Progress®



Learn About Open Clients

OpenEdge®

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

Updated: 2021/06/29

Table of Contents

Preface....7

Introduction to Open Clients....9

Open Client Toolkit....11
Design Open Client applications....12
Build Open Client applications....14
Deploy Open Client applications....17
Update Open Client applications....17

Build and deploy an Open Client application....19

Generate Proxies....21

Open Client interfaces....22
Objects in an Open Client interface....22
Session models....23
Make procedures available to the interface....24
Get started with ProxyGen....24
Versions of ProxyGen....25
Use ProxyGen execution options....26
Use ProxyGen....26
Edit an Open Client interface....27
Specify generation settings....27
Save Open Client interfaces in a project file....28
Generate an Open Client proxy or Web service definition....28
Run batch ProxyGen....28
Convert a project file from Progress Version 9 to OpenEdge....29
Define an Open Client interface using ProxyGen....30
Define an Open Client interface using ProxyGen....30
Define an AppObject or SubAppObject....31
Create AppObject and SubAppObject entries....31
Specify AppObject and SubAppObject definitions....31
Define procedures in AppObjects and SubAppObjects....33
Set PROPATH components....34
Add and delete procedure files....34
Change the procedure execution mode (type) 35
Customize procedure access....35
Customize method and ProcObject definitions (optional)....35

Customize method definitions in a ProcObject (optional)....37
Change the PROPATH setting for existing procedures....38
Save the Open Client interface definition in a project file....38
Set the project work directory....39
Save the project file....39
Specify generation settings....39
Dialog box layout....39
General settings for proxy generation....40
.NET client details....41
Java client details....43
Sonic Native Invocation client details....44
Sonic Web Service client details....45
Web Service client details....46
Validate and generate an Open Client proxy....48
Open Client interface validation....49
Open Client interface generation....49
Object naming in ProxyGen....52
Open Client interface naming conventions....52
Automatic name conversions....53
Naming side effects and restrictions....53
Example....54

Programming Concepts....57

Connect to an application server....58
Get user input for run-time connection information....58
Proxy and Web service object methods....59
Remote ABL methods....60
Pass parameters....61
Handle return values....61
Pass parameters....61
Access Open Client run-time properties....61
Handle errors in an Open Client application....62
Client-side errors....62
Server-side errors before remote ABL method execution....62
Server-side errors during remote ABL execution....62
Access an application server directly without proxies....63

Index....65

Preface

Purpose

The purpose of this content is to introduce the following concepts:

- Open Clients
- Open Client Toolkit
- Open Client development process
- Configuration
- Proxy generation
- Use certificates
- Common programming concepts

Audience

ABL developer and Open Client developers who will work together to build Open Client applications built in .NET or Java that run ABL code running on a Progress Application Server (PAS) for OpenEdge instance.

Organization

Introduction to Open Clients on page 9

Introduces the types of Open Clients, defines interactions with the application server to run ABL business logic, the Open Client Toolkit for generating proxies and the design, build, deploy and update steps in the Open Client development process.

Configuration

Provides details for Building and deploying Open Client applications.

Generate Proxies on page 21

Expands on the introduction to review the use of ProxGen to generate proxy files. Introduces more details of AppObjects, SubAppObjects and procObjects.

Programming Concepts on page 57

Programming considerations for Open Client and ABL develops including connections to the application server, methods, parameters, run-time properties, error handling and direct access with Open APIs.

Documentation conventions

See Documentation Conventions for an explanation of the terminology, format, and typographical conventions used throughout the OpenEdge content library.

Purpose

The purpose of this content is to introduce the following concepts:

- Open Clients
- Open Client Toolkit
- Open Client development process
- Configuration
- Proxy generation
- Use certificates
- Common programming concepts

Audience

ABL developer and Open Client developers who will work together to build Open Client applications built in .NET or Java that run ABL code running on a Progress Application Server (PAS) for OpenEdge instance.

Organization

Introduction to Open Clients on page 9

Introduces the types of Open Clients, defines interactions with the application server to run ABL business logic, the Open Client Toolkit for generating proxies and the design, build, deploy and update steps in the Open Client development process.

Configuration

Provides details for Building and deploying Open Client applications.

Generate Proxies on page 21

Expands on the introduction to review the use of ProxGen to generate proxy files. Introduces more details of AppObjects, SubAppObjects and procObjects.

Programming Concepts on page 57

Programming considerations for Open Client and ABL develops including connections to the application server, methods, parameters, run-time properties, error handling and direct access with Open APIs.

Documentation conventions

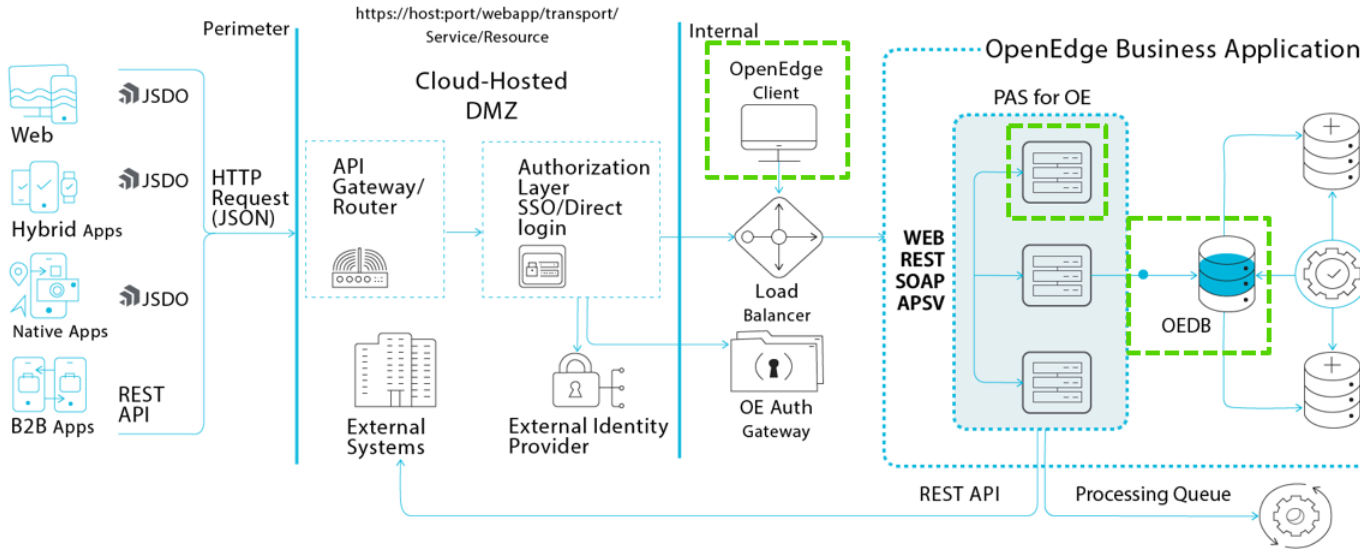
See Documentation Conventions for an explanation of the terminology, format, and typographical conventions used throughout the OpenEdge content library.

1

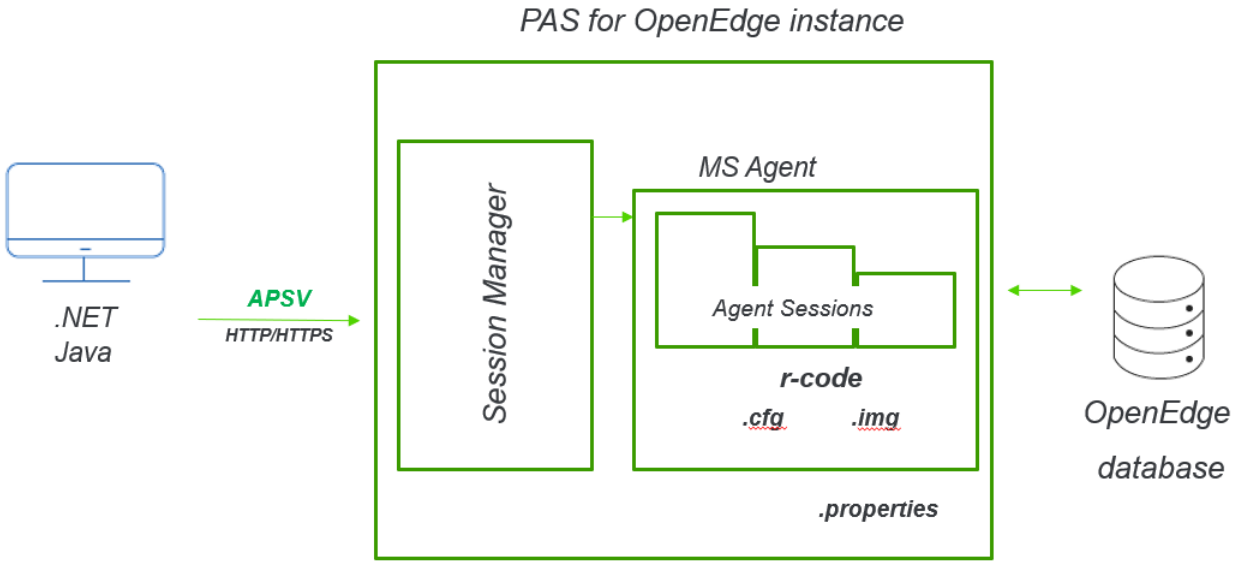
Introduction to Open Clients

Open Clients are a way for developers to connect ***.NET*** or ***Java*** client application to ABL business logic. ***ABL developers*** can expose their ABL applications through ***proxy*** files. These files provide native classes that .NET and Java developers use to interact with ABL applications. Java and .NET developers use their native development tools when building their application based on the proxy files. Alternatively, .NET and Java developers can opt to develop using the OpenAPI instead of generated proxy files..

In this architecture diagram, .NET and Java Open Clients behave like OpenEdge clients connecting to a PAS for OpenEdge instance to access ABL business logic.



Open Clients use ***HTTP/HTTPS*** requests to communicate with the application server. The requests are transformed to ABL requests with the help of the ***Open Client runtime***. The Open Client runtime is part of the client installation that you must supply with your .NET and Java applications. OpenEdge packages and distributes the Open Client Runtime in several file formats containing different network protocol support combinations. This distribution method reduces the overhead and complexity for both you the developer and the Open Client deployer. For .NET and Java applications, requests are sent using the APSV transport, the same transport used by ABL clients. In this diagram, Open Clients are treated the same as an OpenEdge client communicating on the APSV transport accessing r-code on the PROPATH of the application server.



A key feature of the Open Client runtime is the support for relational data exchange. This feature allows ABL ProDataSets and temp-tables to be passed between the application server and the Open Client applications. By making this seamless for Open Client developers, their applications can access any OpenEdge database or DataServer connected to the application server business logic whether you generate proxies using ProxyGen or develop using the OpenAPI.

For details, see the following topics:

- Open Client Toolkit
- Design Open Client applications
- Build Open Client applications
- Deploy Open Client applications
- Update Open Client applications

Open Client Toolkit

The Open Client Toolkit includes ProxyGen utilities for generating proxy objects and the Open Client Runtime to access the application server. The Open Client Runtime provides support to client developers to build their client with proxies or with the OpenAPI.

Use ProxyGen

ABL developers use ***ProxyGen*** to generate proxies, or native client objects to access the ABL application (r-code) and the associated internal procedures and functions hosted on the application server. ABL developers control how much of the underlying application an Open Client developer can access. ABL developers must have access to the .r files to run the ProxyGen tool.



ProxyGen has a graphical version and a batch version. The graphical tool allows you to specify which parts of the backend you want to expose, and then it builds the proxies. It also creates a `.xpxg` file (independent of the proxies) which saves all the information that defines your proxies. Batch proxygen takes as input an `.xpxg` file and generates/regenerates for those ABL developers who want to automate the running of ProxyGen when underlying ABL code changes are checked into a source code control system.

Note: In order to use ProxyGen, you need to have Java installed locally. For more information, see the *OpenEdge Installation Guide*.

Proxy files

Open Client developers do not need to have any knowledge of how to write ABL applications. The proxy files provide the native classes that allow the Open Clients to communicate with a specific ABL application and its associated procedures and functions. The generated file types are unique for each Open Client type.

OpenAPIs

For .NET and Java Open Clients, OpenEdge provides a means for you to access application services directly on an application server without the need to generate proxies using ProxyGen. You can do this using an API (OpenAPI) provided with each Open Client.This Open Client OpenAPI provides a set of classes for you to access application services using general access methods. Because these classes know nothing about the application service they are accessing, you must be familiar with all the requirements for accessing the external procedures, persistent procedures, single-run procedures, singleton procedures, internal procedures, and user-defined functions provided by the application service. With this information, you must set up the procedure and user-defined function signatures according to application service requirements in your client code at run time. Using the OpenAPI may provide more freedom to access the underlying ABL code, but it is typically at the expense of productivity. For more details see the OpenAPI content in the respective .NET and Java Open Client content.

Open Client Toolkit generates other file types

Originally designed to generate proxies for Open Clients, the Open Client Toolkit includes functionality for Web Services, Aurea SonicMQ® and Sonic ESB® not addressed in the Open Client content.

For more information on Web Services, see the Web UI developer and Service developer content.

For more information on Aurea products, see the Aurea documentation.

For information on ordering Open Client Toolkit, see the OpenEdge 12 Product Availability Guide.

If you support Open Clients on a classic AppServer configuration, search the 11.7 Documentation for Open Client products and configurations.

Design Open Client applications

Open Client applications require that non-ABL developers and ABL developers design the application together. Like two contractors working on the same home, they must plan and communicate their specifications so that the finished application works seamlessly for their users.

The following considerations must be agreed upon ahead of time:

- Client type
- Session type
- Business logic
- Data format
- Security model

Identify the client type

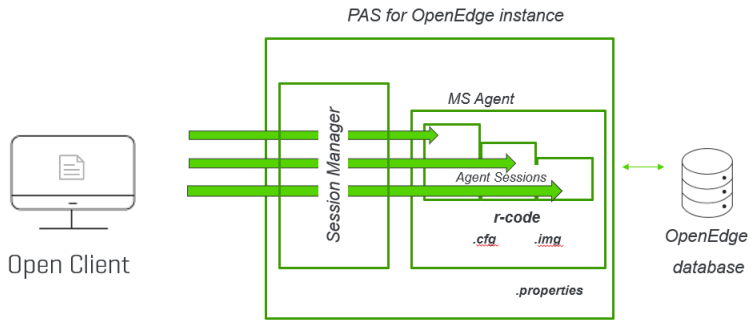
Each client type requires a different proxy to be generated. Additionally, each client type has recommendations for best coding practices. For that reason, once the client type is determined, both the Open Client developer and ABL developer need to review the content specific to each client type to raise awareness about what is and is not possible for each type.

- .NET Open Client development
- Java Open Client development

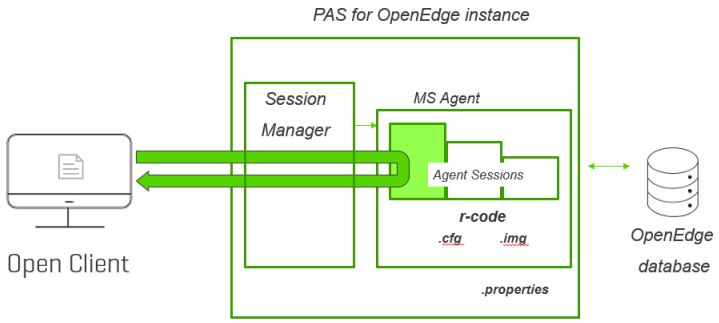
Identify the session type

The session type for a client connection determines how context is or is not shared between the client and server code. Clients connect to the application server using one of two session types, session free or session managed. Open Clients have this same option.

Session free means that a client is not bound to a single session. Subsequent requests from a client can be handled in different agent session.



Session managed means that a client is bound to a single session. Context is maintained between client requests. Subsequent client requests are handled in the context of the same session. Coding for session managed requires more effort to maintain this context.



A PAS for OpenEdge instance can respond to both session managed and session free client requests. When you generate your proxies, you choose the type of session management for your Open Client interaction. A PAS for OpenEdge instance includes the client run-time licensing needed to execute the r-code.

Identify business logic

ABL developers control how much of the underlying ABL code is exposed to the Open Client developers. For this reason, it is important to have discussions around what is needed to build the proxy between the client and server logic. If all the necessary procedures and functions are not exposed in the process, then the testing phase takes longer. ABL developers are required to regenerate proxy files until all the necessary ABL code are exposed in the proxy.

Identify data types to use

A key feature of the Open Client is support for relational data exchange. This feature allows ABL ProDataSets and temp-tables to be passed between the server and the client application which uses the data in its native environment.

Table 1: Data type mappings

ABL Data type	Maps to .NET	Maps to Java
ProDataSet parameter (DATASET or DATASET-HANDLE data type)	ADO.NET DataSet object	ProDataGraph object which is an implementation of the Java Service Data Objects (Java SDO) DataGraph interface
ProDataSet parameter (DATASET or DATASET-HANDLE data type) Temp table parameter (TABLE or TABLE-HANDLE data type)	ADO.NET DataTable object	Developer can choose to map to: • An OpenEdge ProDataGraph object, as the wrapper for a single temp-table • SQL ResultSet object for backwards compatability

Identify the security model

Since communication occurs between clients and the application server using HTTP, both development teams should agree upon the security model they implement. For data in transit, securing the HTTP requests using HTTPS is highly recommended. Additionally, user authentication and authorization techniques help to limit user access to only those parts of the application that meet their specify user or group permission. For details regarding recommendations for securing your applications see the Security content.

Build Open Client applications

Based on the work completed in the pre-planning phase, both the ABL developer and Open Client developer will have a roadmap of the work to be done. Development can start in parallel and will require less rework if each developer completes the work to which they have committed. The steps to building an Open Client application include:

- Develop ABL code
- Generate and share proxy files
- Develop Open Client code

Develop ABL code

The ABL developer will either expose existing code, ABL procedures based on the agreed upon business logic requirements or they may opt to develop new code. In either case, ProxyGen is used to create Open Client object types that can be used in .NET or Java user interfaces.

For Open Clients, the client code accesses the proxy in the client's native environment. As a result, a .NET client sees the proxy as a set of .NET classes, while a Java client sees the proxy as a set of Java classes. To use proxy objects, you must deploy the generated proxy along with the Open Client Runtime to all client application development and deployment systems. Any single Open Client interface maps an application to a series of object types.

Before building your client application, you must prepare your environment. The preparation depends on the build environment and the client type. After building your application, you can deploy your client application according to its Open Client type.

For more information, see *.NET Open Client*s or *Java Open Clients*.

Generate and share proxy files

There are two versions of ProxyGen:

- **ProxyGen** — The full version of ProxyGen, available on Windows
- **Batch ProxyGen** — On Windows and UNIX, a subset of ProxyGen functionality is available as a console application, called Batch ProxyGen. Batch ProxyGen operates only on previously saved Open Client interface definitions.

ABL developers must use ProxyGen (the full version) to define an Open Client interface for an Open Client proxy. Later, ABL developers can use either ProxyGen or Batch ProxyGen to generate the actual Open Client proxy. These proxy files are then sent to Open Client developers to use when developing .NET or Java applications.

To define an Open Client interface, ProxyGen requires that you organize an application service into three types of proxy objects:

- Application objects (AppObjects)
- Sub-application objects (SubAppObjects)
- Procedure objects (ProcObjects)

Table 2: Open Client object types

Object type	Occurrences	Description
AppObject	One required per application	Encapsulates zero or more external ABL procedures as directly callable methods and zero or more ProcObjects as class factory methods

Object type	Occurrences	Description
SubAppObject	Zero or more per application	Encapsulates zero or more external ABL procedures as directly callable methods and zero or more ProcObjects as class factory methods
ProcObject	Zero or more per applications	Maps to a persistent, single-run, or singleton procedure that encapsulates one or more ABL internal procedures and user-defined functions as directly callable methods

Organize code using SubAppObjects

The minimum requirement is one AppObject that encapsulates all external procedures for the application. SubAppObjects allow you to organize these external procedures into collections you find useful for your particular application. ABL developers may use SubAppObjects to help group functionality and limit access based on permissions.

Use ProcObjects for persistent, single-run or singletons

A ProcObject maps to an ABL external procedure that is executed as persistent, single-run, or singleton. The procedure's internal procedures and functions are then available for access by callers external to the procedure that defines them. On the application server, these callers can include other external ABL procedures that execute in the same OpenEdge session, or they can include clients of an application server including Open Clients that call these internal procedures and functions as methods.

ProxyGen maps each persistent, single-run, or singleton procedure that you include in an Open Client interface to a corresponding ProcObject, with the internal procedures and functions becoming its methods.

For session free applications, ProcObjects that encapsulate persistent procedures have limited utility and tend to interfere with session free application performance. Part of the value of persistent procedures is that they provide a means to maintain run-time context between the client and the application server. Much of the value from session free application services is that application server resources are more readily available to serve client requests. However, a persistent procedure binds an application server resource to the client that instantiates the procedure until the client deletes the procedure. This ties up a physical connection and reduces the application server resources available for requests from other clients, which can in turn reduce apparent application performance across the client domain.

Invocation determines how ABL procedures are run

ProxyGen allows you to distinguish between non-persistent, persistent, single-run, and singleton procedures for all external procedures that you include in your Open Client interface. ABL makes an external procedure non-persistent, persistent, single-run, or singleton by the way you invoke the procedure, not by the procedure's definition. You can include any OpenEdge external procedure in an Open Client interface as non-persistent, persistent, single-run, or singleton, with the restriction that a procedure with parameters in its main block cannot be run as single-run or singleton. For Open Client proxies, however, access to the functionality of a non-persistent procedure is very different compared to a ProcObject, and it requires different object management, depending on the session model and object types involved.

The net result of running ProxyGen is to create a project file, or `.xpxg` file. This file includes the list of all selected procedures and any customizations you applied to those procedures, for each Open Client Interface object definition.

Develop Open Client code

.NET and Java developers will use native tools to build the applications based on the proxies provided. For this content see the sample provide in the following topics.

Build .NET Open Client applications

Deploy Open Client applications

Open Clients go through the same deployment phases as other ABL applications.

- Deploy Open Client applications
- Secure Open Client applications
- Test Open Client applications.

Deploy Open Client applications

Open Client deployments require client code deployment and application server code deployment. The client deployment details are addressed in the following topics.

- Deploy .NET Open Clients
- Deploy Java Open Clients

Deployment of ABL business logic on the application server is the same as other ABL application deployments. For Open Client developers who want an overview of PAS for OpenEdge, see 'PAS for OpenEdge overview' in *Develop ABL Services*. For system administrators who will be responsible for deploying ABL applications, see *Learn about Progress Application Server for OpenEdge* and Use PAS for OpenEdge in production.

Secure Open Client applications

Open Clients use HTTP and HTTPS connectivity to exchange data. To protect data in transit, Open Client developers will use HTTPS connections. This requires configuring both the Open Client and ABL business logic hosted on the application server to use the same HTTPS configuration. For more information, see:

- 'Use TLS in PAS for OpenEdge' in *Manage Progress Application Server (PAS) for OpenEdge*
- 'Secure server connections to .NET Open Clients' in *.NET Open Clients*
- 'Secure server connections to Java Open Clients' in *Java Open Clients*

Test Open Client applications

It is important to test the Open Client code against a production staging instance of the application server to verify that application is working as expected with the correct security configuration.

Update Open Client applications

Updates to Open Client applications require that new proxy files are created by ABL developers and distributed to Open Client developers (.NET and Java developers). Changes in exposed business logic signatures and data types will require generating new proxies. If a change is in code that is not exposed directly to the Open Client developers in a proxy, for example an internal procedure called by an exposed procedure, new proxies are not required.

2

Build and deploy an Open Client application

For Java and .NET clients, the Open Client Runtime is part of the client installation you must supply with your application. OpenEdge packages and distributes the Open Client Runtime in several file formats containing different network protocol support combinations. This distribution method reduces the overhead and complexity for both you the developer and the Open Client deployer.

Before running ProxyGen to generate proxies, you must follow specific preparation procedures based on your environment and client type.

Before building your client application, you must prepare your environment. The preparation depends on the build environment and the client type. After building your application, you can deploy your client application according to its Open Client type.

For more information on all the above topics, see *.NET Open Clients*or *Java Open Clients*.

3

Generate Proxies

This content is a general overview of ProxyGen. ProxyGen is the tool for generating proxies for Java or .NET Open Client applications. This content explores the available Open Client interfaces and provides an overview of the tool for creating those interfaces. In the process, the concepts of AppObjects, SubAppObjects and ProcObjects are explained in greater detail. The options for configuring default generation settings are presented as well as saving, validating and generating proxy files. The final section provides conceptual use case as an example .

For details, see the following topics:

- Open Client interfaces
- Get started with ProxyGen
- Define an Open Client interface using ProxyGen
- Define an AppObject or SubAppObject
- Define procedures in AppObjects and SubAppObjects
- Save the Open Client interface definition in a project file
- Specify generation settings
- Validate and generate an Open Client proxy
- Object naming in ProxyGen
- Example

Open Client interfaces

ProxyGen generates a definition for an OpenEdge Web service or an Open Client proxy as a collection of objects that specify an *Open Client interface*. These objects conform to the Open Client object model. This model organizes and maps OpenEdge procedures that run on an application server so they are readily callable from any supported Open Client, such as a Java or a .NET client, or any other client that can access a SOAP Web service.

Objects in an Open Client interface

Any single Open Client interface maps an application server application to a series of object types, as shown in the following table.

Table 3: Objects in an Open Client interface

Object Type	Occurrences	Description
AppObject	One required per application	Encapsulates zero or more external ABL procedures as directly callable methods and zero or more ProcObjects as class factory methods
SubAppObject	Zero or more per application	Encapsulates zero or more external ABL procedures as directly callable methods and zero or more ProcObjects as class factory methods
ProcObject	Zero or more per application	Maps to a persistent, single-run, or singleton procedure that encapsulates one or more ABL internal procedures and user-defined functions as directly callable methods

AppObject and SubAppObjects

The AppObject and any SubAppObjects defined for the application each encapsulate the same type of functionality, consisting of external procedures and ProcObjects. In ABL, an external procedure is any single file containing separately executable ABL code. A ProcObject directly maps to a persistent, single-run, or singleton procedure. So, all the functionality encapsulated by the AppObject and SubAppObjects consists entirely of one or more external procedures.

The minimum requirement is one AppObject that encapsulates all external procedures for the application. SubAppObjects allow you to compartmentalize these external procedures into collections you find useful for your particular application.

ProcObjects

A ProcObject maps to an ABL external procedure that is executed as persistent, single-run, or singleton. The procedure's internal procedures and functions are then available for access by callers external to the procedure that defines them. On the application server, these callers can include other external ABL procedures that execute in the same OpenEdge session, or they can include clients of an application server (including Open Clients and SOAP Web service clients) that call these internal procedures and functions as methods.

ProxyGen maps each persistent, single-run, or singleton procedure that you include in an Open Client interface to a corresponding ProcObject, with the internal procedures and functions becoming its methods.

ProcObjects or procedures?

ProxyGen allows you to distinguish between non-persistent, persistent, single-run, and singleton procedures for all external procedures that you include in your Open Client interface. ABL makes an external procedure non-persistent, persistent, single-run, or singleton by the way you invoke the procedure, not by the procedure's definition. You can include any OpenEdge external procedure in an Open Client interface as non-persistent, persistent, single-run, or singleton, with the restriction that a procedure with parameters in its main block cannot be run as single-run or singleton. For Open Client proxies, however, access to the functionality of a non-persistent procedure is very different compared to a ProcObject, and it requires different object management, depending on the session model (see Session models on page 23) and object types involved.

Encapsulate functionality

For an Open Client proxy or a SOAP Web service definition, all three types of objects—AppObject, SubAppObject, and ProcObject—encapsulate functionality as object methods. All these methods are callable by Open Clients and Web service clients. Only the ProcObject, however, always requires a persistent session context in which to make its methods available.

When you choose external procedures to include in an AppObject or a SubAppObject, you must clearly understand the intent of each procedure—those intended to be called by methods and those intended to be accessed via ProcObjects. Any procedure designed to run as persistent, single-run, or singleton is typically mapped as a ProcObject. ProcObject definitions never stand alone but always must be included as part of an AppObject or a SubAppObject definition in the Open Client interface.

Also, for every Open Client object you define, ProxyGen defines a set of common methods for managing that object. Most of these built-in methods do not execute procedures in the application server application; however, the client application must use these methods to create and manage the objects according to their Open Client object relationships. Depending on the session model and the object type, this management can have an impact on client and application server performance.

For more information on how these built-in methods work with each object type, see Programming Concepts on page 57 and *OpenEdge Development: Java Open Clients*, *OpenEdge Development: .NET Open Clients*, or *OpenEdge Development: Web Services*.

Session models

ProxyGen supports two session models for Open Client interfaces—session-managed and session-free. The session model is determined by how the application server is configured and fundamentally determines how an Open Client and application server can communicate with one another.

Session models and Open Clients

If the chosen model is session-managed, a given client has a persistent physical connection to the application server over which context can be managed. If the chosen model is session-free, a given client has no persistent physical connection to an application server for managing context, but a logical connection that associates the client with the application (or Web) service supported by one or more application server resources.

For a complete definition of session models and how they affect the interaction between an application server and a client, see *OpenEdge Application Server: Developing application server Applications*.

Specify the session model

In ProxyGen, you only need to specify the session model explicitly for SOAP Web services and Sonic ESB, because ProxyGen generates a different interface definition depending on the session model. (The default is session-free.) For .NET and Java Open Clients you do not need to specify the session model because the generated proxy is identical for both session models. It is the responsibility of the Open Client application to specify the session model as a run-time session property for connection to either an application server (session-managed) or an application service (session-free).

The session model is really a property of the application service, and the application server enables support for and determines the session model for the application service by the setting of the application server operating mode. The state-aware, state-reset, and stateless operating modes each specify support for the session-managed model; the state-free operating mode, alone, specifies support for the session-free model. The entire application, including the client and application service, must be programmed differently depending on the session model. The model affects your choice of object types used to define the Open Client interface. For more information, see *OpenEdge Application Server: Developing Application Server Applications*.

Make procedures available to the interface

ABL applications rely on a PROPATH environment variable to tell a given program where, in file-system storage, it can find an external procedure to execute at run time. This PROPATH value consists of path components, similar to an operating system `Path` or Java `Classpath`. Each component of the PROPATH is an absolute or relative pathname to a folder where external procedure files for the application can reside. You can also use dot (`.`) as a PROPATH component.

Similarly, for each AppObject and SubAppObject you define in ProxyGen, you must also specify the PROPATH components that ProxyGen requires to locate the external procedures for the object in its own working environment. You then select from the procedures available for each object's PROPATH to include as methods or as ProcObjects in the given AppObject or SubAppObject. Each AppObject and SubAppObject can specify the same PROPATH components, or each object can specify one or more unique PROPATH components. If the PROPATH components follow a logical organization, it might be useful to map those components to objects, but there is no required mapping between objects and PROPATH components.

Note: The PROPATH component settings in ProxyGen are not necessarily the same as the run-time PROPATH settings on the application server. In ProxyGen, you specify them only to locate procedures in the ProxyGen environment for defining the Open Client interface, not to execute the procedures on the application server. However you set up your PROPATH, you must ensure that the relative paths under the PROPATH component settings remain the same between the application server and ProxyGen.

Get started with ProxyGen

ProxyGen is a development tool that defines and generates Java and .NET proxies and generates SOAP Web service definitions. While this chapter provides basic guidance on using ProxyGen, refer to the online help for complete information on each ProxyGen window.

The following sections provide information about:

- Versions of ProxyGen on page 25
- Use ProxyGen execution options on page 26
- Use ProxyGen on page 26

- Edit an Open Client interface on page 27
- Specify generation settings on page 27
- Save Open Client interfaces in a project file on page 28
- Generate an Open Client proxy or Web service definition on page 28
- Run batch ProxyGen on page 28
- Convert a project file from Progress Version 9 to OpenEdge on page 29

Versions of ProxyGen

There are two versions of ProxyGen:

- **ProxyGen** — The full version of ProxyGen, available on Windows, provides a GUI to:
  - Define an Open Client interface for Java and .NET Open Client proxies and SOAP Web service definitions
  - Save Open Client interface definitions in a project file
  - Update Open Client interface definitions
  - Generate and validate proxies for Java or .NET Open Clients and SOAP Web service definitions, from a new or saved Open Client interface definition
- **Batch ProxyGen** — On Windows and UNIX, a subset of ProxyGen functionality is available as a console application, called Batch ProxyGen. Batch ProxyGen provides similar functionality to ProxyGen, but it does not allow Open Client interfaces to be defined and saved, as you can do using the full version of ProxyGen. Instead, Batch ProxyGen operates only on previously saved Open Client interface definitions, as follows:
  - In Windows, Batch ProxyGen can generate Java and .NET Open Client proxies and SOAP Web service definitions.
  - On UNIX, Batch ProxyGen can generate only Java Open Client proxies and SOAP Web service definitions.

You must use ProxyGen (the full version) to define an Open Client interface for an Open Client proxy or Web service definition that access ABL procedures. You can use either ProxyGen or Batch ProxyGen to generate the actual Open Client proxy or Web service definition.

The following table shows how to complete these basic tasks with ProxyGen and Batch ProxyGen.

Table 4: ProxyGen and Batch ProxyGen tasks

Task	ProxyGen	Batch ProxyGen
Edit an Open Client interface on page 27	X	–
Specify generation settings on page 27 , including client types	X	–
Save Open Client interfaces in a project file on page 28	X	–
Generate an Open Client proxy or Web service definition on page 28	X	X

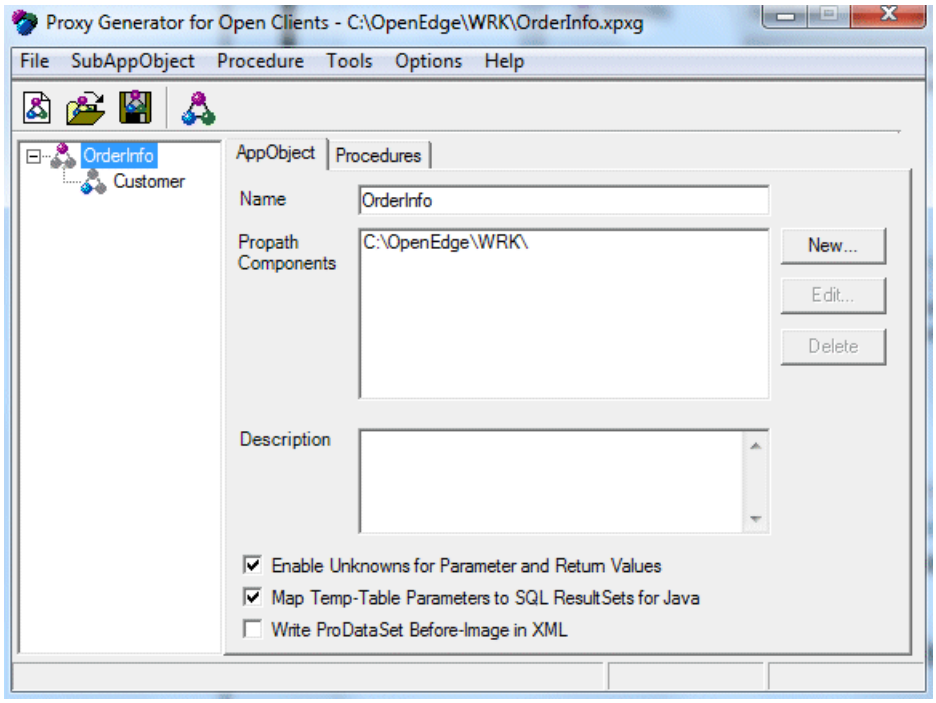
Use ProxyGen execution options

You can modify the operation of ProxyGen and Batch ProxyGen using ProxyGen startup options, specified in the shortcut icon for ProxyGen. You can view and edit the shortcut by selecting the icon, then right-clicking and selecting **Properties**. You can edit the following **Target** information on the icon properties:

- **`/wrkdir`** ***`dir`*** — Specifies the project work directory for ProxyGen. This is the default location for ProxyGen project files. The default is the OpenEdge work directory you specify during installation of the Open Client Toolkit. (Do not change the **Start in** field instead; this has no effect on ProxyGen operation.)
- **`/preffile`** ***`preffile`*** — Specifies the file for ProxyGen generation preferences. The default is *`OpenEdge-install-directory`*`\properties\proxygen.xml`. You can create a local preference file by copying `proxygen.xml` to a local directory and using this option for ProxyGen to use the local preferences file.

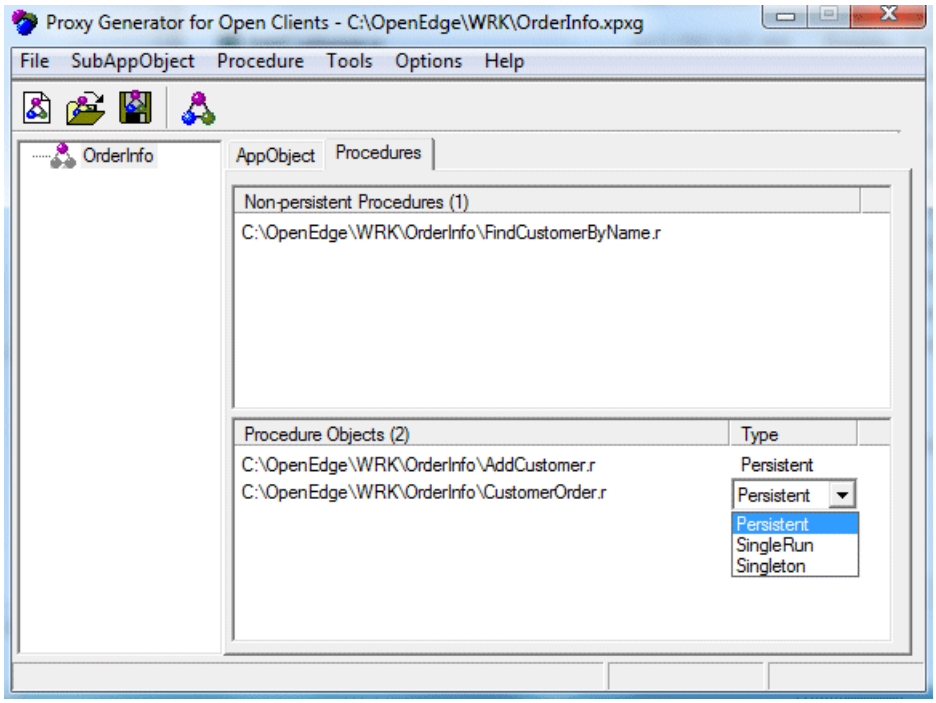
Use ProxyGen

Start ProxyGen on Windows by opening the **Proxy Generator** icon in the **OpenEdge** program group. The **ProxyGen** main window then opens, as shown in the following figure.



The tree view in the figure shows a previously defined Open Client interface consisting of the **OrderInfo** AppObject and one SubAppObject, **Customer**. The **OrderInfo** AppObject is selected, with its definition showing on the **AppObject** tab.

The **Procedures** tab of the ProxyGen main window shows non-persistent procedures and procedure objects, which can be persistent, single-run, or singleton procedures, as shown in the following figure. You can change how a ProcObject will be instantiated using the pull-down menu next to each procedure name. For more information on adding procedure files and setting their types, see Add and delete procedure files on page 34.



Note: For the purposes of this book, *OpenEdge Development: Java Open Clients*, and *OpenEdge Development: .NET Open Clients*, the term "non-persistent procedures" does NOT include single-run or singleton procedures.

Edit an Open Client interface

The **ProxyGen** main window provides a tree view to navigate through Open Client objects as you define them. The root (top) object in the tree view always is the AppObject definition for the Open Client interface (for example, **OrderInfo**). Any child nodes in the tree view (for example, **Customer**) are SubAppObjects associated with this AppObject. Depending on your selection in the tree view, ProxyGen displays tab folders that show the definition for the selected object. You complete the Open Client interface definition by entering information in the tab folders and using the options available from the menu bar and toolbar.

To edit an existing Open Client interface, open the project file (with extension `.xpxg` in OpenEdge Releases 10 or later, or `.pxg` in Version 9) containing a given Open Client interface. To open this file, use the ProxyGen **File** menu or open the project file directly in Windows Explorer. This *project file* serves as a repository for a single Open Client interface definition to generate all Open Client interfaces.

To convert from a Version 9 `.pxg` file to an OpenEdge Release 10 or later `.xpxg` file, see Convert a project file from Progress Version 9 to OpenEdge on page 29.

Specify generation settings

Among the options you can specify to complete an Open Client interface are the generation settings that determine how you want to generate specific Open Client proxies and/or a SOAP Web service definition. This information becomes part of the Open Client interface you define using ProxyGen. You can set and change these generation settings during a ProxyGen session. For more information, see Specify generation settings on page 39.

Save Open Client interfaces in a project file

ProxyGen saves an Open Client interface definition in the project file. You can also save the project file either explicitly or automatically when you generate the specified Open Client proxies and/or a SOAP Web service definition. Once you save the project file, use ProxyGen or Batch ProxyGen to generate. For more information on saving project files, see Save the Open Client interface definition in a project file on page 38.

Generate an Open Client proxy or Web service definition

Open Client interface generation validates the information in the project file against the application server r-code and then generates the specified Open Client proxies and/or SOAP Web service definition. If you make subsequent changes to the application server code after the project file is saved and then run ProxyGen to regenerate, ProxyGen automatically incorporates the changes based on a set of rules. For details, see Validate and generate an Open Client proxy on page 48.

If this automatic update does not produce acceptable results, you must run ProxyGen and update the Open Client interface definition manually. In addition, as application server procedure files are created or deleted, you must update the definition of the Open Client interface. Then, you can generate the Open Client proxy or Web service definition from the newly saved project file using either ProxyGen or Batch ProxyGen.

Note: ProxyGen uses information in the headers of r-code files to generate proxies. Starting in OpenEdge Release 10.1C, the headers include new information to better support XML features. In particular, if the r-code includes a ProDataSet with a NAMESPACE-URI attribute, the header includes a header version number that is independent of the r-code version. When the header version number is present, you cannot use the r-code file with the ProxyGen from a previous release. Versions of ProxyGen before Release 10.1C misread this new element as an unknown code page and return an error.

Before running ProxyGen to generate proxies, you must follow specific preparation procedures based on your environment and client type. For more information on all the above topics, see *.NET Open Client*sor *Java Open Clients*.

Run batch ProxyGen

Batch ProxyGen runs on supported UNIX and Windows platforms. It allows you to generate Open Client proxies and/or a SOAP Web service definition based on the Open Client interface you defined in a project file created using ProxyGen.

To run Batch ProxyGen, enter the following command in the Windows or UNIX command prompt:

Table 5:

```
bproxygen -xpxgfile project-filename.xpxg [ -useWildCard ] [ -leaveproxyfiles ]
```

Where *project-filename* is the name and path of the OpenEdge project file containing the Open Client interface.

This validates the Open Client interface defined in the project file and generates the specified Open Client proxies and/or Web service definition.

If a very large proxy fails to build, the cause might be the length of the compile command line. Specifying the `-useWildCard` option shortens the command line by using a wildcard to include all `.cs` files in the proxy directories. If you use this option, you should specify an output directory that does not contain any `.cs` files. Any `.cs` files in the output directory are included in the proxy DLL.

If you specify the `-leaveproxyfiles` option, Batch ProxyGen saves the generated proxy source files to the output directory specified in the project file. This option provides the same behavior as the **Leave Proxy Files** check box in the ProxyGen **Generate Proxies** and **Default Preferences** dialog boxes. For more information, see Specify generation settings on page 39.

The previous command syntax requires an OpenEdge project file (Release 10 or later). To convert a project file from Progress Version 9 to OpenEdge format, see Convert a project file from Progress Version 9 to OpenEdge on page 29.

Differences on UNIX and Windows platforms

Batch ProxyGen runs on UNIX in the same way as on Windows platforms, with the following limitations:

- It works only for projects that define Java proxies and Web service interfaces. You can generate .NET proxies only on Windows platforms.
- It automatically converts compatible Windows pathnames to UNIX pathnames, as follows:
  - It ignores all drive letter references in project files generated by ProxyGen on Windows platforms. As a result, all absolute path names in the project descend from root (`/`).
  - It automatically changes back slashes (`\`) to forward slashes (`/`).

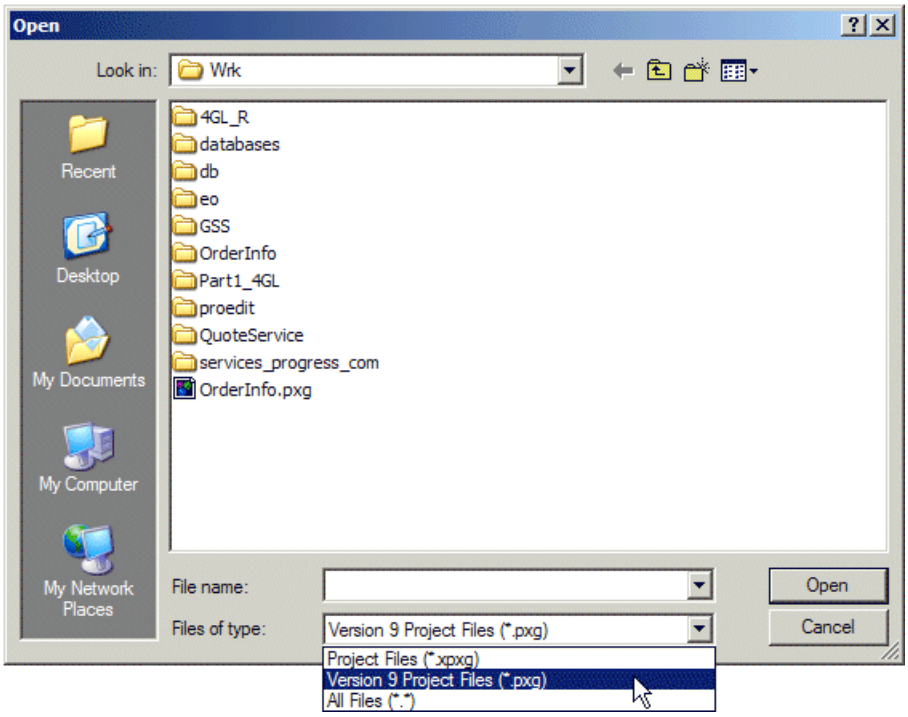
Note: When you generate project (`.xpxg`) files for use with Batch ProxyGen on UNIX systems, make sure all pathnames you specify are portable to your UNIX machine. This includes the output directory, compiler path, and `PROPATH` settings. Using any other Windows-specific naming convention results in a nonportable project file.

Convert a project file from Progress Version 9 to OpenEdge

The Open Client Toolkit for Progress Version 9 saved project files in a binary format with the file extension `.pxg`. In OpenEdge, the format of the project file is changed to XML.

Convert existing projects using ProxyGen

You can convert an existing `.pxg` file simply by opening the file in ProxyGen. To see the Progress Version 9 project files, select **Version 9 Project Files (*.pxg)** in the **Files of type** combo box, as shown in the following figure.



Once you select a Progress Version 9 project file, you are asked if you want to convert the project. Select **OK** to convert the Progress Version 9 project file to an OpenEdge project file.

The new project file is saved to the same location as the selected Version 9 project file, with the same root name, but with the `.xpxg` extension. ProxyGen then continues opening the new project file.

Define an Open Client interface using ProxyGen

This section describes how to define an Open Client interface.

Define an Open Client interface using ProxyGen

1. Define an AppObject for the Open Client interface, and optionally define one or more SubAppObjects.
2. Select ABL procedures to include in these object definitions.
3. Optionally, customize access to the selected procedures in the Open Client interface.
4. Save the Open Client interface definition to a project file.
5. Validate and generate the Open Client proxy or Web service definition.

The following sections provide information about these steps:

- Define an AppObject or SubAppObject on page 31
- Define procedures in AppObjects and SubAppObjects on page 33
- Save the Open Client interface definition in a project file on page 38
- Specify generation settings on page 39
- Validate and generate an Open Client proxy on page 48

- Object naming in ProxyGen on page 52
- Example on page 54

For specific information on ProxyGen options for defining and generating Open Client interfaces, refer to the ProxyGen online help.

Define an AppObject or SubAppObject

You must define one AppObject for a ProxyGen project. You also can define one or more SubAppObjects, depending on how you choose to organize the available functionality. As you specify AppObject and SubAppObject names and select ABL procedures to define the Open Client interface, ProxyGen applies a set of rules to convert and validate the various object and method names you specify. For more information on these naming rules, see Object naming in ProxyGen on page 52 .

Create AppObject and SubAppObject entries

To create a new AppObject definition, choose **File** > **New** from the ProxyGen menu bar or the **New** button from the ProxyGen toolbar. ProxyGen closes any open project and creates a new project for the new Open Client interface definition, adding an unnamed AppObject entry at the root of the navigation tree view. ProxyGen also displays the tab folders for this AppObject with the entry selected in the tree view.

To create a SubAppObject, choose **SubAppObject** > **New** from the menu bar. ProxyGen adds an unnamed SubAppObject entry to the navigation tree view and selects it, displaying the tab folders to define the SubAppObject.

The tab folders for defining both the AppObject and any SubAppObjects are identical; however, the definitions you enter apply only to the entry selected in the tree view. Also, as you select an entry in the TreeView, the field values in the AppObject and Procedures tab folders change to the settings for the selected AppObject or SubAppObject.

Specify AppObject and SubAppObject definitions

As part of defining an AppObject or SubAppObject, you must specify the following information in the AppObject tab folder:

- The object name (**Name** field). A hyphen (-) is not allowed in the name of an AppObject or SubAppObject. This name must be unique among AppObjects and SubAppObjects in a project. If you attempt to move off the tab folder and the name is not unique, ProxyGen displays an error message.
- The `PROPATH` (**Propath Components** list box) is set to the locations of application server procedures for this Open Client interface object. (See Define procedures in AppObjects and SubAppObjects on page 33.)

Optionally, you can specify:

- Proxy object descriptive information (**Description** edit box).
- Whether or not to enable the client to use an object to access the ABL `Unknown value (?)` as a `null` in parameters and return values (**Enable Unknowns for Parameter and Return Values** check box). This check box applies to the entire AppObject or SubAppObject. For more information, see Handle the Unknown value (?) on page 32.

Note: If you are using the Open Client interface object to access a remote SmartDataObject, you must enable access to the ABL `Unknown value (?)` for the entire object definition. Check this box and make sure the boxes in all ProcObject and method definitions remain checked.

- How to map temp-table parameters for Java proxies using the **Map Temp-Table Parameters to SQL ResultSets for Java** check box. This check box applies to the entire AppObject or SubAppObject. A Java Open Client can map temp-tables as SQL `ResultSet` objects or as OpenEdge `ProDataGraph` objects. For more information, see Map temp-table parameters in Java Open Clients on page 32.
- Whether or not to include before-image data for ProDataSets in an OpenEdge Web service using the **Write ProDataSet Before-Image in XML** check box. This check box applies to the entire AppObject or SubAppObject. For more information, see Add ProDataSet before-image data in Web services on page 33.

Handle the Unknown value (?)

You can specify how an Open Client handles the ABL `Unknown value (?)` at all levels of the interface, including:

- At a global level for the AppObject or SubAppObject
- For each procedure or user-defined function
- For individual parameters of a procedure or user-defined function

The settings at each level specify whether to enable access to the `Unknown value (?)` in parameters and return values by allowing the client to access these values as derived objects, which in .NET and Java Open Client applications can pass the ABL `Unknown value (?)` as a `null`. Otherwise, if available in the language, a .NET or Java application must use an intrinsic type, which cannot accept a `null` value and therefore cannot represent the ABL `Unknown value (?)` in the native Open Client environment. For more information on the intrinsic types available in .NET or Java, see the reference documentation for your client development language.

Map temp-table parameters in Java Open Clients

You can specify how a Java Open Client maps temp-table parameters at two levels of the interface:

- At a global level for the AppObject or SubAppObject
- For each procedure or user-defined function

The settings at each level specify if temp-table parameters are mapped to Java SQL `ResultSet` or OpenEdge `ProDataGraph` objects. In projects created with the current release of ProxyGen, all settings indicate that temp-table parameters are mapped by default to `ProDataGraph` objects. In projects created and maintained in releases prior to OpenEdge Release 10.1A, temp-table parameters are always mapped to SQL `ResultSet` objects. If you open one of these earlier projects in the current release of ProxyGen, all settings indicate that any temp-table parameters are mapped to SQL `ResultSet` objects.

However, in any new or existing project that you open in the current release of ProxyGen, if you add a procedure or user-defined function that passes at least one ProDataSet (`DATASET` or `DATASET-HANDLE`) parameter, and at least one temp-table (`TABLE` or `TABLE-HANDLE`) parameter, the temp-table parameter settings for that procedure or user-defined function indicate that temp-table parameters can only be passed as `ProDataGraph` objects. This is because the mechanism for accessing temp-table parameters as SQL `ResultSet` objects is incompatible with the mechanism (`ProDataGraph` objects) for accessing ProDataSet parameters.

For more information on mapping temp-table parameters to Java SQL `ResultSet` or OpenEdge `ProDataGraph` objects, see *OpenEdge Development: Java Open Clients*.

Add ProDataSet before-image data in Web services

You can specify whether a SOAP Web service includes before-image data for ProDataSet parameters at all levels of the interface, including:

- At a global level for the AppObject or SubAppObject
- For each procedure or user-defined function
- For individual ProDataSet parameters of a procedure or user-defined function

By default, the Web service passes only the current data in the ProDataSet. Selecting the **Write ProDataSet Before-Image in XML** check box causes the OpenEdge Web service to include the before-image data for the ProDataSet parameter in the SOAP message.

Specify application server procedure files

You must specify any application server procedure files you want to include using the **Procedure** menu on the menu bar, and if they are non-persistent or persistent procedures (ProcObjects). These procedures are listed on the **Procedures** tab folder. For more information on specifying application server procedure files, see Define procedures in AppObjects and SubAppObjects on page 33.

Define procedures in AppObjects and SubAppObjects

Once you have defined the AppObjects and SubAppObjects in ProxyGen, you can tailor the procedure files present in these objects.

To specify ABL procedures for an AppObject or SubAppObject definition:

1. Set the **Propath Components** list box, to allow ProxyGen to locate the available r-code files. See Set PROPATH components on page 34.
2. Add the r-code files for this object from the set of files that are available under the directories specified as **Propath Components**. Add each procedure according to if it is executed non-persistently or persistently.
3. Optionally, customize each procedure entry to configure the corresponding method (if non-persistent) or ProcObject (if persistent, single-run, or singleton). See Customize method and ProcObject definitions (optional) on page 35.
4. Optionally, only for ProcObjects, customize the method definitions for all specified internal procedures and user-defined functions. See Customize method definitions in a ProcObject (optional) on page 37.

The following sections provide information about:

- Set PROPATH components on page 34
- Add and delete procedure files on page 34
- Change the procedure execution mode (type) on page 35
- Customize procedure access on page 35
- Customize method and ProcObject definitions (optional) on page 35
- Customize method definitions in a ProcObject (optional) on page 37
- Change the PROPATH setting for existing procedures on page 38

Set PROPATH components

To access the procedures for AppObject and SubAppObject definitions, you must first specify, in the **Propath Components** list box, the directories that contain the r-code for these procedures. Typically, these directories correspond to a portion of the application server PROPATH. You can add, edit, or remove a PROPATH component using the **New**, **Edit**, and **Delete** buttons next to the **Propath Components** list box on the AppObject tab folder.

You can only add ABL procedures to an Open Client interface object that ProxyGen can locate using the **Propath Components** setting. The PROPATH itself might be different between the development and the deployment machines, but the relative path for the r-code files below a specified PROPATH directory must remain unchanged. Thus, if you move the r-code files to a separate development machine for access by ProxyGen, make sure you maintain the directory structure used for relative path names under the **Propath Components** setting.

To include procedures located in a Procedure Library, you must have the individual r-code files on disk in the same directory structure as in the Procedure Library. You can then pick the individual r-code files from the disk. For deployment, you can package the files into the Procedure Library, as long as you also include the Procedure Library on the application server PROPATH. You also can use dot (`.`) or some other relative path as a PROPATH component.

Add and delete procedure files

You can add any ABL procedure file to an AppObject or SubAppObject definition, provided the compiled (r-code) version of the file is available to ProxyGen through the **Propath Components** setting. For information on compiling ABL procedures, see *OpenEdge Getting Started: ABL Essentials*.

To add procedure files to your object definition, choose **Procedure** > **Add** and select the procedure type. A dialog box will appear, which will allow you to:

- Select r-code files under a pathname selected from your **Propath Components** setting (**Propath Selection** combo box).
- Specify whether to use the relative pathname of selected procedures as part of the corresponding method name in the Open Client interface using the **Use Path in Name** check box. This helps to automatically generate unique method names.
- Request that any directories you select be searched recursively using the **Perform Recursive Add** check box.
- Add the selected files to the **Procedures** tab folder for the current AppObject or SubAppObject definition using the **Add** button. The procedures you add depend on what files and directories you selected in the **Folders** tree and the **Folder Contents** list.

Note: ProxyGen will not allow you to set a procedure as single-run or singleton if the procedure has parameters in the main block..

For each non-persistent procedure, ProxyGen generates a method in the Open Client interface. For each persistent, single-run, or singleton procedure, ProxyGen generates a ProcObject class factory method in the Open Client interface, along with a ProcObject definition that encapsulates its internal procedures and user-defined functions. For information on class factory methods, see *.NET Open Clients* or *Java Open Clients*.

After selecting and adding the respective procedure files, any non-persistent procedures appear in the **Non-persistent procedures** list, and any persistent, single-run, and singleton procedures appear in the **Procedure Objects** list of the **Procedures** tab folder on the main window.

To delete procedures from either list in the Procedures tab folder:

1. Select one or more procedures in one list.
2. Press the **DELETE** key or choose **Procedure** > **Remove** from the menu bar.

Change the procedure execution mode (type)

You can change the execution type of any procedure files already added to an Open Client interface object definition, using the **Procedures** tab.

To change the execution mode of a procedure:

1. Select the **Procedures** tab.
2. Do one of the following:
   a) Select one or more procedures in either list. Choose the **Procedure** > **Change Type** from the menu bar.
   b) Right-click on a procedure name and select the procedure type from the **Change To** menu option.
   c) If the procedure you want to change is in the **Procedure Object** window, click on the **Type** next to the procedure and use the drop-down menu to select **Persistent**, **SingleRun**, or **Singleton**. (You can use this option to change a ProcObject to a different type of ProcObject, but if you want to change a ProcObject to a non-persistent procedure, you must use the **Procedure** > **Change Type** menu item.)

Note: When changing a ProcObject to a non-persistent procedure, you lose any customizations to internal procedures and user-defined functions that you applied using the **Customize** function.

Customize procedure access

To manage client access to remote procedures, ProxyGen supports three levels of access control for remote procedures:

1. The application server developer can define internal procedures and user-defined functions as `PRIVATE` in the ABL. ProxyGen does not display private procedures and user-defined functions for use in a ProcObject definition.
2. You can further select what public internal procedures and user-defined functions you want ProxyGen to include in a ProcObject, using the **Customize Persistent Procedure** dialog box. For more information, see the Customize method and ProcObject definitions (optional) on page 35.
3. The application server developer can use the business logic on the application server to further restrict what procedures can be run remotely, using the ABL `EXPORT` statement. This statement restricts access at run time, without affecting the Open Client interface definition. For information on how application server applications can use the `EXPORT` statement to control remote procedure access at run time, see *OpenEdge Application Server: Developing Application Server Applications*.

Customize method and ProcObject definitions (optional)

For finer control, you can customize method and ProcObject definitions.

To customize method and ProcObject definitions for procedure files added according to your application requirements:

1. From the **Procedures** tab folder, select a procedure file in one of the lists.
2. Double-click the selection or choose **Procedure** > **Customize** from the menu bar.

This displays a dialog box that provides:

- A **Procedure** tab folder to modify the name and description of the associated method or ProcObject. In this folder, you can:
  - Rename the method or ProcObject (**Method Name** or **ProcObj Name** field)
  - Enter descriptive information for the method or ProcObject (**Description** field)
- A **Parameters** tab folder to modify the behavior of method or ProcObject parameters. The tab has two sections listing the procedure's parameters, one for customizing use of the Unknown value (?) and one for customizing use of ProDataSet before-image data. The dialog box populates each list with the parameters in the procedure. Deselecting either the **Use AppObject Unknown Setting** or **Use AppObject Before-Image Setting** check boxes enables the corresponding parameter list. You can then apply the setting to each specific parameter by selecting the check box beside it in the list.

  Note: Not all settings apply to all the types of objects that ProxyGen can generate from the AppObject description. ProxyGen ignores any settings that do not apply to the client proxy that you specify at generation.

  The **Enable Unknowns** list controls whether one or more method or ProcObject parameters enable passing of the ABL Unknown value (?) using objects to represent the Unknown value (?) as a null. If not checked, and if available in the language, the parameter uses an intrinsic type that cannot represent the ABL Unknown value (?). The **Write XML Before-Images** list controls whether a SOAP Web service includes before-image data for a ProDataSet parameter. Selecting the check box beside a ProDataSet includes the before-image data for that parameter.
- The Return **ABL RETURN-VALUE** check box specifies whether the procedure returns a string containing the value of the ABL `RETURN-VALUE`.
- The Map Temp-Table Parameters to SQL ResultSets for Java check box specifies whether to map temp-table parameters to Java SQL `ResultSet` or OpenEdge `ProDataGraph` objects for Java Open Clients. This check box is disabled and unchecked if one of the following is true:
  - The **Use AppObject Temp-Table Setting** check box is checked.
  - The procedure passes at least one ProDataSet parameter (with or without temp-table parameters). Any temp-table parameters must map to OpenEdge `ProDataGraph` objects.
  - The procedure passes no ProDataSet or temp-table parameters.
- The **Use AppObject Unknown Setting** check box specifies whether to use the AppObject or SubAppObject setting to indicate if all parameters and return values enable passing of the ABL Unknown value (?) using objects to represent the Unknown value (?) as a null. For more information, see Specify AppObject and SubAppObject definitions on page 31.
- The Use AppObject Before-Image Setting check box (checked by default) specifies whether to include ProDataSet before-image data in a SOAP Web service. For more information, see Specify AppObject and SubAppObject definitions on page 31
- A check box (**Use AppObject Temp-Table Setting**), checked by default, that specifies whether or not to use the AppObject or SubAppObject setting to indicate if the procedure uses SQL ResultSet or OpenEdge ProDatGraph objects to map temp-table parameters for Java Open Clients. For more information, see Specify AppObject and SubAppObject definitions on page 31. This check box is disabled and unchecked if one of the following is true:
  - The procedure passes at least one ProDataSet parameter (with or without temp-table parameters). Any temp-table parameters must map to OpenEdge `ProDataGraph` objects

- The procedure passes no ProDataSet or temp-table parameters.

The **Customize Persistent Procedure** dialog box also has an **Internal Procs** tab folder to customize methods (internal procedures and user-defined functions) of the specified ProcObject. In the Internal Procs tab folder for ProcObjects (**Customize Persistent Procedure** dialog box), you can:

- Set check boxes to indicate whether one or more internal procedures or user-defined functions are included as methods of the ProcObject
- Customize the method definition for a selected internal procedure or user-defined function included in the ProcObject (**Customize** button)

Customize method definitions in a ProcObject (optional)

For finer control, you can customize method definitions in a ProcObject.

To customize the definitions for methods included in a ProcObject according to your application requirements:

1. From the **Customize Persistent Procedure** dialog box, select an included internal procedure or user-defined function from the **Internal Procedures/User-Defined Functions** list.
2. Choose the **Customize** button.

The **Customize Internal Procedure** dialog box or the **Customize User Defined Function** dialog box appears, depending on the selection. This dialog box provides:

- A **Procedure** tab folder to modify the name and description of the method associated with the selected internal procedure or user-defined function.
- A **Parameters** tab folder to modify the behavior of any method parameters.
- A check box for internal procedures (**Return ABL RETURN-VALUE**) that specifies whether the procedure returns a string containing the value of the ABL `RETURN-VALUE`.
- A check box for user-defined functions (**Enable Unknown Return Value**) that specifies whether to enable clients to access the function return value as an object in order to return the ABL `Unknown value (?)` as a `null`. If not checked, and if available in the language, the client must use an intrinsic type for the return value that cannot represent the ABL `Unknown value (?)`.
- A check box (**Map Temp-Table Parameters to SQL ResultSets for Java**) that specifies whether to map temp-table parameters to Java SQL `ResultSet` or OpenEdge `ProDataGraph` objects for Java Open Clients. This check box is disabled and unchecked if one of the following is true:
  - The **Use AppObject Temp-Table Setting** check box is checked.
  - The procedure passes at least one ProDataSet parameter (with or without temp-table parameters). Any temp-table parameters must map to OpenEdge `ProDataGraph` objects.
  - The procedure passes no ProDataSet or temp-table parameters.
- A check box (**Use AppObject Unknown Setting**) that specifies whether or not to use the AppObject or SubAppObject setting to indicate if all method parameters and return values enable passing of the ABL `Unknown value (?)` using objects to represent the `Unknown value (?)` as a `null`.
- A check box (**Use AppObject Before-Image Setting**), checked by default, that specifies whether to include ProDataSet before-image data in a SOAP Web service. For more information, see Specify AppObject and SubAppObject definitions on page 31.
- A check box (**Use AppObject Temp-Table Setting**), checked by default, that specifies whether to use the AppObject or SubAppObject setting to indicate if the procedure uses SQL `ResultSet` or OpenEdge `ProDatGraph` objects to map temp-table parameters for Java Open Clients. For more information, see the

Specify AppObject and SubAppObject definitions on page 31. This check box is disabled and unchecked if one of the following is true:

- The procedure passes at least one ProDataSet parameter (with or without temp-table parameters). Any temp-table parameters must map to OpenEdge `ProDataGraph` objects.
- The procedure passes no ProDataSet or temp-table parameters.

In the **Procedure** tab folder, you can:

- Rename the method (**Method Name** field)
- Enter descriptive information for the method (**Description** edit box)

In the **Parameters** tab folder, you can modify the behavior of method parameters. The tab has two sections listing the parameters, one for customizing use of the `Unknown value (?)` and one for customizing use of ProDataSet before-image data. The dialog box populates each list with the parameters in that method. Deselecting either the **Use AppObject Unknown Setting** or **Use AppObject Before-Image Setting** check boxes enables the corresponding parameter list. You can then apply the setting to each specific parameter as needed by selecting the check box beside it in the list.

Note: Not all settings apply to all the types of objects that ProxyGen can generate from the AppObject description. ProxyGen ignores any settings that do not apply to the client proxy that you specify at generation.

The **Enable Unknowns** list controls whether one or more method or ProcObject parameters enable passing of the ABL `Unknown value (?)` using objects to represent the `Unknown value (?)` as a `null`. If not checked, and if available in the language, the parameter uses an intrinsic type that cannot represent the ABL `Unknown value (?)`. The **Write XML Before-Images** list controls whether a SOAP Web service includes before-image data for a ProDataSet parameter. Selecting the check box beside a ProDataSet includes the before-image data for that parameter.

Change the PROPATH setting for existing procedures

You might find it necessary to change the `PROPATH` setting for existing procedures of an AppObject or SubAppObject. You can accomplish this by deleting and then adding these procedures again within your project; however, this results in the loss of any customizations. To accomplish this without losing data, the Open Client Toolkit provides the **Change Propath** tool.

You can use this tool on an open project file by selecting **Tools** > **Change Propath**. The **Change Propath** dialog box allows you to replace selected `PROPATH` components used by non-persistent and persistent procedures in the client interface definition. For instructions on using this dialog box, refer to the online help.

Save the Open Client interface definition in a project file

At any point during the definition of an Open Client interface, you can create or save the definition in a project (`.xpxg`) file. This file includes the list of all selected application server procedures and any customizations you applied to those procedures, for each Open Client interface object definition.

Set the project work directory

By default, ProxyGen saves all project files in a *work directory*. You specify the default path for this work directory during OpenEdge installation of the Open Client Toolkit. The OpenEdge installation saves the path for the ProxyGen working directory in the **ProxyGen** icon. You can modify the work directory in the properties of the **ProxyGen** icon. For more information, see Use ProxyGen execution options on page 26.

ProxyGen allows you to specify a separate directory to save generated Open Client proxies, a SOAP Web service definition, and log file. For details, see Specify generation settings on page 27.

Save the project file

To save the Open Client interface to a project file, choose **File** > **Save** from the menu bar. ProxyGen opens a **Save As** dialog box in the current work directory. You can save the project file in this directory or select a different work directory to save the file. Each time during the same ProxyGen session that you save a project file, the **Save As** dialog box opens to the last directory you selected. ProxyGen automatically saves the project file to the current work directory each time you validate or generate.

Specify generation settings

Among the settings you can specify to complete an Open Client interface are a set of preferences for generating Open Client proxies or SOAP Web service definitions. This information becomes part of the Open Client interface definition. You can set and change these generation settings at different points in a ProxyGen session:

- To set the default preferences for generating a new Open Client interface, use the **Default Preferences** dialog box (choose **Options**® Default **Preferences** from the menu bar). The settings you choose become the default settings for all Open Client interfaces once you click the **OK** button and save the project file.

  These options specify such information as an application service name for the application server connection, compiler options for Open Client proxy generation, and WSDL style and use for Web service generation. By default, ProxyGen initializes these settings from, and saves them to, the *`OpenEdge-install-directory`*`\properties\proxygen.xml` file.

- To set the preferences for generating the current Open Client interface, use the **Generate Proxies** dialog box (choose **File**® **Generate** or the **Generate** toolbar button). The settings you choose remain set for the current Open Client interface once you generate the interface with these settings and save the project file. For more information on generating an Open Client interface, see Validate and generate an Open Client proxy on page 48.

You can specify a local preferences file in the properties of the ProxyGen icon. See the Use ProxyGen execution options on page 26.

Dialog box layout

The **Default Preferences** and the **Generate Proxies** dialog boxes both display an identical layout that includes a fixed region and a dynamic region. The fixed region displays general settings and buttons that apply to all Open Client types. The dynamic region, which is contained in a border box labeled **Client Details**, displays different settings and buttons depending on the Open Client type that you select.

You can use the **Client Type** drop-down combo-box located at the top of the dialog box to select the Open Client type whose settings you want to display in **Client Details**. Clicking on this combo-box displays the following options:

- **.NET** — See .NET client details on page 41 for more information.
- **Java** — Selecting this options displays the settings described in the Java client details on page 43 for more information.
- **Sonic Native Invocation** — See Sonic Native Invocation client details on page 44 for more information.
- **Sonic Web Services** — See Sonic Web Service client details on page 45 for more information.
- **Web Service** — See Web Service client details on page 46 for more information.

General settings for proxy generation

The general settings displayed in the fixed region for all Open Client types include:

- **AppService Name** can contain the name of an application service. To use the name of the AppObject in the **AppService Name** field, select the **Use Default** check box.
- **Output Directory** — This field specifies an output directory where ProxyGen places the generated output files, including the proxy executable files (`.class` and `.dll`), .NET runtime assemblies (`.dll`, for .NET only), SOAP Web service definition files (`.wsm` and `.wsdl`), and the activity log (*`AppObject`*`.log`). If you select the **Leave Proxy Files** check box, ProxyGen also saves proxy source files in this directory.

  You must enter a valid directory path or ProxyGen displays a message prompting for the path before it generates any output files.

  To make the output directory more portable, you can use dot (`.`) or some other relative path as your **Output Directory** setting for all application development.

Note: ProxyGen does not clean the output directory before generation. If there are existing files in the directory with the same name as a generated file, ProxyGen overwrites the existing files. Similarly, ProxyGen ignores any obsolete client proxy or log files, which you can remove manually from the directory.

- **Author** and **Version** — Use these free-form fields to enter any comments you want to include in the Open Client interface definition. These are included as "Author ? and "Version ? source comments in the Open Client proxies and SOAP Web service definition.
- **Return ABL RETURN-VALUE on Connect** — Use this option to control whether or not you want values from the Connect procedure `RETURN` statement passed back to the client by way of the Web Services Adapter or the OpenEdge Adapter for Sonic ESB. When selected, it provides the ability to return a value from the application server connect procedure to the Web Service or Sonic ESB client. The Return Connect Procedure `RETURN-VALUE` option is only available when you use a managed session model.

  This option does not affect the normal process of returning SOAP faults when a connection attempt fails from a client using a Web Services Adapter or the OpenEdge Adapter for Sonic ESB. That is, when the Connect procedure fails and returns a string value, then that string value is sent to the client in the `<faultstring>` element of a SOAP fault. If the Connect procedure does not return a string, then the normal SOAP fault is returned to the client.

Note: This option is ignored when generating proxies for .NET and Java.

- **Session Model** — You must set this field according to the operating mode of all application servers that support this service interface:
  - **Free** — The application servers are configured for the state-free operating mode. This is the default.
  - **Managed** — The application servers are configured for the stateless, state-aware, or state-reset operating mode.

Note: This option is ignored when generating proxies for .NET and Java. For these two Open Clients, the session model is determined by settings in the client code. For more information, see *OpenEdge Development: .NET Open Clients* and *OpenEdge Development: Java Open Clients*.

- **Verbose Logging** — Select this check box to have detailed information output to the log file.
- **Leave Proxy Files** — If you select this check box, ProxyGen writes the generated proxy source files (`.cs` files for .NET and `.java` files for Java clients), as well as the proxy executable files, to the directory specified in the **Output Directory** field.

.NET client details

The **.NET** client type generates executable proxies in the form of .NET assembly (`.dll`) files.

Client Details for the **.NET** client type displays the following generation settings:

- **Namespaces** — Namespace settings to use both for the root in the generated .NET proxy and for DataSet and DataTable classes generated in the proxy.
  - **General** — Enter the namespace to use as the root for generating the classes in a .NET proxy.
  - **DataSet** — By default, ProxyGen creates the namepace of the strongly typed DataSet or DataTable class by appending `"StrongTypedNS"` to the project's **General** namespace. To specify a different namespace for the DataSet or DataTable class, uncheck the **Use Default** option.
- **Compiler** — These radio buttons allow you to specify one of the following compilers to use for generating a .NET proxy:
  - **Default csc** — ProxyGen uses the latest Microsoft `csc` C# compiler on the system.
  - **Custom** — You specify an alternate C# compiler for proxy generation.
- **Advanced** — Click this button to access the **Advanced .NET Options** dialog box which allows you to set compiler and XSD settings for the .NET client proxy. For a detailed description, see Advanced .NET Options dialog box on page 43.
- **Add Namespace to Output Directory** — Select this option if you want ProxyGen to create the project's DLL file in a subdirectory of the output directory. This option allows you to turn off the default behavior of creating the project's DLL file in subdirectories using the namespace, and allows you to put it directly in the output directory.

  For example, if the output directory is `C:/wrk` and the **Namespace** is `ABC.ClientNS`, then ProxyGen creates the DLL file in `C:/wrk/ABC/ClientNS`. Uncheck this option if you want ProxyGen to create the project's DLL file directly into the output directory (for example, `C:/wrk`).
- **Assembly Info** — General information about a .NET assembly is controlled through this set of attributes: **Title**, **Version**, **Desc** (Description), **Company**, **Runtime**, **Product**, **Delay Sign**, and **Key File**. You can

change the attribute values to modify the information associated with an assembly. This information is critical to uniquely identify the proxy.

Version has the following syntax:

Table 6: Syntax

```
major.minor.build.revision
```

You can specify all **Version** values or accept the defaults for build number and revision by using the * (asterisk) wildcard. For example:

- **1.2.3.4** (specify all values)
- **1.2.*.4** (accept default value for *build*)
- **1.2.3.*** (accept default value for *revision*)
- **1.2.*** (accept default values for *build* and *revision*)

The **Runtime** combo-box specifies which type of Open Client runtime assemblies (DLLs) you want to use for your Open Client project:

- **Digitally Signed** — Indicates that the DLLs are digitally signed to identify the author, but are not strongly named for precise DLL version matching. This is the default setting.
- **Strongnamed Signed** — Indicates that the DLLs are both digitally signed to identify the author and strongly named for precise DLL version matching.
- **Strongnamed** — Indicates that the DLLs are not digitally signed to identify the author and will be strongly named for precise DLL version matching. This configuration is not generally recommended but may be necessary in some special cases.

For more information on these .NET runtime assembly options, see the information on .NET client interface generation in Open Client interface generation on page 49.

If **Delay Sign** is checked with one of the **Strongnamed** options, ProxyGen uses delayed signing along with the public key file to strongname the proxy assembly. This means the strongname signing process must be completed outside of ProxyGen on a system where the key pair file is located (normally a secure system). For more information, see the documentation on Microsoft's Strong Name tool (`sn.exe`). By default, this is unchecked.

Key File is the public key file used for delayed strongname signing. This field is enabled only if **Delay Sign** is checked.

- **Unknown Support** — These radio buttons allow you to specify how you want the .NET proxy to support the .NET null value for parameters and return values that can have the ABL `Unknown value (?)`:
  - **Nullable Types** — Use .NET nullable value types. These are built-in .NET structures that support the null value in addition to the values of a corresponding .NET primitive value type (such as `System.Int32` or `System.Decimal`). Each parameter or return type with an ABL primitive data type (such as `INTEGER` or `DECIMAL`) that can have the `Unknown value (?)` is therefore mapped to a corresponding nullable value type.
  - **Holder Classes** — Use OpenEdge holder classes. These are OpenEdge-defined .NET classes that correspond to .NET primitive value types. Each holder class is defined with properties that allow the .NET null value to be set and tested in addition to allowing the values of the corresponding .NET value type to be set and returned. Each parameter or return type with an ABL primitive data type that can have the `Unknown value (?)` is therefore mapped to a corresponding holder class.

Note: For ProxyGen projects created in OpenEdge releases without support for nullable types (releases prior to OpenEdge 11.0), the project defaults this setting to **Holder Classes**. For ProxyGen projects created in OpenEdge releases that support nullable types, the project defaults this setting to **Nullable Types**. For more information on using nullable value types and holder classes in .NET proxies, see *OpenEdge Development: .NET Open Clients*.

Advanced .NET Options dialog box

You access the **Advanced .NET Options** dialog box by clicking the **Advanced** button from the **Client Details** for the **.NET** client type. This dialog box has the following fields for setting .NET proxy options:

- **Compiler Command** — If **Default csc** is selected in the **Compiler** field, the **Compiler Command** field is read-only and displays the appropriate compiler command.

  If **Custom** is selected in the **Compiler** field, you can modify the **Compiler Command** setting as appropriate for your compiler choice. You must specify the full path or a path that is relative to the system path for the compiler executable.

- **Compiler Options** — You can set compiler options for the C# compiler.

- **XSD Generator** — This is the location of the `xsd.exe` tool, which is used in the generation of a strongly typed ADO.NET `DataSet` or `DataTable`. If **Default csc** is selected in the **Compiler** field, the **XSD Generator** field is read-only and displays the appropriate command.

  If **Custom** is selected in the **Compiler** field, you can modify the **XSD Generator** setting as appropriate. You must specify the full path or a path that is relative to the system path for the XSD executable.

Java client details

The **Java** client type generates executable proxies in the form of Java `.class` files.

Client Details for the **Java** client type displays the following generation settings:

- **Package** — Enter the package name to use as the root directory for generating the classes in the Java proxy.

- **Compiler** — Enter the compiler to use for generating a Java proxy. This will initialize to the `javac` for the `jdk` you specified during the installation. Use the **...** button to specify a an alternate Java compiler for proxy generation.

  To make your code more portable, consider using your `PATH` environment variable for finding the Java compiler only by executable name. Doing this also allows you to install the JDK for each platform in vendor-specified default directories, without causing your application code to fail when moving from one platform to another.

- **Compiler Command** — If **Default javac** is selected in the **Compiler** field, the **Compiler Command** field is read-only and displays the appropriate compiler command.

  If **Custom** is selected in the **Compiler** field, you can modify the **Compiler Command** setting as appropriate for your compiler choice. You must specify the full path or a path that is relative to the system path.

- **Classpath Switch** — This field specifies the compiler option used to specify the classpath. If **Default javac** is selected in the **Compiler** field, this field is read-only and displays the appropriate classpath switch. If **Custom** is selected in the **Compiler** field, you can set this value appropriately for your compiler.

- **Classpath** — This field specifies the actual classpath. If **Default javac** is selected in the **Compiler** field, this field is read-only and displays the appropriate classpath. If **Custom** is selected in the **Compiler** field, you can set this value appropriately for your compiler.

Note: The **Classpath** includes the ABL class libraries `o4glrt.jar` and `messages.jar`. The initially displayed value in the **Classpath** field is the one required for proxy generation. Never change this portion of the **Classpath**. Instead, append any additional values to the end.

- **Compiler Options** — You can set other compiler options (in addition to the **Classpath**) for any selected compiler.

Note: Do not set the **Classpath** in **Compiler Options**, because doing so overrides the default values required for proxy generation.

Symbolic references to Java Open Client directory paths

For directory paths, symbolic references (indicated by <>) are often used in place of actual values. These references are resolved during proxy generation.

When ProxyGen is run on UNIX, all directories and filenames are converted automatically to UNIX format. ProxyGen uses symbolic references in several generation options, to allow the greatest portability for the project file, particularly when the file is used on UNIX.

The symbolic references and their expanded values are shown in the following table:

Table 7: Symbolic references to Java Open Client directory paths

`<Default classes.zip>`	Windows: `%DLC%\jdk\lib\classes.zip` UNIX: `$JDKCP`
`<Install Dir>`	Windows: `%DLC%` UNIX: `$DLC`
`<Proxy Dir>`	The **Output Directory** field value described in General settings for proxy generation on page 40. Progress Software Corporation recommends you always leave this symbolic reference and change the **Output Directory** value in the general settings to alter the proxy output directory.

Sonic Native Invocation client details

The **Sonic Native Invocation** client type generates a script file (`.esboe` file) for each specified procedure or function that can be imported directly into Sonic Workbench for deployment to a Sonic ESB container. When a Sonic ESB process is created using the Sonic Native Invocation methodology, ABL procedures are called directly through an OpenAPI call to an OpenEdge application server. For more information on the Sonic Native Invocation methodology, see *OpenEdge Development: Messaging and ESB*.

Note: You can also change the deployment settings for this client type during deployment to a Sonic ESB container.

Client Details for the **Sonic Native Invocation** client type displays the following generation settings:

- **Invocation File Deployment Options** — The following settings to generate the `.esboe` files for deployment:

- **Save to Output Directory** — Specify this option to have your `.esboe` files written to the directory specified in the **Output Directory** field described in the General settings for proxy generation on page 40.
- **Deploy to Directory Service** — Specify this option to have your `.esboe` files written to your Sonic Directory Service. If specified, you must also specify the absolute path of your Sonic Directory Service in **Resource Dir.** For more information on Sonic Directory Services, see your Sonic documentation.
- **Overwrite Files** — If you specified **Deploy to Directory Service**, select **Overwrite Files** to replace any existing files in your Sonic Directory Service.
- **Create Deployment Archive** — Specify this option to have your `.esboe` files written into a deployment archive. If specified, you must also specify an archive (`.xar`) file in **Archive Name**.

- **Advanced** — Click this button to access the **Advanced Sonic ESB Options** dialog box, which allows you to specify settings for the Sonic Domain Manager where the `.esboe` files are to be deployed. For a detailed description, see Advanced Sonic ESB Options dialog box on page 46.

Sonic Web Service client details

The **Sonic Web Service** client type generates deployment information for a Sonic Web Service definition.

Note: You can also change the deployment settings for this client type during deployment to a Sonic ESB container.

Client Details for the **Sonic Web Service** client type displays the following deployment settings:

- **Namespace** — In this field, enter a value for the Sonic Web Service target namespace. This value is required for all SOAP-based applications. It must be unique for the service. The default is `urn:tempuri-org`.

Note: The **Namespace** field for the **Sonic Web Service** client type and the **Namespace** field for the **Web Service** client type are linked. If you change the **Namespace** setting for one client type, it is automatically updated for the other client type.

- **WSDL Style** — The encoding style of the binding. Select one of the following options:
  - **Doc/Literal** — Specify this option if you want the entire message to be defined as a single entity and you want the generating of the messages represented literally using XML schema standards. This is the default as it is the recommended encoding style.
  - **RPC/Literal** — Specify this option if you want each parameter to be defined as a separate entity and you want the formatting of the messages represented literally using XML schema standards.
  - **RPC/Encoded** — Specify this option if you want each parameter to be defined as a separate entity and the formatting of the messages to be encoded according to a set of encoding rules. This option is **not** recommended. Although this option is allowed, the message cannot be validated since not all of the SOAP body is defined by XML schema.
- **Entry Endpoint** — The Sonic ESB endpoint name for invoking the service. The endpoint is an abstract name with an underlying query or topic. If the endpoint does not exist, ProxyGen creates it. You must create the underlying queue in Sonic for this to work properly. The default is `.Entry` appended to the AppObject name (for example, `MyAppObj.Entry`). Uncheck the **Use Default** option to specify a different name.
- **Reject Endpoint** — The Sonic ESB endpoint name for rejected messages sent to the service. The endpoint is an abstract name with an underlying query or topic. If the endpoint does not exist, ProxyGen creates it.

You must create the underlying queue in Sonic for this to work properly. The default is `.Reject` appended to the AppObject name (for example, `MyAppObj.Reject`). Uncheck the **Use Default** option to specify a different name.

- **Fault Endpoint** — The Sonic ESB endpoint name for faults returned from the service. The endpoint is an abstract name with an underlying query or topic. If the endpoint does not exist, ProxyGen creates it. You must create the underlying queue in Sonic for this to work properly. The default is `.Fault` appended to the AppObject name (for example, `MyAppObj.Fault`). Uncheck the **Use Default** option to specify a different name.
- **Container Name** — The name of the Sonic ESB container in which to deploy the service. ProxyGen does not create the container if it does not exist. The default is the Sonic ESB container created when you install the OpenEdge Adapter for Sonic ESB. However, there is no initial value for this field.
- **URL** — The URL of the AppServer to connect the Sonic ESB container to. The default is the `esbbroker1` session-free AppServer installed with OpenEdge by default. The initial value of this field is `AppServerDC://localhost:3091`.
- **Sonic ESB Service** — These options specify the location of files for the ESB:
  - **Filename** — The base filename for the WSM and WSDL files. This is the same name as the AppObject.
  - **Overwrite Files** — Check this option if you want to overwrite existing WSM and WSDL files in SonicFS. If you leave this option unchecked and the files already exist in SonicFS, ProxyGen does not deploy the Sonic Web Service and generates a message stating that deployment did not proceed.
  - **Resource Dir** — The path in SonicFS where you want to store the WSM and WSDL files.
- **Advanced** — Click this button to access the **Advanced Sonic ESB Options** dialog box which allows you to specify settings for the Sonic Domain Manager where the Sonic Web service is to be deployed. For a detailed description, see the Advanced Sonic ESB Options dialog box on page 46.

Advanced Sonic ESB Options dialog box

You access the **Advanced Sonic ESB Options** dialog box by clicking the **Advanced** button from the **Client Details** for either the **Sonic Native Invocation** client type or the **Sonic Web Service** client type. This dialog box has the following fields for setting **Sonic Domain Manager** options:

- **URL** — The network location of the Sonic Domain Manager. The initial value is `tcp://localhost:2506`.
- **Domain** — The name of the Sonic domain. The initial value is `Domain1`.
- **User** — The name of the user account used to log on with. The initial value is `Administrator`.
- **Password** — The password for the specified **User** account. The initial value is blank. The characters you type in this field are masked with asterisks (*).
- **Retype Password** — Retype the **Password** in this field. This field is disabled until you type text in the **Password** field. The value typed in this field must match the value typed in the **Password** field. The characters you type in this field are masked with asterisks (*).

Web Service client details

The **Web Service** client type generates deployment information for an OpenEdge Web service definition.

Note: You can also change the deployment settings for this client type during deployment to a OpenEdge Web Services Adapter (WSA).

Client Details for the **Web Service** client type displays the following deployment settings:

- **Namespace** — In this field, enter a value for the Web service target namespace. This value is typically used to version the Web service in any WSDL that you generate. You can specify any value that meets the following requirements:
  - Uniquely identifies the Web service and its elements in the WSDL file
  - Uniquely identifies the Web service for the WSA instance where it is deployed
  - Meets the requirements of an XML namespace value (a valid URN or URL)

  For example:

```
urn:www-progress-com:OrderSvc:OrderInfo
```

For more information on using the target namespace for Web service versioning, see the sections on versioning Web services in *OpenEdge Application Server: Administration*.

Note: The **Namespace** field for the **Web Service** client type and the **Namespace** field for the **Sonic Web Service** client type are linked. If you change the **Namespace** setting for one client type, it is automatically updated for the other client type.

- **Generate Test WSDL** — If you check this box, ProxyGen generates a WSDL file with the specified *style*/*use* (See the **WSDL Style** radio buttons).
- **URL for WSA** — In this field, enter the URL of the Web services adapter instance to which this Web service is deployed. This value can be modified at deployment time. For more information on this URL, see the sections on Web services administration in *OpenEdge Development: Web Services*.
- **SOAP Action** — These radio buttons specify how you want the string value for the SOAP Action Header to appear in the WSDL file for each SOAP message: **Blank** or **User-defined** (any value you enter, including blank).

  If you select both **User-defined** and the **Append Object Name** check box, the name of the Web service object is appended to the value you enter, to form the SOAP Action Header value.

- **WSDL Style** — These radio buttons specify the format for SOAP messages supported by the Web service. The selected *style/use* must match the SOAP format supported by the client applications you intend to support.

  Style refers to the communications style. Some applications communicate by exchanging messages as remote procedure calls (RPCs), while others communicate by exchanging messages as XML documents.

  Use refers to the syntax or format of each message itself. The formatting of the messages can be encoded according to a set of encoding rules or represented literally using XML schema standards.

  Select one of the following style/use combinations:

  - **Document/Literal** — This option is the default because the entire SOAP message is defined in a single entity.
  - **RPC/Literal** — This option defines the entire SOAP message in a single entity and formats the message literally using XML schema standards.

- **RPC/Encoded** — This option is not recommended. Although this option is legal for Web Services, it is not WS-I compliant, because the message cannot be validated since not all of the SOAP body is defined by XML schema.

For more information on the WSDL style and use attributes and how they affect Web service client communications, see *OpenEdge Development: Web Services*.

Note: To deploy the Web service to the Web Services Adapter (WSA) with multiple SOAP encoding styles, you must either generate the Web service with a unique Namespace value (different version) for each SOAP encoding style or you must deploy the Web service with each SOAP encoding style to a different WSA instance. For more information on the WSA and how to deploy Web services to it, see the OpenEdge Management or OpenEdge Explorer online help.

Caution: >A single deployed Web service can support a single SOAP encoding style. To support multiple encoding styles, change the output directory (field described in the **Output Directory**) to avoid overwriting any WSDL file already generated for this Web service.

- **Suffix** — These options specify the content and formatting of the generated WSM file and optional test WSDL file (See the **Generate Test WSDL** check box):
  - **PortType/Binding** — In this field, enter a custom suffix for the PortType and Binding name in the WSDL file. If you check the **Use Default** box, this field is disabled and displays the default value for the suffix (`Obj`).
  - **Web Service** — In this field, enter a custom suffix for the Service name in the WSDL file. If you check the **Use Default** box, this field is disabled and displays the default value for the suffix (`Service`).

Validate and generate an Open Client proxy

ProxyGen allows you to validate and generate an Open Client interface in one step. Once a project file is saved, you can generate the Open Client proxy, Web, or Sonic ESB service definition by choosing **File® Generate** from the menu bar. You can also choose the **Generate** button from the toolbar. Both options display the **Generate Proxies** dialog box, from which you can then choose the **OK** button to generate the Open Client interface.

During generation, the list of procedures is validated against the r-code files available on disk. If the r-code for any procedure cannot be found, ProxyGen automatically removes that procedure from the Open Client interface definition and records this action in an activity log file (*`AppObject`*`.log`). ProxyGen picks up all prototype changes and reconciles these changes (such as new or removed parameters) with existing customizations. ProxyGen records all such significant reconciliations in the activity log. This continues until all the procedures are validated. For the available r-code files, the Open Client proxy or SOAP Web service definition is then generated with any specified customizations.

Caution: (Java only) On Windows platforms, a compiler error might result if you generate a Java proxy, change the case of an AppObject, SubAppObject, or ProcObject name, and then regenerate the proxy without first deleting the old proxy files.

The following sections summarize the actions and rules that govern:

- Open Client interface validation on page 49
- Open Client interface generation on page 49

Open Client interface validation

The Open Client interface definition is reconciled against the actual r-code on the system, because the r-code can change independently of the saved Open Client interface definition. This process throws away obsolete information and picks up new information, while preserving customizations that are still valid with regard to these changes. This process generates an activity log (`.log`) file that records any significant validation events. This file is created in the output directory specified in the generation preferences.

When validation occurs

Open Client interface validation occurs when you do any of the following:

- Choose **File**® **Generate** from the **ProxyGen** menu bar or the **Generate** button from the toolbar.
- Run Batch ProxyGen on the saved project file.
- Customize a method or ProcObject definition (procedure or persistent procedure). Validation occurs only for the selected procedure.

How validation works

Open Client interface validation does the following:

- Verifies the selected procedure files exist on disk, and if it does not, removes the procedure from the Open Client interface definition
- Compares any customizations made for procedures against the actual r-code and synchronizes them according to the following rules:
  - If a parameter for a procedure in the project file matches a parameter in the r-code by name and data type, any customizations for the parameter are maintained. Otherwise the parameter is assumed to be new, and the default setting is used to determine whether the client can use the parameter to pass an `Unknown value (?)`. If you remove the parameter, it is removed from the interface.
  - If an internal procedure or user-defined function in the project file matches an internal procedure or user-defined function in the r-code by name and type (where type is whether it is a procedure or function), any customizations are maintained. Otherwise the internal procedure or user-defined function is assumed to be new, and it is assigned all default definition settings (method name, description, and so on).
- Validates the method and parameter names and verifies all method names are unique within each Open Client interface object, and that parameter names are unique for each method

---

Note: While you can override method names in ProxyGen, you cannot override parameter names.

- Verifies all object names are unique within the Open Client interface
- Logs discrepancies, errors, and reconciliations to the activity log

Open Client interface generation

Once the Open Client interface definition is validated and updated as appropriate, the Open Client proxy or SOAP Web service definition can be generated. The generated output for a:

- .NET proxy includes a single assembly (`.dll`) file and (optionally) `.cs` files containing the classes
- Java proxy includes a set of `.class` and (optionally) `.java` files
- Sonic Native Invocation proxy includes a set of script (`.esboe`) files, one for each directly invoked application server procedure
- Sonic Web Service definition includes a `.wsm` file.
- Web Service definition includes a `.wsm` file and (optionally) a `.wsdl` file for testing.

In all cases, the activity log records any significant generation events and errors.

For more information on the generated:

- Proxy for a .NET client, see Generate .NET client interfaces on page 50
- Proxy for a Java client, see *OpenEdge Development: Java Open Clients*
- Script (`.esboe`) files for a Sonic Native Invocation client, see *OpenEdge Development: Messaging and ESB*
- Sonic Web Service definition, see *OpenEdge Development: Messaging and ESB*
- Web Service definition, see *OpenEdge Development: Web Services*

Generate .NET client interfaces

The .NET proxy generation options also allow you to choose from three alternative Open Client Runtime assemblies to use and deploy with your .NET proxy, depending on the level of security that your executable code demands.

The basic difference between these assemblies for Open Client deployment is that the most secure runtime assemblies require more of your executable code to be rebuilt when you make a change to the Open Client application:

- *Strong-named and signed* Open Client Runtime assemblies

  This is the most secure option because Progress Software Corporation builds the OpenEdge .NET assemblies both strong-named and digitally signed. Strong-naming an assembly guarantees that the assembly contents have had no unauthorized changes (without a spoof being detected) and digitally signing the assembly identifies who created it (Progress Software Corporation). Using strong-named Open Client Runtime assemblies supports the following options and requirements:

  - You can strong-name both the proxy and the .NET application, and you can digitally sign the proxy and the .NET application. If you want, you can also have the OpenEdge .NET assemblies loaded into the .NET Global Assembly Cache.
  - When you are ready to deploy updated OpenEdge .NET assemblies, you must regenerate the proxy and rebuild the Open Client application, then redeploy all the files.
  - After installing any service pack or other fix, you must regenerate the proxy and rebuild the Open Client application, then redeploy all the files.

- *Signed-only* Open Client Runtime assemblies

  This is the less secure option because Progress Software Corporation builds the OpenEdge .NET assemblies digitally signed, but not strong-named. Digitally signing an assembly identifies who created it (Progress Software Corporation), but does not guarantee that the assembly contents have had no unauthorized changes (allowing a spoofed assembly to go undetected). Using signed-only Open Client Runtime assemblies supports the following options and requirements:

- This is the default option, allowing the least disruption during redeployment, in exchange for lessened security.
- You **cannot** strong-name either the proxy or the .NET application; but you **can** digitally sign both the proxy and the .NET application. You also **cannot** have the OpenEdge .NET assemblies loaded into the .NET Global Assembly Cache.
- When you are ready to deploy updated OpenEdge .NET assemblies, you can simply redeploy the updated .NET assemblies without redeploying the proxy or Open Client application.
- After installing any service pack or temporary fix, you can simply redeploy the updated .NET assemblies without deploying the proxy or Open Client application.

- *Strong-named* (but unsigned) Open Client Runtime assemblies

  This is a less common type of assembly required by some applications. The assemblies are strong-named but are not digitally signed.

  Using strong-named-unsigned Open Client Runtime assemblies supports the following options and requirements:

  - You can strong-name both the proxy and the .NET application. You can also have the OpenEdge .NET assemblies loaded into the .NET Global Assembly Cache.
  - When you are ready to deploy updated OpenEdge .NET assemblies, you must regenerate the proxy and rebuild the Open Client application, then redeploy all the files.
  - After installing any service pack or other fix, you must regenerate the proxy and rebuild the Open Client application, then redeploy all the files.

Note: The strong naming process allows applications to precisely match the version of a DLL needed by a caller. In environments where several versions of an assembly might be needed to support multiple products or versions of the same product, strong naming is vital. Because strong naming uses several criteria to precisely identify the version of a DLL, its use makes unauthorized code changes very difficult to implement. Thus, strong naming can also be seen as a security measure. Strong naming entails the overhead of more careful code management. For this reason, you should carefully research the pros and cons of strong naming before enabling this option. You can choose to use strongly named assemblies with or without digital signing. Strong naming with digital signing is the most common choice. Assemblies with strong naming but no digital signing are a less common use case, but might be required in some environments.

Depending on the option you choose (see .NET client details on page 41), .NET proxy generation causes the corresponding Open Client Runtime assemblies to be copied to the output directory for ease of deployment (See the **Output Directory** field described in General settings for proxy generation on page 40).

ProxyGen generation performs the following actions, writing any output to the output directory that you specify in the generate preferences (**Output Directory** in the general settings):

- Saves the Open Client interface definition to the project file
- Performs Open Client interface validation
- Generates a Java proxy, .NET proxy, and/or SOAP Web service definition, as specified
- For .NET proxies, provides a copy of the selected Open Client Runtime assemblies

For more information on the generated proxy for a .NET client and working with the different runtime assemblies, see *OpenEdge Development: .NET Open Clients*.

Object naming in ProxyGen

Open Client interface generation requires the mapping of ABL identifier names to names appropriate for the specific Open Client interface. This includes restricting the use of certain characters and reserved words.

ProxyGen determines the default names of Open Client interface methods and parameters from the r-code filename and its contents. These names conform to the Open Client interface naming conventions and automatic conversions described in the sections that follow.

Note: You can override the default names for Open Client interface methods by customizing the procedure in ProxyGen.

Open Client interface naming conventions

ProxyGen uses the following default naming conventions for Open Client interface objects and methods:

- **AppObject, SubAppObject, and ProcObject class names** — You must specify the name for each AppObject and SubAppObject class. Unless you override it, each ProcObject class name is the r-code filename of the mapped persistent procedure, matching the case and spelling without the extension. ProxyGen also performs any automatic name conversions. See Automatic name conversions on page 53.

  Each object must be uniquely named.

  For Java and .NET proxies, these object names are used as the actual class names.

  For SOAP Web services, these object names are used according to the conventions and programming language requirements of the Web service client toolkit that builds the client proxies from the WSDL file. For more information on how Web service client platforms use these object names, see the sections on sample Web service client platforms in *OpenEdge Development: Web Services*.

- **Methods in an AppObject or SubAppObject that execute an external, non-persistent application server procedure** — The method name is the procedure filename without the extension. Method names are case sensitive. ProxyGen also performs any automatic name conversions. See Automatic name conversions on page 53.

- **Methods in a ProcObject that execute an internal procedure or user-defined function defined in the procedure** — The method name matches the corresponding ABL internal procedure and user-defined function names found in the procedure file. Method names are case sensitive. ProxyGen also performs any automatic name conversions. See Automatic name conversions on page 53.

- **Class factory methods in an AppObject that create a SubAppObject** — The method name depends on the type of Open Client, where *`SubAppObject`* is the SubAppObject name:

  - **Java proxy** — `createAO_SubAppObject`
  - **.NET proxy and Web service definition** — `CreateAO_`*`SubAppObject`*

- **Class factory methods in an AppObject or SubAppObject that create a ProcObject** — The method name depends on the type of Open Client, where *`ProcObject`* is the ProcObject name:

  - **Java proxy** — `createPO_`*`ProcObject`*
  - **.NET proxy and Web service definition** — `CreatePO_`*`ProcObject`*

Note: On Windows platforms, the case of the r-code filename is affected by the name entered during ABL compilation. For example, suppose you execute the following `COMPILE` statement in the Procedure Editor: `COMPILE aBCdEf.p SAVE.` Then the generated r-code filename is `aBCdEf.r`, even though `Abcdef.p` might be the actual procedure filename. Thus, the generated method name is `aBCdEf` or `CreatePO_aBCdEf` for a persistent procedure.

Automatic name conversions

During name generation, ProxyGen does some automatic conversion that capitalizes the character following each restricted character, and it removes all restricted characters from the name. The restricted characters are:

- Pound (#)
- Ampersand (&)
- Percent (%)
- Dash (-)
- Dot (.)
- Dollar sign ($) — .NET only

Note: Once you edit a method name, ProxyGen no longer does this automatic conversion. If your edit includes a restricted character, ProxyGen displays an error, and you must edit the name again.

Naming side effects and restrictions

Automatic name conversions can cause two methods or parameters in the same namespace to acquire the same name, even when the original names in a procedure file are different. This also can occur if you have two procedure files with the same name under different directories. You can override method names in ProxyGen to handle method name conflicts, but you must change any conflicting parameter names directly in the procedure files, because you cannot customize parameter names in ProxyGen.

When ProxyGen generates an Open Client proxy or SOAP Web service definition, a separate `.java` file for Java proxies, a separate class (`.cs`) file for .NET proxies, and a separate `<binding>` element for Web service definitions is created for each AppObject, SubAppObject, and ProcObject. To avoid naming conflicts for all these object representations, ProxyGen does not allow you to generate if any two objects in the Open Client interface have the same name.

For example, if you have two persistent procedures with the same name but in different directories, you must customize the ProcObject name for at least one of these procedures, to prevent a filename conflict. You can do this by customizing one of the procedures and editing the name, or by including the relative path in the name. Similarly, you must ensure all your AppObject, SubAppObject, and ProcObject names are unique within a single Open Client interface.

Example

An OpenEdge developer at company ABCPets designs and develops an inventory system for pet shops, using the AppBuilder (`.w` files). These source files are in the directory `c:\ABCPets\Source\Inventory`. The files are compiled, and the r-code files are placed in `c:\ABCPets\Bin\Inventory`. ABCPets wants to package this functionality for Java clients. You run ProxyGen and supply the following information:

- **AppObject Name** — `Inventory`
- **Propath Components** — `c:\ABCPets\Bin`
- **Enable Unknowns for Parameter and Return Values** — Selected
- **Non-persistent Procedures** — `Inventory\InvAdd.r`
- **Procedure Objects** — `Inventory\p-Cust.r`, set as type persistent, with the following internal procedures:
  - `AddInventory()`, which takes two `INTEGER` parameters
  - `ValidateInventory()`
  - `UpdateInternals()`, which is defined in `Inventory\p-Cust.r` as `PRIVATE`

You customize the method definition for `AddInventory()` to disable the use of objects for passing the ABL `Unknown value (?)` in either parameter. You also explicitly exclude `ValidateInventory()`. Note that the internal procedure `UpdateInternals()` is defined with the `PRIVATE` keyword and does not appear in ProxyGen as a procedure to include or customize.

When you choose the Generate button, you supply the following information:

- **AppService** — `com.ABCPets.Inventory`
- **Package** — `com.ABCPets`
- **Output Dir** — `c:\Proxies`

Proxy generation places the Java proxy (the `.class` files) into `c:\Proxies\com\ABCPets`, and creates the activity log as `c:\Proxies\Inventory.log`. The generated Java files include:

- `Inventory.java`
- `Inventory.class`
- `InventoryImpl.class`
- `pCust.java` (note the removal of the dash)
- `pCust.class`
- `pCustImpl.class`

You then change `p-Cust.w` but do not change the signature of `AddInventory()` (`AddInventory(INTEGER, INTEGER)`). Add new internal procedures to the persistent procedure `p-Cust.w`. These changes only require you to run ProxyGen to regenerate the proxy as long as you want them included. New non-`PRIVATE` internal procedures are added as methods automatically, with default definitions. To exclude them, you must run ProxyGen and customize the persistent procedure definition. Also, any changes in, or the removal of, `ValidateInventory()` do not cause validation errors because it is excluded from the proxy definition and it is not validated.

Run Batch ProxyGen as part of a regular build cycle, so any similar changes in ABL are picked up automatically.

Later in the development cycle, add a new procedure file for inventory, called `Orders.w`. This procedure file is saved in `c:\ABCPets\Source\Inventory,` and the r-code file is placed in `c:\ABCPets\Bin\Inventory`. Then run ProxyGen before the scheduled build and add this procedure file to the `Inventory` AppObject.

You make another change to the `AddInventory()` procedure, which now takes a third parameter (CHARACTER). Because this is a new parameter, it automatically gets the default (AppObject) setting to enable access to ABL `Unknown value (?)`. If you want to disable access to the ABL `Unknown value (?)` by eliminating support for using objects as parameters, and instead using equivalent intrinsic types (where available in the language), you must run ProxyGen before the scheduled build and update the customizations for this procedure.

4

Programming Concepts

All Open Clients use equivalent mechanisms to access an application server application service. In general, accessing an application server using an Open Client interface follows this process:

1. Connect to an application server that supports the required application service, by instantiating an AppObject (and calling connect for SOAP Web services).
2. Instantiate any other proxy objects whose methods you want to call.
3. Execute proxy methods that run remote procedures and functions on the application server.
4. Disconnect from the application server when it is no longer needed.

For more detailed information about specific clients, see *Java Open Clients*, *.NET Open Clients*, and *Web Services*.

For details, see the following topics:

- Connect to an application server
- Get user input for run-time connection information
- Proxy and Web service object methods
- Pass parameters
- Access Open Client run-time properties
- Handle errors in an Open Client application
- Access an application server directly without proxies

Connect to an application server

To specify an application server connection for a client application using an Open Client interface, you provide the same connection information as for an ABL (Advanced Business Language) client application. How you provide the connection information depends on your type of client:

- For Java and .NET clients, you pass the connection information as a set of parameters to the `Connection` object constructor, and the `Connection` object is passed as a parameter to the AppObject constructor. Alternately, you can pass the connection information directly as a set of parameters to the AppObject's constructor. You specify the session model to use for the connection (session-managed or session-free) using run-time property settings that you make prior to the connection. For more information on run-time property settings, see Access Open Client run-time properties on page 61. For more information on the `Connection` object for Java Open Clients, see *OpenEdge Development: Java Open Clients*. For more information on the `Connection` object for .NET Open Clients, see *OpenEdge Development: .NET Open Clients*.

Note: As a run-time property setting, you must specify the session model according to the requirements of each application service you access, as specified by the application server operating mode configuration.

- For session-managed SOAP Web services, you call a `Connect_`*`AppObject`* method after instantiating the AppObject. For session-free Web services, you simply instantiate the AppObject, with no need to invoke a connect method. For details, see *OpenEdge Development: Web Services*.

The Java Open Client also provides a client principal class, `com.progress.open4gl.javaproxy.ClientPrincipal`. A `ClientPrincipal` object is associated with the `Connection` object and provides a means to store and transmit user credentials. The `ClientPrincipal` class supports the following use cases:

- Creating an unsealed `ClientPrincipal` object containing a user's credentials and sending this object to an application server for a direct login.
- Creating a sealed `ClientPrincipal` object (verified by an external authentication system) and sending this object to an application server for a single sign-on.

For more information on the use of the client principal class, see *OpenEdge Development: Java Open Clients*. The client principal class is not currently supported on .NET Open Clients.

Get user input for run-time connection information

.NET and Java Open Clients provide the capability to set connection parameters and certain other parameters at run time (run-time properties). They can be specified when launching the client, or the user can be prompted to supply them. For more information, see the Access Open Client run-time properties on page 61.

Your Open Client application might need to obtain information from the end user if:

- You need to provide the names and locations of root digital certificates, for an HTTPS connection, to authenticate the identity of a Web server, or for a TLS intranet connection, to authenticate the identity of an application server. Every Web server that hosts the AIA and supports the HTTPS protocol and every TLS-enabled application server must have a server digital certificate of a Certificate Authority (CA). Authentication is done using a root digital certificate located on the client machine. Because the location of this certificate depends on the location of the certificate store where the root digital certificate was installed, the location of the root digital certificate must be available at run time.

For .NET Open Clients, this is not an issue, because .NET maintains a common certificate store for all .NET applications that require access to root digital certificates. However, for Java Open Clients, there is no standard installation location for certificate stores. Therefore, a Java Open Client application might have to determine the location of the certificate store that it requires at run time and pass it to the Open Client Runtime using the `RunTimeProperties` class.

- You need to access the Internet using a proxy server. If your application must go through an HTTP proxy server to connect to the AIA or WSA, the application needs to know the proxy host address and port and any required user ID and password. A .NET or Java application passes this information to the Open Client Runtime, using the methods and properties of the `RuntimeProperties` class. A SOAP Web service client has its own mechanism for handling Internet access using a proxy server, depending on the client platform.
- An HTTP or HTTPS URL requires a userid and password for authentication to the Web server hosting the AIA or WSA.
- An application server requires a userid and password to access its application service.
- The connection URL format differs between the AppServer/AppServerDC and AppServerS/AppServerDCS protocols and the HTTP and HTTPS protocols.
- You communicate with more than one AIA, in which case, you might need to use:
  - Different URL file paths
  - More than one application service

The application might automatically prompt the user for the information when it starts, or it might choose to dynamically handle the Connection Error status returns specific to failed server authentications and retry the connection after getting the information from the end user.

For more information on setting Open Client run-time properties in .NET, see *OpenEdge Development: Web Services*, and in Java, see *OpenEdge Development: Java Open Clients*.

For more information on handling user input for Web service clients, see *OpenEdge Development: Web Services*.

Proxy and Web service object methods

Proxy objects provide methods that allow you to manage proxy objects, create other proxy objects, and access application server procedures. ProxyGen generates proxy objects for Java and .NET. The actual proxies for a SOAP Web service typically are generated by Web service client tools that use the Web service WSDL file.

Proxy objects support four types of methods you can call from an Open Client application:

- **Connection methods** — Establishes a connection to an application server.

  For more information about connecting to an application server, see Connect to an application server on page 58. For more information about establishing a connection for a specific type of Open Client, see *Java Open Clients*, *.NET Open Clients*, or *Web Services*.

- **Common methods** — Manages a proxy object and its application server connection. The common methods available depend on the client type. For details, see *Java Open Clients*, *.NET Open Clients*, or *Web Services*.
- **Class factory methods** — Allows you to create SubAppObjects and ProcObjects. Executing a SubAppObject class factory method does not access the application server. If a ProcObject encapsulates a persistent procedure, the class factory method executes the corresponding persistent procedure on the application server. If a ProcObject encapsulates a single-run or singleton procedure, the class factory method does

not execute the corresponding procedure. The single-run or singleton procedure will only be instantiated on the application server when one of its internal procedures or user-defined functions is called.

ProxyGen generates class factory methods using standard naming conventions for each type of Open Client. For more information, see Generate Proxies on page 21.

For Java Open Clients, a built-in class factory method also allows you to create a predefined ProcObject, `SDOResultSet`, for a given SmartDataObject procedure file. For more information, see *OpenEdge Development: Java Open Clients*.

- **Remote ABL methods** — Invokes non-persistent procedures, internal procedures, and user-defined functions on the application server. For more information, see Remote ABL methods on page 60.

The following table shows which types of methods are available for each type of proxy object.

Table 8: Types of methods available for different proxy objects

Type of method	AppObject	SbAppObject	ProcObject
Connection	X	–	–
Common	X	X	X
Class factory: • SubAppObject class factory • ProcObject class factory • SDOResultSet class factory	X X X	- X X	- - -
Remote ABL	X	X	X

Remote ABL methods

ProxyGen maps each ABL non-persistent procedure, internal procedure, and user-defined function that you expose on the application server, to a remote ABL method. When invoked, each remote ABL method executes the procedure (or user-defined function) on the application server, passes any parameter values, and makes any return values available to the Open Client.

Note: If an ABL persistent, single-run, or singleton procedure has a super procedure, the proxy also contains remote ABL methods for the super procedure, as long as the remote procedure declares the super procedure prototypes. For more information, see the information on programming for Open Client applications in *Developing Application Server Applications*.

Identifying generated methods

ProxyGen generates method names using automatic conversions and conventions. For more information, see Generate Proxies on page 21. In addition, you can customize the method names in ProxyGen. Thus, each method name might not match the name of the corresponding ABL procedure or function. As a reference for method names in your proxy, use the generated Java source files for Java clients, the generated assembly for .NET clients, and the WSDL for Web services.

Pass parameters

ProxyGen maps ABL data types to equivalent data types in the Open Client for ABL `INPUT`, `OUTPUT`, and `INPUT-OUTPUT` parameters. For more information on data type mapping for your type of Open Client, see *Java Open Clients*, *.NET Open Clients*, and *Web Services*.

Handle return values

For any remote ABL procedure, the corresponding proxy method can be customized to return the ABL `RETURN-VALUE`; otherwise, the method returns void. Alternatively, after calling the method, the client can access the current value of the ABL `RETURN-VALUE` function by calling the common method for Java and .NET Open Clients shown in the following table:

Java method	`_getProcReturnString`
.NET method	`_GetProcReturnString`

This method returns the current value of `RETURN-VALUE` as set by the `RETURN` statement most recently executed on the application server for this connection. It returns a string, which can be null, so the client should check for an ABL `Unknown value (?)`. AppObjects, SubAppObjects, and ProcObjects all support this method.

The `RETURN-VALUE` function is set in ABL using the `RETURN string` or the `RETURN ERROR` *`string`* statement. For more information on these statements, see *ABL Reference*, and for how they function in an application server application, see *Developing Application Server Applications*.

For any remote user-defined function, ProxyGen defines the corresponding method to return a value that is compatible with the ABL data type returned by the user-defined function.

Note: For a complete discussion or error handling techniques, including the use of the new structured error handling features, see *OpenEdge Development: ABL Error Handling*.

Pass parameters

The Open Client passes parameters to the application server for proxy object methods. The mechanism used to send and receive parameters depends on the type of Open Client.

For information about how a specific Open Client type passes parameters, see *Java Open Clients*, *.NET Open Clients*, and *Web Services*.

Access Open Client run-time properties

Several properties are provided to govern behavior across the entire application. These properties do not affect any particular object, but they affect the behavior of all objects created by the application. These are referred to as run-time properties and include tracing, thread control, Proxy server support, and digital certificate management.

For more information on how to access these properties in your type of Open Client, see *Java Open Clients*, *.NET Open Clients*, and *Web Services*.

Handle errors in an Open Client application

Calling a remote ABL method can generate three different types of errors:

- Client-side errors on page 62
- Server-side errors before remote ABL method execution on page 62
- Server-side errors during remote ABL execution on page 62

A *client-side error* is an error that occurs on the client machine before the client sends a particular request to the application server. A *server-side error* is an error that occurs on the application server machine, either before the application server runs a remote procedure or during procedure execution.

No matter what type of error ABL returns, the error message and number are made available to the client application, in the form of an exception for Open Clients and a SOAP exception for Web service clients.

For more information on how errors are returned for your type of Open Client, see *Java Open Clients*, *.NET Open Clients*, and *Web Services*.

The following sections describe these types of errors.

Client-side errors

For Java and .NET, the proxy returns all client-side errors to the client. An example of a client-side error is an input parameter whose schema does not match that of the corresponding ABL temp-table. The proxy throws an exception for any such error. in a manner appropriate for your type of client. SOAP Web service clients may detect client-side errors based on the WSDL; however, the Open Client has no influence on this behavior.

Because the error occurs before the request is sent to the application server, the request is cancelled with no effect on the application server. In general, client-side errors have no effect on the application server to which the client is connected. However, the application server does try to detect the abnormal termination of a client connection and execute the configured disconnect sequence when it happens.

Server-side errors before remote ABL method execution

ABL returns all server-side errors to the client that occur before any remote procedure is executed. An example of this type of server-side error is where the client tries to execute a remote procedure that does not exist. Generally, both the client and application server can resume normal execution.

ABL also returns all network connection and authentication errors; for example, if an HTTP connection cannot be made, authentication cannot be established with the application server, or the application server is not configured properly.

Server-side errors during remote ABL execution

ABL returns server-side errors that occur during execution of a remote procedure. There are three types of remote procedure conditions an application server can return to the client:

- **ERROR condition** — A server-side ERROR condition is sent to the client only when a remote procedure or function executes a RETURN ERROR statement. All other server-side ERROR conditions (such as failure to connect to a database) are not raised beyond the scope of the ABL procedure where they occur; therefore, such ERROR conditions do not propagate to the client. Instead, the application server writes error messages for these conditions to the server log file.
- **STOP condition** — Any unhandled STOP condition raised on the application server is sent to the client. This type of server-side error can occur, for example, when the called remote procedure calls another procedure that cannot be found.
- **QUIT condition** — Any unhandled QUIT condition raised on the application server is sent to the client. The server-side ABL application can use a QUIT statement to raise a QUIT condition because the client exceeded a resource, did something it should not do, or for any other reason it chooses.

Access an application server directly without proxies

For .NET and Java Open Clients, OpenEdge provides a means for you to access application services directly on an application server without the need to generate proxies using ProxyGen. You can do this using an API (*OpenAPI*) provided with each Open Client. This Open Client OpenAPI provides a set of classes for you to access application services using general access methods.

Because these classes know nothing about the application service they are accessing, you must be familiar with all the requirements for accessing the external procedures, persistent procedures, single-run procedures, singleton procedures, internal procedures, and user-defined functions provided by the application service. With this information, you must set up the procedure and user-defined function signatures according to application service requirements in your client code at run time.

You can use the principles of programming with proxy AppObjects and ProcObjects when accessing an application server using the OpenAPI. However, if you have a basic understanding of how ABL provides application services on the application server and you know the programming requirements for the application service you are accessing, you have enough information to use the OpenAPI to access that application service.

For information on the application server, see *Developing Application Server Applications*.

The following table lists the common classes provided for use with both the .NET and Java versions of the OpenAPI. OpenEdge provides these classes to each type of Open Client in different assemblies or packages appropriate for the Open Client.

Table 9: OpenAPI Classes

Class	Description
OpenAppObject	For accessing remote external ABL procedures in an application service using the OpenAPI. It supports general functionality similar to the application-specific functionality of a proxy AppObject.
OpenProcObject	For instantiating and accessing a remote ABL persistent, single-run, or singleton procedure using the OpenAPI, including its internal procedures and user-defined functions. It supports general functionality similar to the application-specific functionality of a proxy ProcObject.
ParamArray	Array for passing parameters to an ABL procedure or user-defined function.

Class	Description
`ParamArrayMode`	Constants for specifying the mode of an ABL procedure or user-defined function parameter (`INPUT`, `INPUT-OUTPUT`, or `OUTPUT`).
`Parameter`	Constants for specifying the ABL data type of a procedure or user-defined function parameter, return type, or temp-table field.

The functionality of these common classes is essentially identical for each type of Open Client, with minor variations appropriate to the client platform and implementation. In addition to the common OpenAPI classes, each Open Client has a set of classes to define ProDataSet and temp-table parameters for use with the OpenAPI that are unique to the client platform and its requirements for mapping these data objects.

The OpenAPI is documented completely for each Open Client type. For more information, see:

- *.NET Open Clients* for the .NET OpenAPI
- *Java Open Clients* for the Java OpenAPI

Index

.NET Open Client
    Runtime assemblies 50
.NET proxy
    generating 48
    generation rules 50
    validation rules 49

A

ABL internal procedures 37
ABL procedures
    customizing access 35
ABL user-defined functions 37
Activity log file 48
Adding procedure files 34
AppObjects
    defining in ProxyGen 31
    Open Client interface 22
    specifying ABL procedures and functions 33
AppServer
    connecting, general programming 58
    interface access to procedures 24
Authentication
    server digital certificates 58

C

Class factory methods
    using in an Open Client 59
Client principal class 58
Client-side errors
    client-proxy interactions 62
    definition 62
Common methods 59
Connection methods 59
Connections
    user input 58
Creating
    AppObjects 31
    ProcObjects 33
    SubAppObjects 31
Customizing access to ABL procedures 35

D

Default Preferences dialog box
    .NET client settings 41
    general settings 40
    Java client settings 43
    layout 39
    Sonic Native Invocation definition settings 44
Default Preferences dialog box *(continued)*
    Sonic Web Service definition settings 45
    Web Service definition settings 46
Defining an interface 30
Deleting procedure files 34
Directory paths
    Java symbolic references 44
Distribution packages, Open Client Runtime 19

E

Encapsulating functionality 23
ERROR condition handling 62
Error handling
    ProxyGen validation and generation 48

G

Generate Proxies dialog box
    general settings 40
    .NET client settings 41
    Java client settings 43
    layout 39
    Sonic Native Invocation definition settings 44
    Sonic Web Service definition settings 45
    Web Service definition settings 46
Generating proxies 48
Generation
    Open Client interface 49
Generation rules 49
Generation settings
    specifying 27, 39
Getting started with ProxyGen 24

I

Internal procedures
    mapping in ProxyGen 37

J

Java proxy
    generating 48
    generation rules 49
    temp-table parameter mapping
        AppObjects and SubAppObjects 31
        external procedures 35
        internal procedures and UDFs 37

M

Methods
- class factory *59*
- common *59*
- connection *59*
- customizing for ProcObjects *37*
- customizing in ProxyGen *35*
- passing parameters *61*
- remote ABL *59–60*

N

Naming conventions
- automatic conversions *53*
- proxy object *52*
- restrictions and side effects *53*

Non-persistent procedures
- adding and deleting in ProxyGen *34*

O

Open Client
- applications
  - requirements *19*
  - specifying an AppServer connection *58*
- generating proxies *48*
- interface *22*

Open Client interface
- .NET generation *50*
- AppServer procedure access *24*
- defining *30*
- definition example *54*
- encapsulation *23*
- generating *28*
- generation *49*
- generation and validation *48*
- naming conventions *52*
- objects *22*
- saving in project file *28*
- specifying generation settings *27*
- validation *49*

Open Client Proxy Generator, See ProxyGen

Open Client Runtime distribution packages *19*

OpenAPI
- description *63*
- supporting classes *63*

P

Parameter passing in Open Clients *61*

Persistent procedures
- adding and deleting in ProxyGen *34*

Preferences
- ProxyGen *39*

Procedure files
- adding and deleting *34*
- methods *35*

Procedures
- customizing access *35*

ProcObjects
- compared to procedures *23*
- customizing *35*
- defining in ProxyGen *33*
- Open Client interface *22*

ProDataSets
- including before-image data in Web service *35*
- including before-image data in Web services *31*, *33*

Programming Open Clients
- AppServer connections *58*
- error handling *62*
- See also Error handling
- method types *59*
- OpenAPI without proxies *63*
- passing parameters *61*
- run-time properties *61*

Progress Version 9
- project file conversion *29*

Project
- file
  - conversion *29*
  - definition *27*
  - saving *38*
- work directory *39*

PROPATH
- accessing AppServer procedures *24*
- AppObjects and SubAppObjects *31*
- changing existing setting *38*

Proxies
- adding and deleting procedures *34*
- defining *30*
- generating *48*
- generation and validation *48*
- methods *59*
- naming conventions *52*
- parameter passing *61*
- run-time properties *61*

ProxyGen
- activity log *48*
- basic tasks *25*
- converting project files *29*
- creating a ProcObject *33*
- creating a SubAppObject *31*
- creating an AppObject *31*
- Default Preferences dialog box *39*
- defining interfaces *30*
- Generate Proxies dialog box *39*
- generating and validating proxies *48*
- generation settings
  - general *40*
  - Java client *43*

ProxyGen *(continued)*
generation settings *(continued)*
.NET client *41*
Sonic Native Invocation definition *44*
Sonic Web Service definition *45*
Web Service definition *46*
getting started *24*
GUI execution options *26*
main window *27*
overview *24*
preferences *39*
saving the project file *38*
starting
batch *28*
.pxg, See Project

Q

QUIT condition handling *62*

R

Remote ABL methods *60*
Return values *61*
Root digital certificates
authenticating servers *58*
Rules
proxy generation *49*
proxy validation *49*
Run-time properties *61*

S

Server digital certificates, validating *58*
Server-side errors
client-server interactions *62*
definition *62*
Servers, digital certificates *58*
Session models
specifying for Open Clients *24*
Signed-only .NET assemblies *50*
Single-run procedures
adding and deleting in ProxyGen *34*
Singleton procedures
adding and deleting in ProxyGen *34*
Specifying an AppServer connection
Open Client application *58*
Starting
Batch ProxyGen *28*
ProxyGen *26*
STOP condition handling *62*
Strong-named .NET assemblies *50*
structured error handling *61*
SubAppObjects
defining in ProxyGen *31*
SubAppObjects *(continued)*
Open Client interface *22*
specifying ABL procedures and functions *33*
Symbolic references
Java directory paths *44*
System requirements
building applications *19*
ProxyGen *19*

T

Temp-table parameters
passed in Java proxies *32*
Thread control *61*
Tracing *61*

U

UNIX
Batch ProxyGen *29*
Unknown value (?)
enabling for function return values *37*
enabling for parameters and return values of Sub/AppObjects *31*
enabling for procedure parameters
external *35*
internal *37*
example for managing *54*
passed as a parameter in proxy objects *32*
setting for procedure file methods and ProcObjects *37*
User input for connections *58*
User-defined functions
mapping in ProxyGen *37*

V

Validation
Open Client interface *49*
rules *49*
server digital certificates *58*

W

Web service interface
defining *30*
Web services
including ProDataSet before-image data *31*, *33*, *35*
methods *59*
run-time properties *61*
Windows
Batch ProxyGen *29*
Work directory
project *39*