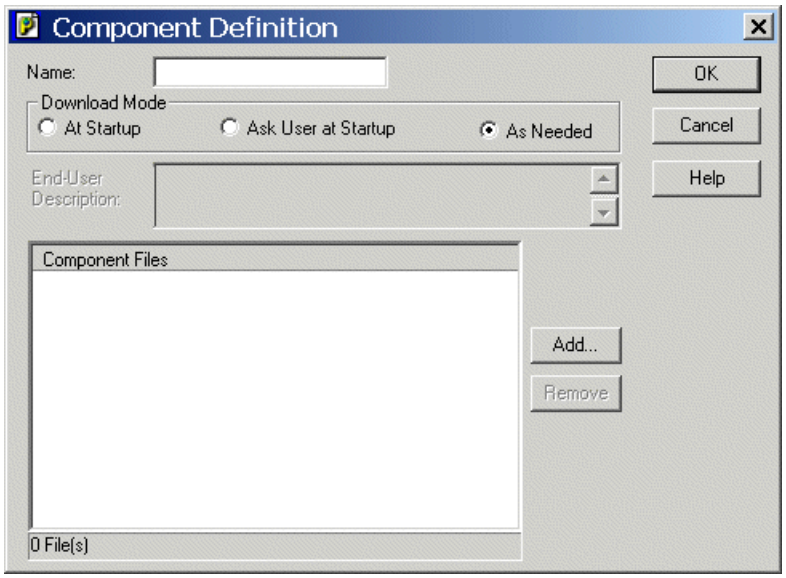
On the **Component** tab, you can add, remove, or change components for your application. Use this tab only if you use IntelliStream to install or apply updates for your application:



Click **Locator** to open the **Codebase Locator Definition** dialog. Here, you can specify the server of your application components and component updates (`.CAB`) files:

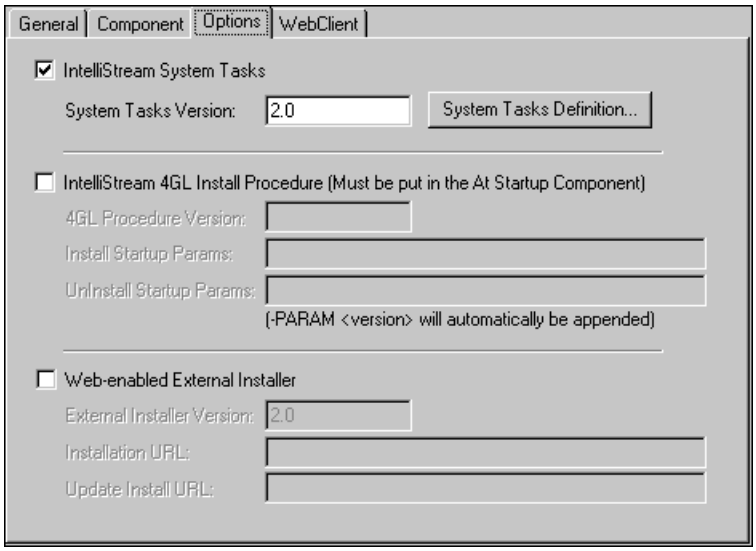


Click **Add** or **Edit** to open the **Component Definition** dialog. Use this dialog to define your component files and specify the component type:

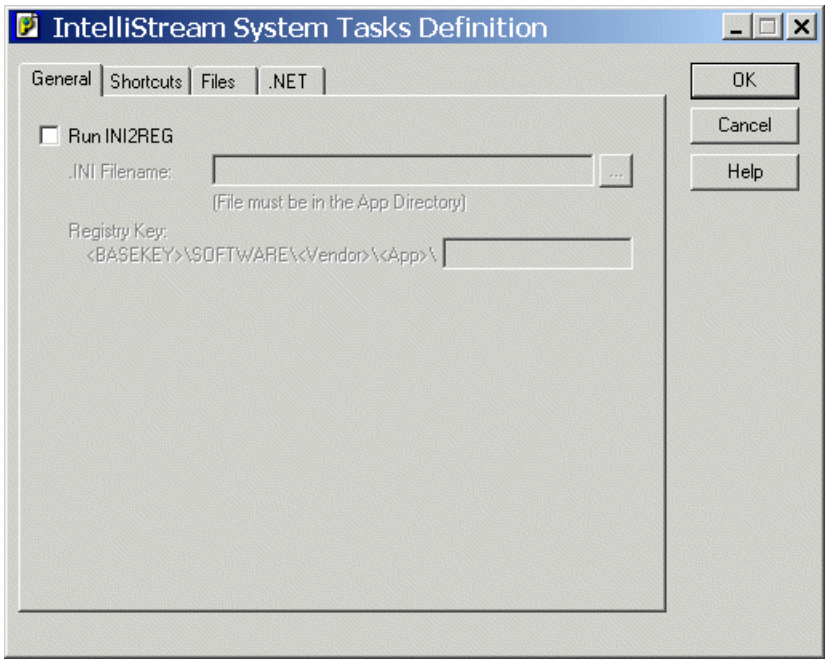


Options tab

The **Options** tab specifies options for the application installation. On this tab, you can indicate whether you are using an external installer, identify any system tasks you want IntelliStream to perform, and specify an ABL install procedure to handle additional installation steps.



Click **System Tasks Definition** to open the **IntelliStream System Tasks Definition** dialog:



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

- Running `ini2reg.exe`
- Adding shortcuts on the desktop
- Installing system or common files
- Registering application-specific files, such as the following ActiveX controls: `CSCombo`, `CSSpin`, and `PSTimer`
- Including a .NET Framework installer to support applications using OpenEdge GUI for .NET
- Including run-time versions of the OpenEdge Ultra Controls for .NET

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

ABL install procedure

You can use an ABL install procedure to perform installation tasks not available with IntelliStream system tasks. The ABL 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 an ABL install procedure, WebClient passes "*<CurrentVersion>, <NewVersion>*" as the value of the `-param` parameter to the procedure. As a result, when the ABL install procedure is run, the `-param` parameter is added to the startup parameters specified in the **Install Startup Params** field in the **Options** tab.

For example, if the last ABL install procedure was Version 4 of the installer, and the next installer is Version 5, the `-param` value is now `"4,5"`.

When the ABL 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 ABL 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 an ABL install procedure to perform installation tasks, you can also provide an uninstall procedure.

Web-enabled external installer

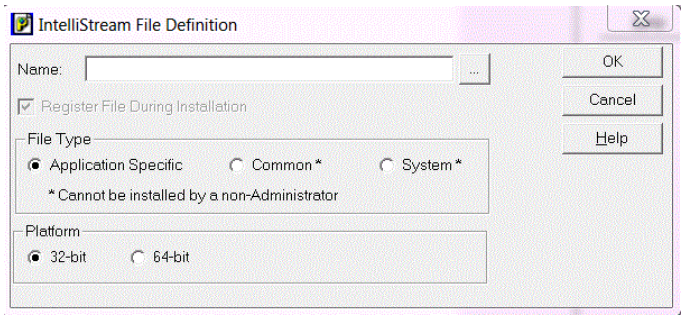
Use these options to provide information about your external installer. For more detailed information about this option, see the External installer on page 74.

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.

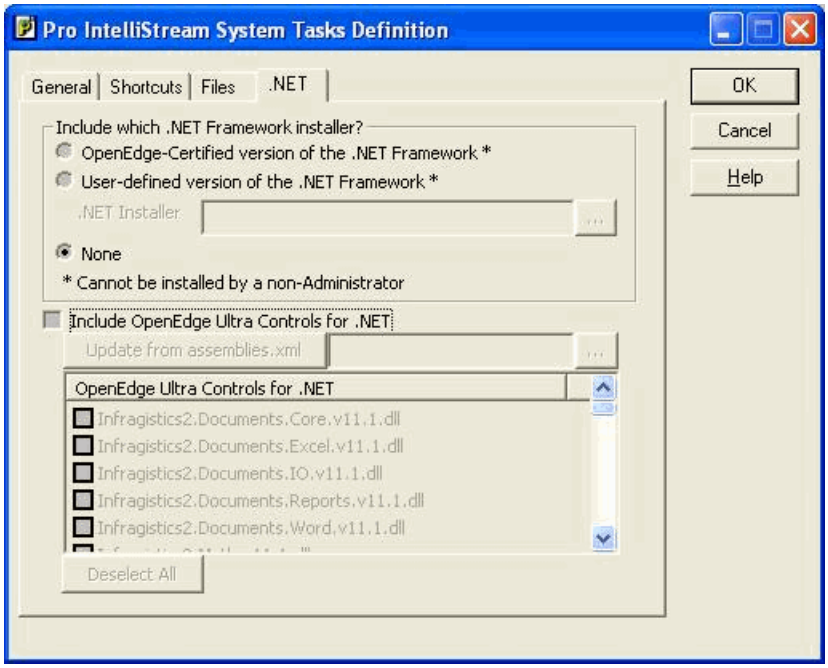
System Tasks and personal installations

Note that certain System Tasks require privileges usually restricted to members of the local or domain Administrators groups. If installing your application requires these tasks, users cannot install personal instances of the application. Such an application must use an Administrator install.

For example, certain files get installed to directories to which only an Administrator can write. The **IntelliStream File Definition** dialog flags these file types and platform, as shown:



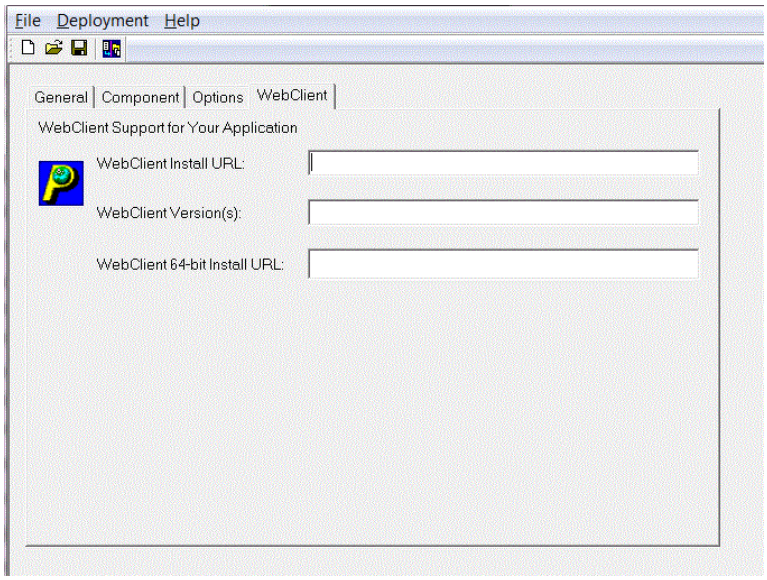
If your application uses the OpenEdge GUI for .NET, the user's machine requires an appropriate version of the .NET Framework. Only an Administrator can install the framework, so the **.NET** tab flags the options for including a .NET Framework installer, as shown:



For more information on personal installations and .NET, see the WebClient and OpenEdge GUI for .NET on page 109.

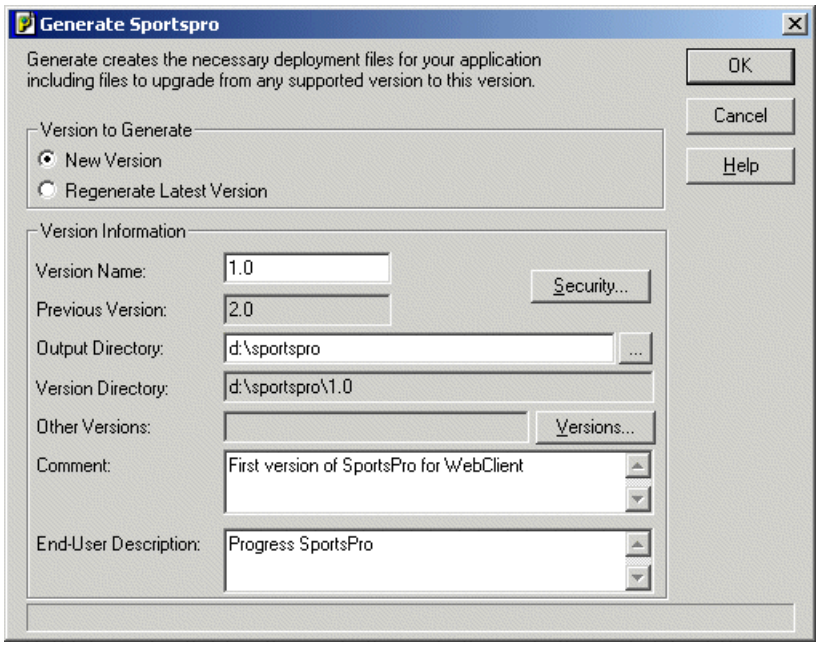
WebClient tab

The **WebClient** tab provides information about installing WebClient for your application:



Files generated by the WebClient Application Assembler

After you define the installation requirements for your application and you are ready to deploy, use the WebClient Application Assembler **Generate** dialog to create the deployment files for a version of your application. To access the **Generate** dialog, choose **Deployment Generate** or click **Generate** on the toolbar. The **Generate** dialog appears:



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

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 can 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** — 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 click **Security** and select a digital signature option in the **Security** window, the `.prowcapc` file and all the component files are signed with a digital certificate. (See the Defining an application as signed on page 43 for more information.)

- **WebClient version information file (`.wcv`)** — Contains version information for all the files within the application, including `.r` files within procedure libraries. A `.wcv` file is generated for each version and is saved in the version directory.

  Note: The `.wcv` file is not deployed.

- **Application component files (`.CAB`)** — 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.
- **Application components update files (`.CAB`)** — Contain the files necessary to update the application components. The WebClient Application Assembler generates one `.CAB` file for each component that needs updating, and saves it 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.

Customizing deployment configurations

Users of your application might have specific requirements for how it is deployed. Often you cannot be aware of these requirements at the time that you generate the `.prowcapc` file. Even when custom requirements are known, maintaining separate development streams to support deployment to multiple users can be burdensome and error-prone. The WebClient Deployment Packager enables you to address this problem without changing the `.wcp` project file and regenerating the entire application. With the Deployment Packager, you can:

- Extract deployment information from an existing `.prowcapc` file.

- Modify the deployment information and save the changes to a file referred to as a *custom deployment configuration*. This file has a `.wcd` extension and the same name as the `.prowcapc` file (for example, a custom deployment configuration extracted from `sportspro.prowcapc` is stored as `sportspro.wcd`).
- Generate a new `.prowcapc` file incorporating the changes made to the `.wcd` file.

You might use this capability in the following cases:

- Supporting customers with different code pages or languages
- Providing access to a test environment in addition to your production environment
- Customizing deployments that each customer hosts internally

After the initial generation of the `.prowcapc` file with the WebClient Application Assembler, the developer or the customer can use the Deployment Packager to modify the configuration details as required. OpenEdge provides a standalone Deployment Packager executable that you can make available to customers, enabling them to customize their own configurations.

Distribution of the Deployment Packager

The Deployment Packager is an integral part of the PRO*Tools suite and is installed with Progress Developer Studio for OpenEdge.

End users can also create their own custom deployment configurations with the Deployment Packager by working with the `.prowcapc` files that they receive from application developers. Since end users typically do not have access to Progress Developer Studio for OpenEdge, a standalone version of the tool and the required libraries is provided in a file named `prowcdp.zip`. You can distribute this version to your customers as appropriate.

Distributing and installing the standalone version

The `prowcdp.zip` file is installed in the OpenEdge `bin` directory. You can copy and distribute this single file to customers and other users. On the target system, extract the contents of the archive into a single directory. Then, in that directory, execute the following commands to register components:

```
regsvr32 psccab.dll
regsvr32 psccert.dll
regsvr32 pscsecreg.dll
```

You should see a success message after each command. The Deployment Packager is now ready for use.

Changes allowed through the Deployment Packager

The Deployment Packager can change the following elements of the application definition:

- Application name
- WebClient install URL
- Configuration file URL and associated security information
- Codebase URL and associated security information
- Digital signature information for the `.prowcapc` file
- URLs for Web-enabled external install and upgrade programs

You cannot use the Deployment Packager to modify the content of the application itself.

Defining a custom deployment configuration

Before running the WebClient Deployment Packager, determine the correct values for your custom deployment configuration, and make sure that the source `.prowcapc` file is accessible.

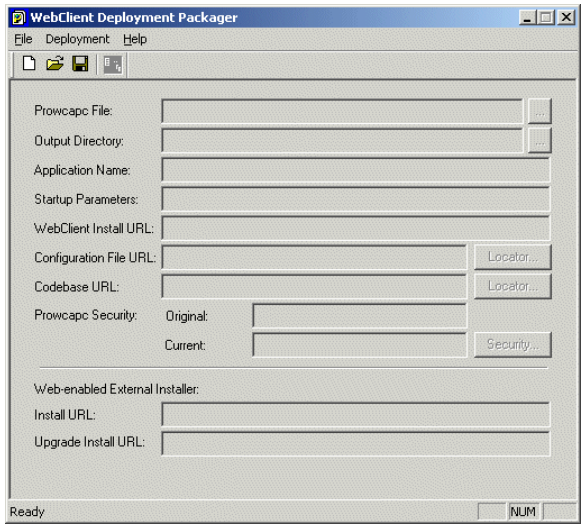
To run the WebClient Deployment Packager:

1. Choose **WebClient Deployment Packager** from the PRO*Tools palette; or execute the following command from either <*OpenEdge-install-dir*>\bin, or the directory where you unzipped `prowcdp.zip`:

   ```
   prowcappmgr.exe -deploy
   ```

   The **WebClient Deployment Packager** window appears:

2. Create a new custom deployment configuration, or open an existing one.
3. You can save the deployment configuration to a `.wcd` file at any time. If the configuration has not been saved previously, you see a standard **Save As** dialog that enables you to name the file with a `.wcd` extension and to specify its location.
4. Most of the Deployment Packager fields get their initial values from the source `.prowcapc` file. Enter new values as appropriate for the custom deployment configuration. See the Defining applications in WebClient Application Assembler on page 80 and the online help for more information.

Caution:

Your .prowcapc file and your application .CAB files must be either **both signed** or **both unsigned**. The application will **not** run if there is a mismatch. Therefore, you must be sure that the security setting for your new .prowcapc file matches that of the .CAB files. The value that you specify in the **Current** security field applies only to the new .prowcapc file; it does not affect the .CAB files. The value (**Signed** or **Unsigned**) that the Deployment Packager shows for the **Original** .prowcapc file typically reflects the status of the .CAB files, as well as that of the source .prowcapc file, so you should usually match that setting. If the security settings do not match when you validate or generate the new configuration, a warning is displayed.

5. After defining the custom deployment configuration, save your changes and generate the new .prowcapc file, as explained in the Generating the .prowcapc file on page 93. Optionally, you can validate the configuration in a separate operation before generating the new file. (Before executing a Generate operation, the Deployment Packager automatically performs the same validation.)

Custom deployment security considerations

Customers who host WebClient applications developed by an outside application provider must coordinate with the provider to make sure security is properly configured. These customers also must instruct their end users to recognize and trust their digital signatures.

If the application provider makes the prowcdp.zip file available, the customer can run the Deployment Packager and manage the security settings and other settings independently. Alternatively, the customer may specify the requirements and ask the application provider to define the deployment configuration accordingly.

Working as necessary with the application provider, the customer must make sure that:

- Both the .prowcapc file and the .CAB files are signed if digital signatures are used. (You do not need to use the same certificate for both; either the customer or the provider can sign the files.) If one is signed and the other is unsigned, the application will not run. You can only specify the digital signature (or the absence of it) for the .prowcapc file with Deployment Packager. Typically, the application provider must make any required changes to the .CAB files.
- The authentication settings, as well as the URLs, are correctly specified for the Configuration File Locator and the Codebase Locator. Either the customer or the application provider can adjust these settings through the Deployment Packager.
- End users are properly informed how to respond to security prompts. For digitally signed files, you should instruct end users to trust content signed by either the customer or the application provider, as applicable.

Validating the custom deployment configuration

To check for problems before generating a new .prowcapc file, open the .wcd file and choose **Deploy** > **Validate**. The Deployment Packager analyzes the settings you specified and reports any of the following that it detects:

- **Error conditions** — Problems that you must correct before generating the new .prowcapc file, such as invalid or missing values in the **Prowcapc File**, **Application Name**, and **Output Directory** fields.
- **Warning conditions** — Settings that might cause problems when the application runs, but that do not prevent generation of the new .prowcapc file. For example, it displays a warning if the new application name differs from the name of the source .prowcapc file.

If there are no errors, the Deployment Packager displays the "Validation succeeded" prompt that lists the changes you made—that is, differences between the original configuration and the new one—as well as any warnings. Review the information before dismissing this prompt.

If errors are found, the Deployment Packager lists them in the "Validation failed" prompt. You must correct the errors before the Deployment Packager can generate the new file.

Generating the .prowcapc file

When you are ready to store your custom deployment configuration in a new `.prowcapc` file, you can do so in the following modes:

- **Interactive mode** — Starts the generation process from the Deployment Packager main window.
- **Batch mode** — If Progress Developer Studio for OpenEdge is installed, you can execute a command-line script that performs the operation without user interaction.

In both cases, the Deployment Packager writes a log of the operation in the output directory where it creates the `.prowcapc` file. The log file has the same name as the `.prowcapc` file, but with a `.log` extension.

Interactive mode

Open the `.wcd` file and choose **Deploy** > **Generate**, or click



on the toolbar. The Deployment Packager validates the configuration, as described in the preceding section.

If there are no errors, the Deployment Packager displays a prompt that lists the changes you have made—that is, differences between the original configuration and the new one—as well as any warnings. At this point, you have the option of continuing or canceling the operation. After reviewing the information, click **Yes** to generate the new `.prowcapc` file, or click **No** to abort the operation.

When you click **Yes**, the Deployment Packager generates the new `.prowcapc` file in the specified output directory. The filename matches the value of the **Application Name** field.

Upon completion, a message identifies the new `.prowcapc` file (or indicates the errors that prevented successful generation of the file). It also identifies the log file for the operation.

Batch mode

If Progress Developer Studio for OpenEdge is installed, you can use the `bprowcappdep.bat` script in *<OpenEdge-install-dir>*`\bin` to generate the `.prowcapc` file.

Note: Progress Developer Studio for OpenEdge is not available to users of the standalone Deployment Packager tool extracted from the `prowcdp.zip` file.

You execute the script passing it the path to the deployment configuration (`.wcd`) file, for example:

```
bprowcappdep c:\WebClient\deployment\MyApp.wcd
```

Unless the validation process encounters errors (in which case the operation terminates), generation of the `.prowcapc` file proceeds with no option to cancel. Messages that precede the actual start of the generation process, including error notifications and warnings, are written to the console window; they do not appear in the log file. Therefore, when using the `bprowcappdep.bat` script, you should redirect the screen output to a file or take other steps to make sure you do not overlook important messages.

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 some options when deciding on the type of server you want to use to host your application. You can use an Internet-based Web server or file server, or you can use an application server. For more information about design considerations to help you make these decisions, see Host your WebClient Deployment on page 37

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

Hosting the configuration file

Place the application configuration file in the directory specified by the URL in the WebClient Application Assembler's **Configuration File Locator Definition** dialog.

Note: The application configuration file cannot be hosted on an application server.

The first time the end user downloads the configuration file, the `bootstrap.htm` file specifies its location. For more information about this `bootstrap.htm` file, see the Bootstrapping a WebClient application on page 101. When downloaded, the configuration file is cached on the user's machine.

Although the application configuration file is generated in the version directory, you should place the file in a version-independent directory for deployment. 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 specified in the **Configuration File Locator Definition** dialog. 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 rerun the bootstrap process from the initial installation Web page to automatically update the shortcut.

Hosting your application component files

Place the component (`.CAB`) and component update files for your application in the directory that you specified in the WebClient Application Assembler's **Configuration File Locator Definition** dialog. For more information about specifying a codebase locator, see Host your WebClient Deployment on page 37

You should copy the version directory and its contents, including all its subdirectories, as specified in the dialog. The application version directory on the server machine must match the the version directory used in the WebClient Application Assembler **Generate** dialog.

Where you can place your application files depends on which type of server you use:

- **Application server** — The application version directory must be relative to the Application server's working directory.
- **Web server** — The application version directory must be relative to the root directory of your Web server.

- **File server** — The application version directory should be a subdirectory of the directory specified in the file URL.

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.
- 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.

For more information on specifying these values, see the Options tab on page 85.

Hosting on a Web server

Deploying your application component files or external installer on a Web server presents extra considerations.

Case sensitivity on UNIX Web servers

Web servers that use the UNIX operating system, such as Apache, are case-sensitive. Mismatched cases between the installation filenames on the Web server and those in your application configuration file will cause a run-time error.

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. 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 Microsoft IIS, select only the **Basic (Clear Text)** authentication option in the **WWW Services Properties** for the Web server. If you select the **Windows Challenge/Response** authentication option, end users who are not running Internet Explorer receive an authentication dialog that they cannot satisfy, preventing them from starting an installation. This prevents end users from using any other Web browser with your application.

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`

Note: If an Internet Service Provider (ISP) hosts your Web installation, you must give them the information to configure the MIME type properly on their Web server.

The following procedures cover configuring the MIME type on common Web servers.

1. To register the MIME types for all directories on an Apache Web server, do one of the following:
   a) Add the following line to the `srm.conf` file:

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

   b) Add the following line to the `mime.types` file:

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

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

   Note:

   For a Sun Java System Web Server, you must change the association of `.exe files` in addition to adding the definition for `application/progress-wcappcab`.

   ---

2. To configure MIME types on **Sun Java System Web Server**:
   a) Open the Administration Server.
   b) Select a server and choose the **Manage** button.
   c) Choose **Preferences** > **Mime Types** .
   d) Add the new content-type definition `application/progress-wcappcab`, and associate it with the file suffix `prowcapc`.
   e) Edit the **magnus-internal/cgi** MIME type definition, and remove **exe** from the list of associated extensions.
   f) Edit the **application/octet-stream** MIME type definition, and add **exe** to the list of associated extensions.
   g) Save and apply your changes so the Web server recognizes the changes to the MIME type definitions.

   Note:

   The Microsoft 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 found in <*`OpenEdge-install-dir`*>\webinstall\webclient\web_image.

   ---

3. To merge the MIME type entries from iis_wc.reg into the registry and configure the Web server:

a) Copy the `iis_wc.reg` file to your Web server.

b) To register the MIME type by doing one of the following on the Web server, either select `iis_wc.reg` in the Windows Explorer, right-click, and select `Merge`. This imports the contents of the file into the registry or double-click the file in the Windows Explorer. This imports the contents of the file into the registry.

c) Restart your Web server to register these changes.

Installing WebClient

You must make an installer available to install WebClient on the user's machine. OpenEdge provides a default InstallShield image that you can customize to install WebClient.

You can also point the user to a page on the Progress Web site where the user can run a WebClient installation.

Your OpenEdge installation includes the following WebClient install images:

- A stand-alone InstallShield that you can use to create an installation flash drive. Invoke the setup-exe directly to install WebClient.
- A One-Click Install 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 on page 97.

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

- An OpenEdge-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 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 flash drive installation, Progress strongly recommends you customize the One-Click Install. For more information, about customizing the WebClient installation, see Customizing the WebClient installation on page 98.

Note: Whatever Web page you provide for accessing 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.

Accessing the WebClient installation files

The prebuilt WebClient InstallShield image is a ready-to-deploy WebClient installer that you can find in *<OpenEdge-install-dir>*`\webinstall\webclient\web_image`.

To use this installer, do either of the following:

- To create a flash drive for installing WebClient, copy the `web_image` directory contents to the flash drive that you want to distribute.

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

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 installer's original operation. You can only add functionality to the operations that it performs by default, and change the appearance of the page. Any other changes might produce unpredictable results in the WebClient installation.

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

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

To implement one of the supported customizations, you must modify the **bolded** portions of the following `webclient.htm` fragment:

```
function scriptPlayer() {
    // Catch player initialization error
    if ((!player) || (typeof(player.LastError) != "number")) {
        iHelp("player","init", ERR_CONTROL_LOAD);
        return;
    }
    else {
        if (player.LastError != 0) {
            iHelp("player","init", player.LastError);
            return;
        }
    }

    // URL from address bar
    var strSetupURL = getSetupURL();

    // Open the installation
    ether = player.Open(strSetupURL);

    // Catch player open error
    if (!ether) {
        iHelp("player", "open", player.LastError);
        return;
    }

 // 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:
    //
    //       ProwcappURL = "http://server1/webclient/app_installer/yourapp.prowcapc";
    //
    //   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:
    //
    //       ApplicationURL =  "http://server1/webclient/app_installer";
    //      ProwcappURL  =  "http://server1/webclient/app_installer/yourapp.prowcapc";

 //By Default set the URL as below:
 //  ProwcappURL = "none";
 //  ApplicationURL ="none";
    //Set command line

 ApplicationURL ="none";
 ProwcappURL = "none";


 // DO NOT EDIT THE NEXT LINE!!!
    SetupPlayerID = "InstallShield.OCISetupPlayerOCX17";
```

```
 var commandline =  "/hide_usd" + " -psc_prowcappURL=" + ProwcappURL +
"-psc_applicationURL=" + ApplicationURL + "-psc_setupPlayerID=" + SetupPlayerID;
 if (IS_NAV == true)
        commandline = commandline + "-psc_UsingNetscape"

 ether.SetProperty( "is::CmdLine", commandline );

    // Ready to install
    writeText("txtMsg", "Click <a href='javascript:startInstall()'>Install</a> to
<b>start</b> this installation now.  You will be prompted when the InstallShield Wizard
 starts.");
    document.images["imgOCI"].src = "_Graphics/Install.gif";
}

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

function scriptPlayer() {
ProwcappURL ="http://server1/webclient/app_installer/yourapp.prowcapc";
```

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

```
function scriptPlayer() {
ProwcappURL ="http://server1/webclient/app_installer/yourapp.prowcapc";
```

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 scriptPlayer() {
ApplicationURL ="http://server1/webclient/app_installer";
ProwcappURL ="http://server1/webclient/app_installer/yourapp.prowcapc";
```

This customization first executes your application's One-Click Install from the specified `BakeWare_installer` directory on your Web server. 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.

Bootstrapping a WebClient application

Whether you use IntelliStream or an external installer to install your application, the WebClient Initializer always starts your WebClient application.

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

The Initializer uses information in an application configuration file to control how it starts a WebClient application. The information that you provide in the 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.

How the bootstrapping process works

The prescribed way to bootstrap your WebClient application is to use a customized version of the `bootstrap.htm` file that OpenEdge provides. The `bootstrap.htm` file initiates the installation process.

The `bootstrap.htm` file has the following paths:

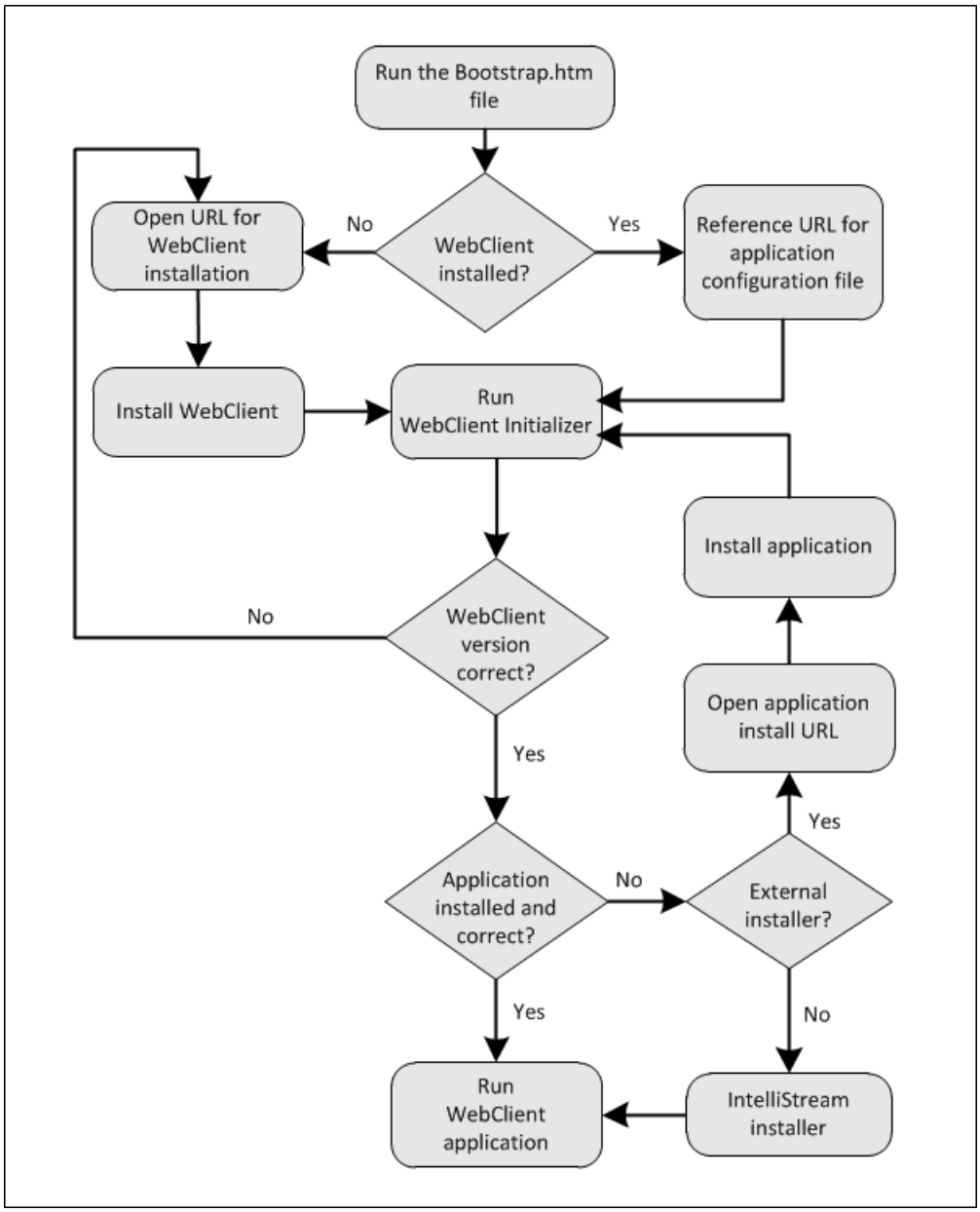
- If WebClient is not installed, your end user is taken to a Web page to install WebClient.
- If WebClient is installed, WebClient Initializer runs and performs the following tasks:
  - Checks that an acceptable version of WebClient is installed on the user's system and gives the user an opportunity 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.

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.

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

Figure 1 illustrates the bootstrapping process and shows the bootstrap process flow depending on Initializer's verification at the different points in the process. This process is transparent to the end user, varying slightly depending on the installation method and any customizations that you made to install and start your application.

Figure 5: Figure 1. Bootstrapping process



Bootstrapping and personal instances

To prevent the WebClient installer from starting a personal install without an appropriate version of the WebClient, the `bootstrap.htm` file contains the variable `supportPersonalInstall`. This variable defaults to FALSE. If your WebClient application supports personal installs, you must set this variable to TRUE. This forces an early check that there is an appropriate version of WebClient.

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. You must specify the WebClient installation URL and customize the bootstrap HTML file accordingly.

Bootstrapping with a 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. You must specify the configuration file URL and customize the `bootstrap.htm` file accordingly.

Note: If you set `supportPersonalInstall` to TRUE and the user's machine does not have an appropriate version of WebClient installed, the bootstrap process forces an install before running WebClient Initializer.

Customizing the bootstrap.htm file

If you use the `bootstrap.htm` file, you must customize it.

To customize the `bootstrap.htm` file, you substitute your own URLs for the two default URL settings:

- **`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 Software Corporation Web site.

The option for supporting personal instances and the default URL settings are **bolded** in this sample `bootstrap.htm` file.

Sample bootstrap.htm file

```
<html>
<script language="JavaScript">
  installed = false;
  // NOTE: Set supportPersonalInstall to true to require a version of     WebClient
that supports personal instances,
  // even if there is already a version of WebClient installed
  supportPersonalInstall=false;  netscape = (navigator.appName.indexOf("Netscape") !=
-1);

// See if WebClient is installed
  if (netscape)
  {
    var mimeTypes;
    if (supportPersonalInstall)
      mimeTypes = navigator.mimeTypes["application/progress-wcappna"];
    else
      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
    if (supportPersonalInstall) then
      installed = supportsNonAdmin(ctrl.Version)
    else
      installed = true
    end if
  end if
```

```
' Check if the version of WebClient supports personal installs
Function supportsNonAdmin(verstring)
  tmp = false
  cdot1 = InStr(verstring,".")
  if cdot1 > 0 then
    cmajor = Cint(Mid(verstring,1,cdot1 - 1))
    cdot2 = InStr(cdot1 + 1,verstring,".")
    if cdot2 > 0 and cmajor >= 10 then
      tmp = true
    else
      cminor = Cint(Mid(verstring,cdot1 + 1,1))
      cmaint = Mid(verstring,cdot1 + 2,1)
      if cmajor > 10 then
        tmp = true
      elseif cmajor = 10 then
        if cminor < 2 then
          tmp = false
        elseif StrComp(cmaint,"B") < 0 then
          tmp = false
        else
          tmp = true
        end if
      end if
    end if

  end if  'cdot1 > 0
  supportsNonAdmin = tmp
End Function
</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://MyWebServer/webclient/myapplication.prowcapc";  }
  else
  {
    // Start install process for Web Client
    // NOTE: MODIFY THIS URL TO POINT TO THE WebClient Install page
    window.location = "http://MyWebServer/webclient/webclient.htm";  }
</script>
<body>

<!--// 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>
```

Checking for an acceptable version of WebClient

The WebClient Initializer first verifies that an acceptable version of WebClient is installed on the 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 to run with an existing version. If there is no older but acceptable version, the Initializer gives the user the 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 use an installed acceptable version, 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 the user's Web browser to the URL of the application install page and exits.

WebClient application updates

To make changes to your application, generate a new version of the application with the WebClient Application Assembler.

Generating a new version of the application creates a new configuration file and any necessary component files and component update files, placing 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.

Note: A major benefit of IntelliStream is that only the individual modified files are downloaded during an update. The WebClient Application Assembler determines what files are modified by comparing the existing and new r-code signature values.

If you use an external installer to update the application, you can re-use the Web page and logic for initially installing your 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 then exits. At this point, the user can continue with the update over the Web.

Updating procedure libraries

If your WebClient application uses procedure libraries, they might require special handling to update. While IntelliStream can perform incremental updates to standard procedure libraries, the technology does not support incremental updates to memory-mapped procedure libraries.

Progress recommends that you do not use memory-mapped procedure libraries in WebClient applications. If your WebClient application must use memory-mapped procedure libraries, you can consider the following alternatives:

- Do not attempt incremental updates of the memory-mapped procedure library. Instead, whenever the contents of the procedure library change, create a new version of your WebClient application and do not support previous versions with the older procedure library.
- Do not use IntelliStream to update your WebClient application. Use another solution for upgrades.
- Include the standard procedure library from which you create the memory-mapped procedure library in your deployment. Update the contents of the standard procedure library with IntelliStream. Then, use a procedure that creates a new memory-mapped procedure library on the client with `PROLIB`.

Suppressing the WebClient Application Update dialog

You can suppress the WebClient Application Update dialog to allow an update to run without end user intervention.

If the installed version of the WebClient application on the user's system is no longer acceptable to run, the WebClient Initializer provides the user the opportunity to update the application by displaying a WebClient Application Update dialog. You can suppress the WebClient Application Update dialog to allow the update to run without user intervention.

To suppress the WebClient Application Update dialog on a user's system, you must create a string value `AppAutoUpdate` in the Windows registry of the user's system. To disable suppressing the WebClient Application Update dialog, remove the `AppAutoUpdate` entry from the Windows registry of the user's system.

You can create the *`AppAutoUpdate`* registry entry by either:

- Using Windows registry
- Running the `ini2reg` utility

Caution: Using **Registry Editor** incorrectly can cause serious, system-wide problems. You might have to re-install the Windows operating system to correct them. Use this tool with caution.

Before you edit the registry, you must backup your current registry information.

To back up your Windows registry information:

1. From the Windows menu, click **Start** > **Run**.
2. In the **Open** field, type `regedit` and click **OK**.
3. In the **File** menu of the **Registry Editor**, click **Export**.

   The **Export Registry File** window appears.

4. In the **Save in** drop-down, browse to the location where you want to save the registry backup file.
5. In the **File name** field, type a name for the backup file.
6. Click **Save**.

   Your current Windows registry information is saved as a backup file at the location you specified.

Creating the string value `AppAutoUpdate` entry using Windows registry

To create the string value *`AppAutoUpdate`* entry using Windows registry:

1. Open **Registry Editor** window.
2. Do one of the following:
   - To install as an Administrator, browse to `HKEY_LOCAL_MACHINE\SOFTWARE\<`*`VendorName`*`>\<`*`ApplicationName`*`>`.
   - To install as a user, browse to `HKEY_CURRENT_USER\SOFTWARE\<`*`VendorName`*`>\<`*`ApplicationName`*`>`.

   Note:

   This registry entry is where the application information gets updated when the application is installed.

3. Right-click the <*`AppName`*> and select **New** > **String Value** from the context menu.
4. Enter the string name as *`AppAutoUpdate`* and click **ENTER**.

   The *`AppAutoUpdate`* registry entry is created under <*`AppName`*>.
5. Close the **Registry Editor** window.

Creating the string value `AppAutoUpdate` entry using the `ini2reg` utility

To create the string value *`AppAutoUpdate`* entry using the **`ini2reg`** utility:

1. Create an `.ini` file as follows:
   a) Open **Notepad**.
   b) Enter the following in separate lines:

   ```
   [ApplicationName]
   AppAutoUpdate=1
   ```

   c) From the main menu, click **File** > **Save As**.
   d) In the **File name** field, type a file name and save the file with `.ini` extension. For example, save the file as `<filename>.ini`.
   e) Click **Save**.
2. Run the `ini2reg` utility to update the registry:

a) Open the Windows command prompt.

b) At the command prompt, type `cd\` and press **ENTER**.

c) At `C:\>` prompt, type the following in a single string and press **ENTER**:`ini2reg -ao -b HKEY_LOCAL_MACHINE -s "SOFTWARE\<VendorName>" -i <filename.ini>`

Note:

To install as a user, use `HKEY_CURRENT_USER` node.

The string value *`AppAutoUpdate`* entry is added to the Windows registry.

WebClient and OpenEdge GUI for .NET

When deploying applications that use the OpenEdge GUI for .NET with WebClient, there are additional issues to consider. These include handling the following:

- .NET Framework on page 109
- .NET assemblies on page 111
- Other files on page 111

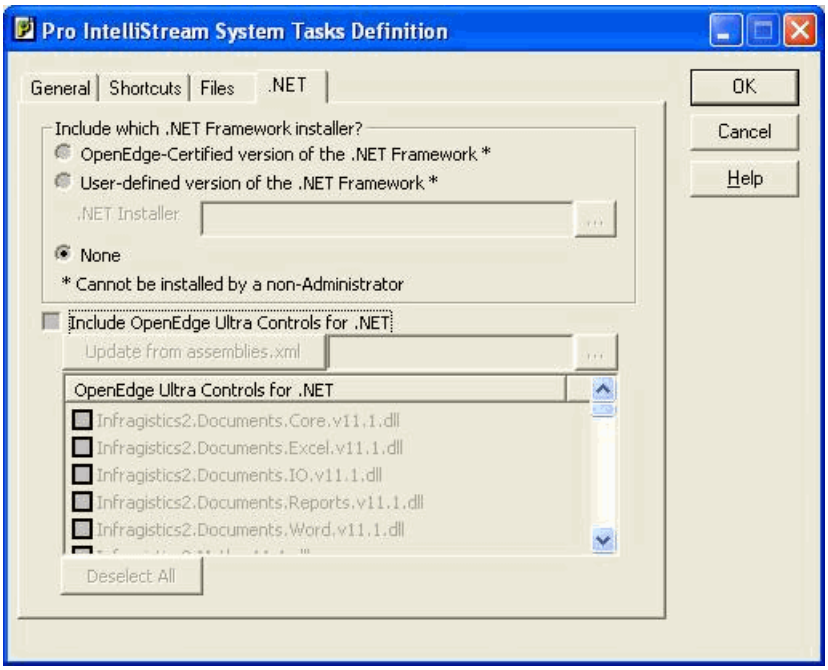
.NET Framework

Applications using the OpenEdge GUI for .NET require an appropriate version of the .NET Framework to run. An end user cannot start an OpenEdge GUI application without having an appropriate version of .NET installed. When you use WebClient to deploy one of these applications, your deployment must accommodate this requirement.

Note: Because users without Administrator privileges cannot install the .NET framework, you should consider this requirement carefully if your WebClient application supports personal installations.

You can address this requirement in the following ways:

- Include the OpenEdge-certified version of the .NET Framework in your WebClient deployment through the **IntelliStream System Tasks Definition** dialog:



This option packages a .NET Framework installer in the initial WebClient install. The WebClient Initializer checks the end-user's machine for either that version of the .NET Framework or a later version. If one is found, it aborts the install. This result enables both Administrator and personal installations to continue.

Otherwise, it launches the .NET Framework installer. An end-user with Administrator privileges must accept the Microsoft EULA and complete the .NET Framework installation to continue installing your application. However, the framework installer fails for a non-Administrator, and the WebClient Initializer displays an error and aborts the personal installation.

- Include a user-defined version of the .NET Framework in your WebClient deployment through the **IntelliStream System Tasks Definition** dialog. This option packages a .NET Framework installer that you supply in the initial WebClient install. Unlike with the OpenEdge-certified version, the WebClient Initializer does not check the end-user's machine for an existing .NET Framework. The WebClient Initializer always launches the user-defined .NET Framework installer. An end-user with Administrator privileges must accept the Microsoft EULA and complete the .NET Framework installation. However, the framework installer fails for a non-Administrator, and the WebClient Initializer displays an error and aborts the personal installation.

  For more information on appropriate versions of the .NET Framework, see the chapter on maintaining user environments in *OpenEdge Deployment: Managing ABL Applications*.

Caution:

Some .NET Framework installers might uninstall existing versions of the framework. You should verify what a particular installer does before including it in your WebClient deployment.

- Do not include a .NET Framework installer in your WebClient deployment. This option leaves it to the end users to ensure that an appropriate version of the .NET Framework is available on their machines before they install your application.

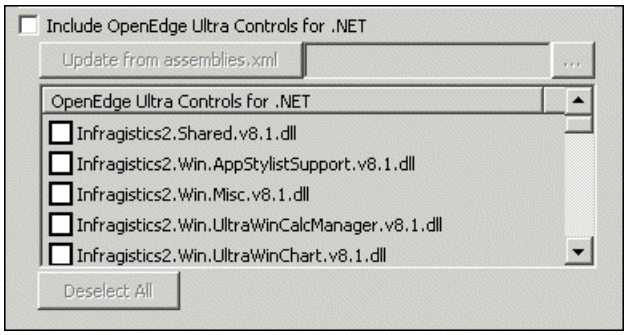
Because WebClient cannot determine what other applications might require the framework, uninstalling WebClient or your WebClient application does not uninstall the .NET Framework.

.NET assemblies

An *assembly* is a .NET library or collection of classes. The WebClient Application Assembler can handle assemblies like other application files, enabling you to package them into components. Your OpenEdge GUI application might use the following types of assemblies that you need to include in your deployment:

- Optional OpenEdge Ultra Controls for .NET
- Classes used by the application that reference .NET objects
- Third-party .NET controls

The **IntelliStream System Tasks Definition** dialog includes functionality to aid you in deploying the assemblies for the run-time versions of the OpenEdge Ultra Controls for .NET:



This functionality reads your application's Assembly References File (`assemblies.xml`) and chooses only those Ultra Controls used in your application for deployment. For more information on using this functionality, see the online help for the WebClient Application Assembler.

Note: Because other applications might require them, uninstalling WebClient or your WebClient application does not uninstall the OpenEdge Ultra Controls for .NET.

You must ensure that you deploy assemblies in the correct location. Progress Software Corporation recommends that you deploy assemblies in the "At Startup" component to ensure that they are available when required. If your application includes third-party assemblies that must be installed in the Global Assembly Cache (GAC), you must include a separate installer for them that the WebClient can launch. For example, you might add an IntelliStream ABL Install Procedure to your WebClient deployment that executes the necessary installers.

For an Administrator installation, the WebClient Initializer defaults to copying the controls to the <*`WebClient-install`*>`\bin` directory. For a personal installation, it defaults to copying the controls to the <*`WebClient-App-install`*> directory. However, if the application's command line includes the Assemblies (`-assemblies`) startup parameter, the Initializer copies the controls to that directory instead for both installation types.

For more information on deploying assemblies, see the chapter on maintaining user environments in *OpenEdge Deployment: Managing ABL Applications*.

Other files

In addition to assemblies, every OpenEdge GUI application requires an Assembly References File (`assemblies.xml`). You must deploy this file in the "At Startup" component to ensure proper execution of your WebClient application.

In exceptional circumstances, your OpenEdge GUI application might use an `assemblies.config` file. You must deploy this file in the "At Startup" component to ensure proper execution of your WebClient application.

For more information on these files, see the chapter on maintaining user environments in *OpenEdge Deployment: Managing ABL Applications*.

Index

.NET
    deploying framework installers *109*
    WebClient considerations *109*

A

ABL for Web-enablement
    PROPATH *55*
ABL install procedure
    WebClient applications *33*
Administrator installations *70–71*
Applications
    compiling *57*
    component files *88*
    components *73*
    components update files *88*
    configuration file *88*
    mode *61*
    registry keys *76*, *78*
    signed and unsigned *91–92*
ASYNC requests *52*
Authentication
    dialogs *47*
Authenticity (of a downloaded file) *41*

B

Best practices
    recommended for WebClient *21*
Bootstrap file
    customizing *104*

C

CD-ROM
    installing WebClient *32*
Compiling applications *57*
Components
    application *73*
    registry keys *78*
Customizing
    bootstrap.htm *104*
    WebClient install *32*
    WebClient installer *97–98*

D

Deploying WebClient applications
    end user documentation *59*
Deployment configuration
    modifying *89*
Documentation
    for end users *59*

E

End user
    providing documentation *35*
External installer
    installation overview *34*
    installation tasks *34*
    registry keys for uninstall *79*
    uninstalling WebClient *35*
    Web-enabled installer *87*
    WebClient application updates *34*
    WebClient applications *33*

F

Features
    IntelliStream *33*
Firewalls
    setting after WebClient install *62*

I

Installation types
    Administrator *71*
    checking *52*
    personal *72*
Installing WebClient applications
    external installer *34*
    IntelliStream *33*
    recommendations *33*
InstallShield One-Click Install
    customizing *98*
Integrity (of a downloaded file) *41*
IntelliStream
    application components *73*
    features *33*
    installation overview *33*
    installing WebClient applications *33*
    uninstalling WebClient *35*
    WebClient application updates *34*

L

LOAD-IMAGE support for URLs
    authentication *55*
LOAD-IMAGE() method *55*

M

Microsoft Vista
  User Access Control *32*
MIME type settings
  Apache Web Server *95*
  configuring for Web servers *95*
  Microsoft IIS *95*
  Sun Java System Web Server *95*
Modes
  WebClient Application Manager *61*
Modifying deployment configuration *89*

O

One-Click Install image
  WebClient *32*
OpenEdge GUI for .NET
  WebClient deployment *109*
Overview
  uninstalling WebClient *35*
  WebClient Application Assembler *17*
  WebClient application updates *34*
  WebClient install *32*

P

Personal installations
  System Tasks *87*
Procedure libraries
  updating *107*
Project file, WebClient (.wcp file) *80*
PROPATH support for URLs
  authentication *55*
Providing documentation
  for end users *35*
.prowcapc file
  modifying *89*
  digitally signed *91*
.prowcapp file
  digitally signed *88*
  modifying existing *89*
ProwcappLocator
  registry keys *77*
Proxy
  -proxyhost startup parameter *62*
  -proxyport startup parameter *62*
  servers, setting after WebClient install *62*

R

Registry keys
  application name *76*, *78*
  applications *78*
  checking programmatically *52*
Registry keys *(continued)*
  components *78*
  external installer *79*
  ProwcappLocator *77*
  suppressing WebClient application dialog *107*
  vendor name *76*
  WebClient *74*
  WebClient installer *74*
Running applications
  Web access *59*

S

Sample application *24*
SEARCH function support for URLs
  authentication *55*
Security
  designing *41*, *47*
  for custom deployment configuration *92*
SESSION system handle
  WC-ADMIN-APP attribute *52*
Setting MIME type
  Apache Web Server *95*
  Microsoft IIS *95*
  Sun Java System Web Server *95*
Shortcuts
  Start WebClient applications *35*
Single sign-on *46*, *53*
Source code versioning *57*
SportsPro application *24*
Starting WebClient applications *35*
Startup parameters
  proxy host (-proxyhost) *62*
  proxy port (-proxyport) *62*
Suppressing WebClient application
  updating *107*
System tasks
  defining *85*
System Tasks
  personal install limits *87*
  .NET *109*

T

Third-party installers
  assemblies *111*
  .NET Framework *109*

U

Uninstalling WebClient
  external installer *35*
  IntelliStream *35*
  overview *35*

Uninstalling WebClient applications
registry keys 79
Updates
procedure libraries 107
suppressing WebClient application dialog 107
URL support
PROPATH 55
starting WebClient applications 35
User Access Control (UAC)
WebClient installs 32

V

Vendor name
registry keys 76
Versioning
source code 57

W

WC-ADMIN-APP attribute 52
.wcd file 89
.wcp file 80
Web access
running applications 59
WebClient applications 59
Web install
WebClient applications 33
Web servers
configuring MIME types 95
Web-enabled installer
external installer 87
WebClient
CD-ROM install 32
end-user requirements 17
One-Click install image 32
project (.wcp) file 80
registry keys 74
WebClient Application Assembler
generating files 73
managing applications 69
overview 17
WebClient Application Manager
Applications tab 63
defined 61
edit dialog 63
functions 61
General tab 64
WebClient Application Manager *(continued)*
Log tab 66
modes of execution 61
setting a firewall 62
shortcut syntax 74
WebClient tab 62
WebClient application updates
external installer 34
IntelliStream 34
overview 34
suppressing dialog 107
WebClient applications
ABL install procedure 33
best practices 21
external installer 33
installation recommendations 33
installing 33
installing with IntelliStream 33
launch mechanism overview 101
OpenEdge GUI for .NET 109
starting 35
WebClient Deployment Packager
overview 18
WebClient deployments
.NET Framework installers 109
WebClient Initializer
defined 101
shortcut syntax 74
validating acceptable versions
Progress WebClient 106
WebClient install
customizing 32
overview 32
WebClient installations
in Vista 32
WebClient installer
customizing 98
registry keys created 74
running application installer example 98
running WebClient Initializer
example 98
WebClient mode 61
WebClient platform
defining 23
running 24
WebClient Platform 22
webclient.htm
customizations 98
customizing a WebClient installer 98