## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>Preface–1</td>
</tr>
<tr>
<td>1. Overview</td>
<td>1–1</td>
</tr>
<tr>
<td>- Architecture</td>
<td>1–2</td>
</tr>
<tr>
<td>- Developing and accessing an Open Client interface</td>
<td>1–4</td>
</tr>
<tr>
<td>- Accessing AppServer functionality</td>
<td>1–5</td>
</tr>
<tr>
<td>- The session model</td>
<td>1–5</td>
</tr>
<tr>
<td>- The object model</td>
<td>1–7</td>
</tr>
<tr>
<td>2. Configuration</td>
<td>2–1</td>
</tr>
<tr>
<td>- Building and deploying an Open Client application</td>
<td>2–2</td>
</tr>
<tr>
<td>- Managing Open Client root digital certificates</td>
<td>2–3</td>
</tr>
<tr>
<td>- Java Open Clients</td>
<td>2–3</td>
</tr>
<tr>
<td>- .NET Open Clients</td>
<td>2–4</td>
</tr>
<tr>
<td>- Web services</td>
<td>2–4</td>
</tr>
<tr>
<td>3. Generating Proxies and Web Service Definitions</td>
<td>3–1</td>
</tr>
<tr>
<td>- Open Client interfaces</td>
<td>3–2</td>
</tr>
<tr>
<td>- Objects in an Open Client interface</td>
<td>3–2</td>
</tr>
<tr>
<td>- Session models</td>
<td>3–3</td>
</tr>
<tr>
<td>- Making procedures available to the interface</td>
<td>3–4</td>
</tr>
<tr>
<td>- Getting started with ProxyGen</td>
<td>3–5</td>
</tr>
<tr>
<td>- Versions of ProxyGen</td>
<td>3–5</td>
</tr>
<tr>
<td>- Using ProxyGen execution options</td>
<td>3–6</td>
</tr>
<tr>
<td>- Starting and using ProxyGen</td>
<td>3–7</td>
</tr>
<tr>
<td>- Editing an Open Client interface</td>
<td>3–8</td>
</tr>
<tr>
<td>- Specifying generation settings</td>
<td>3–9</td>
</tr>
<tr>
<td>- Saving Open Client interfaces in a project file</td>
<td>3–9</td>
</tr>
<tr>
<td>- Generating an Open Client proxy or Web service definition</td>
<td>3–9</td>
</tr>
<tr>
<td>- Running Batch ProxyGen</td>
<td>3–10</td>
</tr>
<tr>
<td>- Converting a project file from Progress Version 9 to OpenEdge</td>
<td>3–11</td>
</tr>
<tr>
<td>- Converting a preferences file from Progress Version 9 to OpenEdge</td>
<td>3–12</td>
</tr>
<tr>
<td>- Defining an Open Client interface using ProxyGen</td>
<td>3–13</td>
</tr>
</tbody>
</table>
Defining an AppObject or SubAppObject ........................................ 3–14
  Creating AppObject and SubAppObject entries ............................. 3–14
  Specifying AppObject and SubAppObject definitions ....................... 3–14
Defining procedures in AppObjects and SubAppObjects ...................... 3–17
  Setting PROPATH components ........................................... 3–17
  Adding and deleting procedure files ..................................... 3–18
  Changing the procedure execution mode (type) ............................ 3–19
  Customizing procedure access ........................................... 3–19
  Customizing method and ProcObject definitions (optional) ............... 3–20
  Customizing method definitions in a ProcObject (optional) ............. 3–22
  Changing the PROPATH setting for existing procedures .................... 3–23
Saving the Open Client interface definition in a project file ................. 3–24
  Setting the project work directory ..................................... 3–24
  Saving the project file .................................................. 3–24
Specifying generation settings .................................................. 3–25
  General tab ............................................................... 3–25
  .NET tab ........................................................................ 3–27
  Java tab ......................................................................... 3–29
  Web Services tab .......................................................... 3–31
  Sonic Web Service tab ..................................................... 3–33
  Sonic Native tab ............................................................ 3–35
Validating and generating an Open Client proxy or Web service definition .............................................................................. 3–36
  Open Client interface validation ............................................ 3–36
  Open Client interface generation .......................................... 3–37
Object naming in ProxyGen ......................................................... 3–41
  Open Client interface naming conventions ................................ 3–41
  Automatic name conversions ............................................... 3–42
  Naming side effects and restrictions ...................................... 3–43
Example ................................................................................. 3–44

4. Programming Concepts ............................................................. 4–1
Connecting to an AppServer .......................................................... 4–2
  Types of connections ................................................................ 4–2
  Secure connections ................................................................... 4–3
  Specifying the AppServer connection ....................................... 4–4
Getting user input for run-time connection information .................... 4–5
Understanding proxy and Web service object methods ..................... 4–6
  Remote ABL methods .......................................................... 4–7
Passing parameters ...................................................................... 4–9
Accessing Open Client run-time properties ................................... 4–10
Handling errors in an Open Client application ................................. 4–11
  Client-side errors .................................................................. 4–11
  Server-side errors before remote ABL method execution ............... 4–11
  Server-side errors during remote ABL execution ....................... 4–12
Accessing an AppServer directly without proxies ............................. 4–13

Index ......................................................................................... Index–1
Tables

Table 1–1: Types of proxy objects .......................................................... 1–10
Table 2–1: Root digital certificate packages .............................................. 2–4
Table 3–1: Objects in an Open Client interface .......................................... 3–2
Table 3–2: ProxyGen and Batch ProxyGen tasks ....................................... 3–6
Table 4–1: Types of methods available for different proxy objects .......... 4–6
Table 4–2: OpenAPI Classes ................................................................. 4–13
### Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1–1</td>
<td>Open Client architecture</td>
<td>1–2</td>
</tr>
<tr>
<td>3–1</td>
<td>Starting ProxyGen (Windows)</td>
<td>3–7</td>
</tr>
<tr>
<td>3–2</td>
<td>ProxyGen main window—AppObject tab</td>
<td>3–7</td>
</tr>
<tr>
<td>3–3</td>
<td>ProxyGen main window—Procedures tab</td>
<td>3–8</td>
</tr>
<tr>
<td>3–4</td>
<td>Opening a Progress Version 9 file in OpenEdge</td>
<td>3–11</td>
</tr>
</tbody>
</table>
Preface

This Preface contains the following sections:

- Purpose
- Audience
- Organization
- Using this manual
- Typographical conventions
- Examples of syntax descriptions
- OpenEdge messages
- Third party acknowledgements
Purpose

The AppServer™ for OpenEdge® applications excels at encapsulating business functions written in ABL (Advanced Business Language) and making them available to ABL client applications anywhere on an enterprise network. With the Open Client Toolkit (a component of OpenEdge Studio), you can now extend your departmental applications (for example, written in .NET) or your Web-enabled applications written in Java™ to work with the same business functions used by your OpenEdge enterprise applications.

Using the Open Client Toolkit, you can create Open Client interfaces to a Java or .NET application or a Web service definition, to access the business functions on an AppServer. This guide describes how to use the tools provided with the Open Client Toolkit to accomplish this task.

Audience

This guide assumes that you are experienced in programming Java applications and applets, .NET applications, or Web services for your development and deployment environment, and that you have access to documentation on OpenEdge Release 10, especially OpenEdge Application Server: Developing AppServer Applications and its prerequisites.

Before building an application with the help of the Open Client Toolkit, you should be familiar with the information provided by the first two chapters of OpenEdge Application Server: Developing AppServer Applications.

Organization

Chapter 1, “Overview”

Describes the basic mechanisms provided to access AppServer functionality from a non-ABL (Open Client) application.

Chapter 2, “Configuration”

Details the configuration requirements for developing Open Client applications.

Chapter 3, “Generating Proxies and Web Service Definitions”

Explains how to use the Open Client Proxy Generator (ProxyGen) to encapsulate any ABL API available on an AppServer for access by an Open Client application, and how to manage the result for the application environment.

Chapter 4, “Programming Concepts”

Describes the mechanisms and techniques available to access the functionality encapsulated by ProxyGen from all Open Client applications.
Using this manual

OpenEdge provides a special purpose programming language for building business applications. In the documentation, the formal name for this language is ABL (Advanced Business Language). With few exceptions, all keywords of the language appear in all UPPERCASE, using a font that is appropriate to the context. All other alphabetic language content appears in mixed case.

For the latest documentation, see the OpenEdge Product Documentation Overview page on PSDN: http://communities.progress.com/pcom/docs/DOC-16074.

References to ABL compiler and run-time features

ABL is both a compiled and an interpreted language that executes in a run-time engine. The documentation refers to this run-time engine as the ABL Virtual Machine (AVM). When the documentation refers to ABL source code compilation, it specifies ABL or the compiler as the actor that manages compile-time features of the language. When the documentation refers to run-time behavior in an executing ABL program, it specifies the AVM as the actor that manages the specified run-time behavior in the program.

For example, these sentences refer to the ABL compiler’s allowance for parameter passing and the AVM’s possible response to that parameter passing at run time: “ABL allows you to pass a dynamic temp-table handle as a static temp-table parameter of a method. However, if at run time the passed dynamic temp-table schema does not match the schema of the static temp-table parameter, the AVM raises an error.” The following sentence refers to run-time actions that the AVM can perform using a particular ABL feature: “The ABL socket object handle allows the AVM to connect with other ABL and non-ABL sessions using TCP/IP sockets.”

References to ABL data types

ABL provides built-in data types, built-in class data types, and user-defined class data types. References to built-in data types follow these rules:

- Like most other keywords, references to specific built-in data types appear in all UPPERCASE, using a font that is appropriate to the context. No uppercase reference ever includes or implies any data type other than itself.

- Wherever integer appears, this is a reference to the INTEGER or INT64 data type.

- Wherever character appears, this is a reference to the CHARACTER, LONGCHAR, or CLOB data type.

- Wherever decimal appears, this is a reference to the DECIMAL data type.

- Wherever numeric appears, this is a reference to the INTEGER, INT64, or DECIMAL data type.

References to built-in class data types appear in mixed case with initial caps, for example, Progress.Lang.Object. References to user-defined class data types appear in mixed case, as specified for a given application example.
## Typographical conventions

This manual uses the following typographical conventions:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold</strong></td>
<td>Bold typeface indicates commands or characters the user types, provides emphasis, or the names of user interface elements.</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Italic typeface indicates the title of a document, or signifies new terms.</td>
</tr>
<tr>
<td>SMALL, BOLD CAPITAL LETTERS</td>
<td>Small, bold capital letters indicate OpenEdge key functions and generic keyboard keys; for example, GET and CTRL.</td>
</tr>
<tr>
<td>KEY1+KEY2</td>
<td>A plus sign between key names indicates a simultaneous key sequence: you press and hold down the first key while pressing the second key. For example, CTRL+X.</td>
</tr>
<tr>
<td>KEY1 KEY2</td>
<td>A space between key names indicates a sequential key sequence: you press and release the first key, then press another key. For example, ESCAPE H.</td>
</tr>
<tr>
<td><strong>Syntax:</strong></td>
<td></td>
</tr>
<tr>
<td>Fixed width</td>
<td>A fixed-width font is used in syntax statements, code examples, system output, and filenames.</td>
</tr>
<tr>
<td><em>Fixed-width italics</em></td>
<td>Fixed-width italics indicate variables in syntax statements.</td>
</tr>
<tr>
<td><em>Fixed-width bold</em></td>
<td>Fixed-width bold indicates variables with special emphasis.</td>
</tr>
<tr>
<td>UPPERCASE fixed width</td>
<td>Uppercase words are ABL keywords. Although these are always shown in uppercase, you can type them in either uppercase or lowercase in a procedure.</td>
</tr>
<tr>
<td>![three arrows]</td>
<td>This icon (three arrows) introduces a multi-step procedure.</td>
</tr>
<tr>
<td>![one arrow]</td>
<td>This icon (one arrow) introduces a single-step procedure.</td>
</tr>
<tr>
<td>Period (.) or colon (:))</td>
<td>All statements except DO, FOR, FUNCTION, PROCEDURE, and REPEAT end with a period. DO, FOR, FUNCTION, PROCEDURE, and REPEAT statements can end with either a period or a colon.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Large brackets indicate the items within them are optional.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Small brackets are part of ABL.</td>
</tr>
<tr>
<td>{ }</td>
<td>Large braces indicate the items within them are required. They are used to simplify complex syntax diagrams.</td>
</tr>
<tr>
<td>{ }</td>
<td>Small braces are part of ABL. For example, a called external procedure must use braces when referencing arguments passed by a calling procedure.</td>
</tr>
</tbody>
</table>
Examples of syntax descriptions

In this example, ACCUM is a keyword, and aggregate and expression are variables:

**Syntax**

```plaintext
ACCUM aggregate expression
```

FOR is one of the statements that can end with either a period or a colon, as in this example:

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

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

**Syntax**

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

In this example, the outer (small) brackets are part of the language, and the inner (large) brackets denote an optional item:

**Syntax**

```plaintext
INITIAL [ constant [ , constant ] ]
```

A called external procedure must use braces when referencing compile-time arguments passed by a calling procedure, as shown in this example:

**Syntax**

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

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

**Syntax**

```plaintext
PRESELECT [ EACH | FIRST | LAST ] record-phrase
```
In this example, you must include two expressions, and optionally you can include more. Multiple expressions are separated by commas:

**Syntax**

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

In this example, you must specify MESSAGE and at least one expression or SKIP [ (n) ], and any number of additional expression or SKIP [ ( n ) ] is allowed:

**Syntax**

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

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

**Syntax**

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

**Long syntax descriptions split across lines**

Some syntax descriptions are too long to fit on one line. When syntax descriptions are split across multiple lines, groups of optional and groups of required items are kept together in the required order.

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

**Syntax**

```
WITH [ ACCUM max-length ] [ expression DOWN ]
[ CENTERED ] [ n COLUMNS ] [ SIDE-LABELS ]
[ STREAM-IO ]
```
Complex syntax descriptions with both required and optional elements

Some syntax descriptions are too complex to distinguish required and optional elements by bracketing only the optional elements. For such syntax, the descriptions include both braces (for required elements) and brackets (for optional elements).

In this example, ASSIGN requires either one or more field entries or one record. Options available with field or record are grouped with braces and brackets:

Syntax

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

OpenEdge messages

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

- **Execution messages** inform you of errors encountered while OpenEdge is running a procedure; for example, if OpenEdge cannot find a record with a specified index field value.

- **Compile messages** inform you of errors found while OpenEdge is reading and analyzing a procedure before running it; for example, if a procedure references a table name that is not defined in the database.

- **Startup messages** inform you of unusual conditions detected while OpenEdge is getting ready to execute; for example, if you entered an invalid startup parameter.

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

- Continues execution, subject to the error-processing actions that you specify or that are assumed as part of the procedure. This is the most common action taken after execution messages.

- Returns to the Procedure Editor, so you can correct an error in a procedure. This is the usual action taken after compiler messages.

- Halts processing of a procedure and returns immediately to the Procedure Editor. This does not happen often.

- Terminates the current session.

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

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

If you encounter an error that terminates OpenEdge, note the message number before restarting.
Obtaining more information about OpenEdge messages

In Windows platforms, use OpenEdge online help to obtain more information about OpenEdge messages. Many OpenEdge tools include the following Help menu options to provide information about messages:

- Choose Help → Recent Messages to display detailed descriptions of the most recent OpenEdge message and all other messages returned in the current session.
- Choose Help → Messages and then type the message number to display a description of a specific OpenEdge message.
- In the Procedure Editor, press the HELP key or F1.

On UNIX platforms, use the OpenEdge pro command to start a single-user mode character OpenEdge client session and view a brief description of a message by providing its number.

To use the pro command to obtain a message description by message number:

1. Start the Procedure Editor:

   OpenEdge-install-dir/bin/pro

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

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

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

Third party acknowledgements

OpenEdge includes AdventNet - Agent Toolkit licensed from AdventNet, Inc.
http://www.adventnet.com. All rights to such copyright material rest with AdventNet.

OpenEdge includes ANTLR (Another Tool for Language Recognition) software Copyright © 2003-2006, Terence Parr All rights reserved. Neither the name of the author nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes software developed by the Apache Software Foundation (http://www.apache.org/). Copyright © 1999 The Apache Software Foundation. All rights reserved (Xerces C++ Parser (XML) and Xerces2 Java Parser (XML)); Copyright © 1999-2002 The Apache Software Foundation. All rights reserved (Xerces Parser (XML); and Copyright © 2000-2003 The Apache Software Foundation. All rights reserved (Ant). The names “Apache,” “Xerces,” “ANT,” and “Apache Software Foundation” must not be used to endorse or promote products derived from this software without prior written permission. Products derived from this software may not be called “Apache”, nor may “Apache” appear in their name, without
prior written permission of the Apache Software Foundation. For written permission, please contact apache@apache.org. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes Concurrent Java software Copyright 1994-2000 Sun Microsystems, Inc. All Rights Reserved. -Neither the name of or trademarks of Sun may be used to endorse or promote products including or derived from the Java Software technology without specific prior written permission; and Redistributions of source or binary code must contain the above copyright notice, this notice and the following disclaimers: This software is provided "AS IS," without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN MICROSYSTEMS, INC. AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN MICROSYSTEMS, INC. OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN MICROSYSTEMS, INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

OpenEdge includes DataDirect software Copyright © 1991-2007 Progress Software Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect for JDBC Type 4 driver); Copyright © 1993-2009 Progress Software Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect for JDBC); Copyright © 1988-2007 Progress Software Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect for ODBC); and Copyright © 1988-2007 Progress Software Corporation and/or its subsidiaries or affiliates. All Rights Reserved. (DataDirect Connect64 for ODBC).

OpenEdge includes DataDirect Connect for ODBC and DataDirect Connect64 for ODBC software, which include ICU software 1.8 and later - Copyright © 1995-2003 International Business Machines Corporation and others All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

OpenEdge includes DataDirect Connect for ODBC and DataDirect Connect64 for ODBC software, which include software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/). Copyright © 1998-2006 The OpenSSL Project. All rights reserved. And Copyright © 1995-1998 Eric Young (eay@cryptsoft.com). All rights reserved.

OpenEdge includes DataDirect products for the Microsoft SQL Server database which contain a licensed implementation of the Microsoft TDS Protocol.

OpenEdge includes software authored by David M. Gay. Copyright © 1991, 2000, 2001 by Lucent Technologies (dtoa.c); Copyright © 1991, 1996 by Lucent Technologies (g_fmt.c); and
Copyright © 1991 by Lucent Technologies (rnd_prods). Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which includes or contains a copy or modification of this software and in all copies of the supporting documentation for such software. THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.

OpenEdge includes software authored by David M. Gay. Copyright © 1998-2001 by Lucent Technologies All Rights Reserved (decsrtd.c; strtodg.c); Copyright © 1998, 2000 by Lucent Technologies All Rights Reserved (decsrtof.c; strtord.c); Copyright © 1998 by Lucent Technologies All Rights Reserved (dimp.c; gtod.h; gethex.c; gmisc.c; sum.c); Copyright © 1998, 1999 by Lucent Technologies All Rights Reserved (gdtoa.c; misc.c; smisc.c; ulp.c); Copyright © 1998-2000 by Lucent Technologies All Rights Reserved (gdtoaimp.h); Copyright © 2000 by Lucent Technologies All Rights Reserved (hd_init.c). Full copies of these licenses can be found in the installation directory, in the c:/OpenEdge/licenses folder. Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of Lucent or any of its entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

OpenEdge includes http package software developed by the World Wide Web Consortium. Copyright © 1994-2002 World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All rights reserved. This work is distributed under the W3C® Software License [http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231] in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

OpenEdge includes ICU software 1.8 and later - Copyright © 1995-2003 International Business Machines Corporation and others All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.


OpenEdge includes Infragistics NetAdvantage for .NET v2009 Vol 2 Copyright © 1996-2009 Infragistics, Inc. All rights reserved.
OpenEdge includes JSTL software Copyright 1994-2006 Sun Microsystems, Inc. All Rights Reserved. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes OpenSSL software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/). Copyright © 1998-2007 The OpenSSL Project. All rights reserved. This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com). Copyright © 1995-1998 Eric Young (eay@cryptsoft.com) All rights reserved. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project. Software distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes Quartz Enterprise Job Scheduler software Copyright © 2001-2003 James House. All rights reserved. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product. This product uses and includes within its distribution, software developed by the Apache Software Foundation (http://www.apache.org/).

OpenEdge includes code licensed from RSA Security, Inc. Some portions licensed from IBM are available at http://oss.software.ibm.com/1cu4j/.

OpenEdge includes the RSA Data Security, Inc. MD5 Message-Digest Algorithm. Copyright ©1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.

OpenEdge includes Sonic software, which includes software developed by Apache Software Foundation (http://www.apache.org/). Copyright © 1999-2000 The Apache Software Foundation. All rights reserved. The names “Ant”, “Axis”, “Xalan,” “FOP,” “The Jakarta Project”, “Tomcat”, “Xerces” and/or “Apache Software Foundation” must not be used to endorse or promote products derived from the Product without prior written permission. Any product derived from the Product may not be called “Apache”, nor may “Apache” appear in their name, without prior written permission. For written permission, please contact apache@apache.org.

OpenEdge includes Sonic software, which includes software Copyright © 1999 CERN - European Organization for Nuclear Research. Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. CERN makes no representations about the suitability of this software for any purpose. It is provided "as is" without expressed or implied warranty.

OpenEdge includes Sonic software, which includes software developed by ExoLab Project (http://www.exolab.org/). Copyright © 2000 Intalio Inc. All rights reserved. The names “Castor” and/or “ExoLab” must not be used to endorse or promote products derived from the Products without prior written permission. For written permission, please contact info@exolab.org. Exolab, Castor and Intalio are trademarks of Intalio Inc.
OpenEdge includes Sonic software, which includes software developed by IBM. Copyright © 1995-2003 International Business Machines Corporation and others. All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation. Software distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product. Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

OpenEdge includes Sonic software, which includes the JMX Technology from Sun Microsystems, Inc. Use and Distribution is subject to the Sun Community Source License available at http://sun.com/software/communitysource.

OpenEdge includes Sonic software, which includes software developed by the ModelObjects Group (http://www.modelobjects.com). Copyright © 2000-2001 ModelObjects Group. All rights reserved. The name “ModelObjects” must not be used to endorse or promote products derived from this software without prior written permission. Products derived from this software may not be called “ModelObjects”, nor may “ModelObjects” appear in their name, without prior written permission. For written permission, please contact djacobs@modelobjects.com.

OpenEdge includes Sonic software, which includes code licensed from Mort Bay Consulting Pty. Ltd. The Jetty Package is Copyright © 1998 Mort Bay Consulting Pty. Ltd. (Australia) and others.

OpenEdge includes Sonic software, which includes files that are subject to the Netscape Public License Version 1.1 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.mozilla.org/NPL/. Software distributed under the License is distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License. The Original Code is Mozilla Communicator client code, released March 31, 1998. The Initial Developer of the Original Code is Netscape Communications Corporation. Portions created by Netscape are Copyright 1998-1999 Netscape Communications Corporation. All Rights Reserved.

OpenEdge includes Sonic software, which includes software developed by the University Corporation for Advanced Internet Development http://www.ucaid.edu Internet2 Project. Copyright © 2002 University Corporation for Advanced Internet Development, Inc. All rights reserved. Neither the name of OpenSAML nor the names of its contributors, nor Internet2, nor the University Corporation for Advanced Internet Development, Inc., nor UCAID may be used to endorse or promote products derived from this software and products derived from this software may not be called OpenSAML, Internet2, UCAID, or the University Corporation for Advanced Internet Development, nor may OpenSAML appear in their name without prior written permission of the University Corporation for Advanced Internet Development. For written permission, please contact opensaml1@opensaml.org.

OpenEdge includes the UnixWare platform of Perl Runtime authored by Kiem-Phong Vo and David Korn. Copyright © 1991, 1996 by AT&T Labs. Permission to use, copy, modify, and
distribute this software for any purpose without fee is hereby granted, provided that this entire
notice is included in all copies of any software which is or includes a copy or modification of
this software and in all copies of the supporting documentation for such software. THIS
SOFTWARE IS BEING PROVIDED “AS IS”, WITHOUT ANY EXPRESS OR IMPLIED
WARRANTY. IN PARTICULAR, NEITHER THE AUTHORS NOR AT&T LABS MAKE
ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE
MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR
PURPOSE.

OpenEdge includes Vermont Views Terminal Handling Package software developed by

OpenEdge includes XML Tools, which includes versions 8.9 of the Saxon XSLT and XQuery
Processor from Saxonica Limited (http://www.saxonica.com/) which are available from
SourceForge (http://sourceforge.net/projects/saxon/). The Original Code of Saxon
comprises all those components which are not explicitly attributed to other parties. The Initial
Developer of the Original Code is Michael Kay. Until February 2001 Michael Kay was an
employee of International Computers Limited (now part of Fujitsu Limited), and original code
developed during that time was released under this license by permission from International
Computers Limited. From February 2001 until February 2004 Michael Kay was an employee
of Software AG, and code developed during that time was released under this license by
permission from Software AG, acting as a “Contributor”. Subsequent code has been developed
by Saxonica Limited, of which Michael Kay is a Director, again acting as a “Contributor”. A
small number of modules, or enhancements to modules, have been developed by other
individuals (either written especially for Saxon, or incorporated into Saxon having initially been
released as part of another open source product). Such contributions are acknowledged
individually in comments attached to the relevant code modules. All Rights Reserved. The
contents of the Saxon files are subject to the Mozilla Public License Version 1.0 (the “License”);
you may not use these files except in compliance with the License. You may obtain a copy of
the License at http://www.mozilla.org/MPL/ and a copy of the license can also be found in the
installation directory, in the c:/OpenEdge/licenses folder. Software distributed under the
License is distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either
express or implied. See the License for the specific language governing rights and limitations
under the License.

OpenEdge includes XML Tools, which includes Xs3P v1.1.3. The contents of this file are
subject to the DSTC Public License (DPL) Version 1.1 (the “License”); you may not use this
file except in compliance with the License. A copy of the license can be found in the installation
directory, in the c:/OpenEdge/licenses folder. Software distributed under the License is
distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or
implied. See the License for the specific language governing rights and limitations under the
License. The Original Code is xs3p. The Initial Developer of the Original Code is DSTC.
Portions created by DSTC are Copyright © 2001, 2002 DSTC Pty Ltd. All rights reserved.

OpenEdge includes YAJL software Copyright 2007, Lloyd Hilaiel. Redistribution and use in
source and binary forms, with or without modification, are permitted provided that the
following conditions are met: 1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form
must reproduce the above copyright notice, this list of conditions and the following disclaimer
in the documentation and/or other materials provided with the distribution. 3. Neither the name
of Lloyd Hilaiel nor the names of its contributors may be used to endorse or promote products
derived from this software without specific prior written permission. THIS SOFTWARE IS
PROVIDED BY THE AUTHOR “AS IS” AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Overview

The Open Client Toolkit exposes an application service running on an AppServer™ to Open Clients (non-ABL clients). It is a component of several products: OpenEdge® Architect, OpenEdge Studio, WebSpeed® Workshop, 4GL Development System, and OpenEdge Development Server. The Open Client Toolkit allows you to generate an interface, tailored for your particular type of Open Client, that encapsulates the remote ABL procedures and functions supported on an AppServer. Your Open Client application can then access these AppServer procedures and functions through methods of the generated Open Client interface.

The supported Open Clients are Java, .NET, and OpenEdge Web services. Open Clients support both intranet and Internet access to the AppServer. When directly accessing the AppServer (.NET and Java only), you can use the Secure Sockets Layer (SSL) to exchange encrypted data transmissions. When you use the Internet (all Open Clients), you can use HTTP or HTTPS (abbreviated as HTTP/S) to communicate through firewalls and HTTPS to exchange encrypted data transmissions.

This chapter describes the Open Client architecture and how the Open Client Proxy Generator (ProxyGen) maps AppServer procedures to proxy objects for access by .NET and Java Open Clients, and to produce Web service definitions for consumption by Web service clients. Other chapters provide information about how to develop and deploy Open Client applications and OpenEdge Web services. The sections of this chapter include:

- Architecture
- Developing and accessing an Open Client interface
- Accessing AppServer functionality
Architecture

The Open Client architecture allows Open Clients to access application services that are organized into separate ABL source files (usually .p (procedure) files, .w files, or both).

There are several components involved in accessing AppServer functionality from Open Clients. Figure 1–1 shows the basic components involved when an Open Client application communicates with an AppServer on a company intranet or the Internet.

![Open Client architecture diagram](image)

**Figure 1–1: Open Client architecture**

*Note:* OpenEdge also supports SSL intranet connections between the WSA or AIA and the AppServer. For more information, see *OpenEdge Application Server: Administration*.

For Java and .NET, the client programmer writes the client application in Java™ or any .NET language. This application typically executes remote procedures and functions in an AppServer session through methods on a proxy object you generate using ProxyGen.

For OpenEdge Web services, the client programmer writes the client application in any Web-service-enabled language, such as Java, VB.NET, or ABL. This application sends requests to an OpenEdge Web Service Adapter (WSA), to execute remote procedures and functions in an AppServer session. The available methods and location of the WSA are specified in a Web Services Definition Language (WSDL) file generated when the Web service is deployed to the WSA. ProxyGen is used before deployment, to define the Web service.
ProxyGen is an Open Client development tool that defines and generates Java and .NET proxies (for use by Java and .NET Open Clients, respectively) and generates Web service definitions (for use by Web service clients). The proxy objects generated by ProxyGen are first class Java or .NET classes and use the Open Client Runtime to access the AppServer. For more information about using ProxyGen to generate proxies and Web service definitions, see the “Building and deploying an Open Client application” section on page 2–2 and Chapter 3, “Generating Proxies and Web Service Definitions.”

A key feature of the Open Client is support for relational data exchange. This allows ABL ProDataSets (.NET and Java Open Clients only) and temp-tables to be passed between the AppServer and the client application, which sees the data in its native environment. For Java, a ProDataSet parameter (DATASET or DATASET-HANDLE data type) maps to an OpenEdge ProDataGraph object, which is an implementation of the Java Service Data Objects (Java SDO) DataGraph interface. For .NET, a ProDataSet parameter maps to an ADO.NET DataSet object. Similarly, for Java, a temp-table parameter (TABLE or TABLE-HANDLE data type) can map (your choice) to either an SQL ResultSet object or an OpenEdge ProDataGraph object (as the wrapper for a single temp-table). For .NET, a temp-table parameter maps to an ADO.NET DataTable object. In this way, Open Client applications can access any OpenEdge database or DataServer connected to and exposed by the AppServer application.

Note: OpenEdge Web services do not support ProDataSet parameters. However, you can serialize ProDataSets to XML and pass them in the Web service as LONGCHAR variables.
Developing and accessing an Open Client interface

To prepare an Open Client application to access an AppServer application service through an Open Client interface (proxy or Web service definition) requires using the Open Client Toolkit to run ProxyGen. ProxyGen is an OpenEdge development tool that allows an ABL developer to identify and expose the AppServer functionality in an application service through an interface.

To develop and run an Open Client application to access an Open Client interface, you must use several tools and procedures.

To develop and access an Open Client interface:

1. Ensure your development environment meets the requirements for developing Open Client applications. For more information, see Chapter 2, “Configuration.”

2. Use ProxyGen to define and generate a Java proxy, .NET proxy, or Web service definition. See Chapter 3, “Generating Proxies and Web Service Definitions,” for additional information.

   **Note:** You can skip this step if you want to access only remote SmartDataObjects from a Java application. For more information, see OpenEdge Development: Java Open Clients.

3. Write a client application that uses the proxy objects generated in Step 2 or the predefined SmartDataObject proxy objects, or write a Web service client application based on the methods defined in the WSDL document. For more information, see Chapter 4, “Programming Concepts,” OpenEdge Development: Java Open Clients, OpenEdge Development: .NET Open Clients, and OpenEdge Development: Web Services.

4. Manage Digital Certificates, if your Open Client uses HTTPS or intranet SSL to communicate with the AppServer. For more information, see the “Managing Open Client root digital certificates” section on page 2–3.

5. For Java and .NET, deploy the proxy, Open Client application, the Open Client Runtime, your optional digital certificates, and supporting software to your client machine. For Web services, deploy your Web service definition to the WSA using OpenEdge Explorer and Progress Explorer. For more information, see Chapter 2, “Configuration.”

6. Start any DataServers or database servers accessed by the AppServer. For more information, see the appropriate OpenEdge DataServer Guide and OpenEdge Data Management: Database Administration.

7. Start the AppServer. For more information, see OpenEdge Application Server: Developing AppServer Applications.

8. If you are using HTTP/S with .NET and Java Open Clients, start the Java Servlet Engine (JSE) in a Web server and run the AppServer Internet Adapter (AIA). For more information on the AIA, see OpenEdge Application Server: Administration. For Web services, start the JSE in a Web server and run the WSA. For more information on the WSA, see OpenEdge Development: Web Services.

9. Run your Open Client application as you have designed it.
Accessing AppServer functionality

Application services on an AppServer are organized into physical procedure files. To generate an Open Client interface, you must compile and save the r-code for these files before ProxyGen can access them, even if your deployment strategy ultimately relies on the source files. (R-code is the binary run code executed by the ABL interpreter.) Once the AppServer r-code files are available, you can use ProxyGen to identify and customize Open Client access to the procedures and functions defined by these files. To make the r-code files available to ProxyGen, you can:

- Run ProxyGen on the same machine where the r-code resides or a machine with access to a network file system where the r-code resides.
- Copy the r-code files to the machine where you run ProxyGen (if this is different than the AppServer machine), maintaining the directory structure underneath the AppServer PROPATH.

When you organize these procedure files using ProxyGen, and program the application to use them, you must consider two fundamental features of an Open Client application:

- The session model
- The object model

The session model

Open Client applications can function in two different session models that control how the client and AppServer interact:

- **Session-managed** — The client maintains a dedicated physical connection to a single AppServer that handles each request for a given application service. In this session model, the client and AppServer share a common and persistent connection context over which run-time state can be maintained for the duration of the connection. Therefore, this session model is ideally suited for applications that must manage a large and complex transaction scope that includes multiple requests between the client and the AppServer.

- **Session-free** — The client maintains a logical connection to a given application service, and each client request for that application service can be handled by any one of many AppServer resources (that is, one or more AppServers and one or more AppServer agents of any one AppServer) that support the given application service. In this session model, the client shares no persistent connection context with any one AppServer resource. Each client request can execute on a different AppServer and on a different agent of any one AppServer. This session model supports maximum availability of AppServer resources. Therefore, this session model is ideally suited for applications that require maximum upward scalability and a transaction scope of no more than a single client request.

Session model configuration

The session model is determined by the AppServer configuration, in particular the setting of its operating mode, which can be state-reset, state-aware or stateless for a session-managed application service, and can only be state-free for a session-free application service. However, you must also implement your Open Client interface and program any Open Client applications according to the requirements of the application service, including its session model.
For more information on AppServer configuration and specific programming requirements for the different session models, see *OpenEdge Application Server: Developing AppServer Applications*.

**Connection management**

One of the most important requirements in session model programming is how you manage the client connection for each session model. For a session-managed application, the client connection is a *physical connection*, a one-to-one relationship between the client and AppServer that supports a particular application service. For a session-free application, the client connection is a *logical connection*, a one-to-many relationship between the client and any number of AppServer resources that support a single application service. To support logical connections, the session-free Open Client maintains a *connection pool*, which is a configured connection resource that maintains a number of physical AppServer connections to use for all client requests invoked on a given application service.

Each Open Client manages this the session-free connection pool in its own way on behalf of the Open Client application. For .NET and Java Open Clients, the proxies manage the connection pool from run-time property settings that you can modify before and sometimes after establishing the logical connection. For Web service Open Clients, the WSA manages the connection pool for each Web service from similar Web service property settings that can be changed by the Web service administrator.

For more information on physical and logical connections and how they interact with the AppServer, see the sections on the AppServer connection model in *OpenEdge Getting Started: Application and Integration Services*.

**Open Client interface and programming**

As noted previously, you must implement the Open Client interface and program the Open Client application according to the session model.

The Open Client interface includes various object types whose function is generally designed to support the session model. You define these objects in ProxyGen to support the given application session model for all Open Clients. For Web services and Sonic ESB, you must also indicate the session model for which the interface is defined, because ProxyGen generates a different Web service definition depending on the session model. For more information, see Chapter 3, “Generating Proxies and Web Service Definitions.”

You must program an Open Client application to reflect the Open Client object design and its support of the required session model. For information on the basic session model requirements for programming applications, see *OpenEdge Application Server: Developing AppServer Applications*. The following section describes the Open Client object model and how it supports the different session models for all Open Clients. For more information on session model programming for each Open Client type, see:

- *OpenEdge Development: .NET Open Clients* for .NET Open Clients
- *OpenEdge Development: Java Open Clients* for Java Open Clients
- *OpenEdge Development: Web Services* for Web services
The object model

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)

Note: OpenEdge supports an API (OpenAPI) for Java and .NET Open Clients that allows you to directly access an AppServer application service from an Open Client without the need to define and generate proxy objects using ProxyGen. However, to use this API, you must organize and manage all access to the AppServer at run time. Although, you do not use proxy objects with the OpenAPI, the basic principles of working with AppObjects and ProcObjects also apply to accessing an AppServer using the OpenAPI. For more information on the OpenAPI, see Chapter 4, “Programming Concepts.”

All these proxy objects organize ABL procedures on the AppServer, but in different ways. The following sections describe the different ways of organizing proxy objects and contain information about:

- ABL procedures
- AppObjects
- SubAppObjects
- ProcObjects
- ADM SmartObjects and SmartDataObjects (Java only)
- Object relationships
- Open Client access to objects
ABL procedures

Before describing the three types of proxy objects, it helps to understand basic concepts governing the operation of ABL procedures. First, any file that is executable by ABL is an external procedure. An external procedure can contain one or more internal procedures or user-defined functions that execute within and share the context of the external procedure. You can execute an external procedure as either:

- **A non-persistent procedure** — The procedure executes and returns to the caller, removing all trace of its context from memory after returning. Any internal procedures and functions that it defines can be executed only by the procedure itself. A non-persistent procedure executes and returns as a unit without exposing any of its context to the caller.

- **A persistent procedure** — The procedure leaves its context active after completing execution of the main block. Internal procedures and user-defined functions remain available for future execution. A persistent procedure instantiates a procedure object that exposes its context to the caller through the internal procedures and user-defined functions it defines. For more information on ABL procedures, see *OpenEdge Getting Started: ABL Essentials*.

AppObjects

Each AppObject encapsulates a set of business logic deployed at a particular AppServer and establishes a connection to that AppServer (session-managed) or to the application service represented by one or more AppServers accessed from a connection pool (session-free).

Although you can encapsulate an AppServer’s entire functionality (application service) within one AppObject, dividing a large AppServer application into one AppObject and several SubAppObjects offers advantages, particularly for session-managed applications. Multiple objects provide better logical organization and separate name spaces for each object. Also, they help avoid the instantiation of large objects that may be slow to load and difficult to maintain.

For session-free applications, it is often best to define the entire application service within the AppObject alone. The behavior of session-free connections makes other types of Open Client objects less useful, and even an obstacle to good session-free application performance, as the following sections describe.

SubAppObjects

Each SubAppObject encapsulates a set of business logic deployed at a particular AppServer. It differs from an AppObject only in that it does not establish its own connection to an AppServer but shares the one established by its associated AppObject.

The associated AppObject defines a class factory method for creating each SubAppObject that shares its AppServer connection. This method exchanges no communications with the AppServer; rather, it only performs operations to instantiate the SubAppObject on the client.

SubAppObjects are less useful for session-free applications, especially for those that support Web services. For .NET and Java Open Clients, SubAppObjects can help to organize functionality for session-free application services. However, because of the implementation of SubAppObjects for Web services, SubAppObjects add complexity to Web service request management on the client that you might want to avoid. Therefore, if you intend your Open Client interface to define Web services, you might prefer to avoid the use of SubAppObjects and encapsulate the application service interface in the single AppObject for all Open Clients that use it. For more information on how SubAppObjects work with session-free Web services, see *OpenEdge Development: Web Services*.
ProcObjects

Each ProcObject encapsulates one persistent procedure running on an AppServer. The ProcObject exposes all non-PRIVATE internal procedures and user-defined functions to the client that you do not explicitly omit in ProxyGen when you define the ProcObject.

A ProcObject shares a connection established by an associated AppObject. You can create the ProcObject using a class factory method on the AppObject or SubAppObject to which this procedure was added in ProxyGen.

When you call the ProcObject class factory method on the AppObject or SubAppObject, it creates the ProcObject on the client and executes the corresponding procedure persistently on the AppServer. Any parameters required by this procedure are required as parameters on the class factory method. The ProcObject saves a handle to the persistent procedure for future calls to its internal procedures and functions and for releasing and disconnecting the object from the AppServer application.

For session-free applications, ProcObjects have limited utility and tend to interfere with session-free application performance. Part of the value from ProcObjects is that they provide a means to maintain run-time context between the client and AppServer. Much of the value from session-free application services is that AppServer resources are more readily available to serve client requests. However, a ProcObject reserves an AppServer resource for use by the client that instantiates the ProcObject until that ProcObject is released. This ties up a physical connection and reduces the AppServer resources available for requests from other clients, which can in turn reduce apparent application performance across the client domain, possibly to intolerable levels if every client instantiates a ProcObject and AppServer resources are scarce to begin with.

Therefore, Progress Software Corporation strongly recommends that you avoid the use of ProcObjects in session-free applications.

ADM SmartObjects and SmartDataObjects (Java only)

The AppBuilder, which is part of the OpenEdge Application Development Environment (ADE), allows you to build preprogrammed persistent procedures that adhere to the OpenEdge Application Development Model (ADM). The ADM allows you to build persistent procedures as reusable application building blocks known as SmartObjects. The Open Client Toolkit allows you to access a type of SmartObject known as a SmartDataObject, which dynamically accesses and updates temp-table data. As a result, you can run a SmartDataObject on the AppServer like any persistent procedure and, using the Open Client Runtime, you can access the SmartDataObject from a Java application. For more information on SmartDataObjects, see OpenEdge Development: AppBuilder.

In Java, you also can access the SmartDataObject as an extended Java Database Connectivity (JDBC) 2 ResultSet.

For more information, see the chapter on using SmartDataObjects from Java clients in OpenEdge Development: Java Open Clients.

Note: Because you must access SmartDataObjects as ProcObjects in the Open Client application, you might consider avoiding them for use in session-free applications. For more information, see the “ProcObjects” section on page 1–9.
Object relationships

In an Open Client application, you create an AppObject first, to establish a connection to an AppServer (session-managed) or application service (session-free). Then you can create SubAppObjects (through the associated AppObject) and ProcObjects (through an associated AppObject or SubAppObject). You use a special Class Factory method to create a SubAppObject or ProcObject. A proxy or Web Service definition comprises one AppObject and all SubAppObjects and ProcObjects that share the same AppServer connection.

Although there is a creation hierarchy among the different types of proxy objects, there is no functional hierarchy. All objects share the AppServer connection with equal status. Objects can be released (have their context removed from client memory) in any order, with no effect on objects still in use. The AppServer (session-managed) or application service (session-free) connection remains intact until the last object using it is released.

Table 1–1 summarizes how each type of proxy object encapsulates and accesses AppServer functionality.

Table 1–1: Types of proxy objects

<table>
<thead>
<tr>
<th>This type of proxy object . . .</th>
<th>Provides this functionality . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppObject</td>
<td>• A connection to an AppServer (session-managed) or application service (session-free)</td>
</tr>
<tr>
<td></td>
<td>• Application methods that execute ABL non-persistent procedure files</td>
</tr>
<tr>
<td></td>
<td>• Class factory methods that create SubAppObjects</td>
</tr>
<tr>
<td></td>
<td>• Class factory methods that create ProcObjects</td>
</tr>
<tr>
<td></td>
<td>• (Java Only) A special class factory method that creates a built-in ProcObject for accessing a remote SmartDataObject as an extended JDBC 2 ResultSet</td>
</tr>
<tr>
<td>SubAppObject</td>
<td>• A shared AppServer (session-managed) or application service (session-free) connection through an AppObject</td>
</tr>
<tr>
<td></td>
<td>• Application methods that execute ABL non-persistent procedure files</td>
</tr>
<tr>
<td></td>
<td>• Class factory methods that create ProcObjects</td>
</tr>
<tr>
<td></td>
<td>• (Java Only) A special class factory method that creates a built-in ProcObject for accessing a remote SmartDataObject as an extended JDBC 2 ResultSet</td>
</tr>
<tr>
<td>ProcObject</td>
<td>• A shared AppServer (session-managed) or application service (session-free) connection through an AppObject</td>
</tr>
<tr>
<td></td>
<td>• Execution of a remote ABL persistent procedure</td>
</tr>
<tr>
<td></td>
<td>• Application methods that execute ABL internal procedures and user-defined functions</td>
</tr>
</tbody>
</table>

In addition to the functionality described in Table 1–1, each object type from the Open Client object model provides a set of common methods for run-time management of the object and its application service connection.
Open Client access to objects

Once you define an AppObject, its application methods and any related SubAppObjects and ProcObjects, ProxyGen can generate the proxy or Web service definition that exposes these objects to Open Clients.

For Java and .NET clients, the client code accesses the proxy in the client’s native environment. As a result, a Java client sees the proxy (and the AppServer application service) as a set of Java classes, while a .NET client sees the proxy as a set of .NET 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. See Chapter 2, “Configuration.”

For Web service clients, the client accesses the Web service in the client’s native environment. The Web service definition (the WSDL document) is available through the WSA instance hosting the Web service. To use the Web service, there is no deployment requirement for the client.
Configuration

Each step in the development of an Open Client application has prerequisites. You must fulfill all these prerequisites before beginning the specified task.

This chapter describes the configuration prerequisites for performing Open Client development and deployment tasks, as outlined in the following sections:

• Building and deploying an Open Client application

• Managing Open Client root digital certificates
Building and deploying 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 and/or a Web service definition, 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 OpenEdge Development: .NET Open Clients, OpenEdge Development: Java Open Clients, or OpenEdge Development: Web Services.
Managing Open Client root digital certificates

OpenEdge supports secure access between the Open Client and the AppServer over an intranet using the Secure Sockets Layer (SSL-enabled AppServer) and over the Internet using HTTPS (see Chapter 4, “Programming Concepts”). When a client connects to an SSL-enabled AppServer or to a Web server using HTTPS, the server sends its digital certificate to the client to prove its identity. The client is responsible for authenticating that identity. Authentication is done using root digital certificates. The client does this by locating the local root digital certificates on the client machine that corresponds to the server certificate issued by the Certificate Authority (CA) for the AppServer (intranet) or Web server (Internet). This local certificate is then validated against the server certificate to authenticate the identity of the server.

To meet the demands of the worldwide software distribution that Progress Software Corporation supports, a set of international CA root digital certificates is distributed with the OpenEdge installation in the OpenEdge certificate store (OpenEdge-Install-Dir/certs directory). Though these root digital certificates can be distributed and used as is, the size might make it impractical to use. For example, you might not want to use these root digital certificates with applets, due to the download time required.

Included with the OpenEdge distribution is a built-in OpenEdge CA root digital certificate (pscca.cer) to support SSL-enabled AppServer access without external CA support. For more information on digital certificates and the OpenEdge certificate store, see OpenEdge Getting Started: Core Business Services. Each Open Client type supports certificate management in a different way.

Java Open Clients

Progress Software Corporation recommends your application or applet be distributed with a minimal set of root digital certificates from the set provided. You then can provide application or applet deployers with the capability of adding one or more root digital certificates using the setCertificateStore() method (or by setting the certificateStore property), to satisfy their specific requirements. This provides deployers with a way to develop their specific method of distributing their own root digital certificates to their users and then dynamically configuring the Open Client application or applet to use them.

Note: If the Web server is configured to use a digital certificate issued by a private Certificate Authority, it would never be included in the set distributed with OpenEdge.

The compressed filenames of the root digital certificates do not indicate which certificates are included; however, OpenEdge provides a certificate management utility you can use to view and manage the files. For more information, see OpenEdge Development: Java Open Clients.
Root digital certificate packages

Table 2–1 lists the Open Client application type and root digital certificate packages OpenEdge distributes.

Table 2–1: Root digital certificate packages

<table>
<thead>
<tr>
<th>For . . .</th>
<th>OpenEdge supplies this root digital certificate package . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Open Client applications</td>
<td>psccerts.jar</td>
</tr>
<tr>
<td>Java applets running only in the Netscape browser</td>
<td>psccertsn.jar</td>
</tr>
<tr>
<td>Java applets running only in the Internet Explorer browser</td>
<td>psccerts.zip</td>
</tr>
</tbody>
</table>

In all cases, the root digital certificates OpenEdge distributes are in binary (DER) format. All certificate files use compressed filenames and have the .cer file extension. The exception is for the Netscape Internet Browser, where the root digital certificates are in files with the .txt file extension. These root digital certificate packages also include a copy of the root digital certificate for the built-in OpenEdge CA, pscca.cer.

.NET Open Clients

Microsoft .NET has its own method for managing digital certificates that you can use to access root certificates from the OpenEdge certificate store. For more information, see the Microsoft .NET documentation.

Web services

Access to root digital certificates for Web service clients depends on the type of client; for example, Java or .NET. For more information, see the appropriate documentation for your type of client.
Generating Proxies and Web Service Definitions

This chapter is a general guide for defining OpenEdge® Web services and for generating Open Client proxies for Java or .NET Open Client applications using ProxyGen, as described in the following sections:

- Open Client interfaces
- Getting started with ProxyGen
- Defining an Open Client interface using ProxyGen
- Defining an AppObject or SubAppObject
- Defining procedures in AppObjects and SubAppObjects
- Saving the Open Client interface definition in a project file
- Specifying generation settings
- Validating and generating an Open Client proxy or Web service definition
- 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 (see Chapter 1, “Overview”). This model organizes and maps OpenEdge procedures that run on an AppServer 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 Web service.

Objects in an Open Client interface

Any single Open Client interface maps an AppServer application to a series of object types, as shown in Table 3–1.

Table 3–1: Objects in an Open Client interface

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Occurrences</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppObject</td>
<td>One required per application</td>
<td>Encapsulates zero or more external ABL procedures as directly callable methods and zero or more ProcObjects as class factory methods</td>
</tr>
<tr>
<td>SubAppObject</td>
<td>Zero or more per application</td>
<td>Encapsulates zero or more external ABL procedures as directly callable methods and zero or more ProcObjects as class factory methods</td>
</tr>
<tr>
<td>ProcObject</td>
<td>Zero or more per application</td>
<td>A persistent procedure that encapsulates one or more ABL internal procedures and user-defined functions as directly callable methods</td>
</tr>
</tbody>
</table>

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 procedure, which is a special type of ABL external 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 in a special manner that leaves an active named context available in session memory when its own execution completes. This named session context (or persistent procedure) contains one or more internal procedures or user-defined functions declared within the persistent procedure, along with any commonly accessible data declarations.
These internal procedures and functions are then available for access by callers external to the persistent procedure that defines them. On the AppServer, these callers can include other external ABL procedures that execute in the same OpenEdge session, or they can include clients of an AppServer (including Open Clients and Web service clients) that call these internal procedures and functions as methods. A persistent procedure is, in itself, a kind of ABL object that encapsulates functionality for use by other external procedures. ProxyGen maps each such persistent 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 procedures and persistent procedures (ProcObjects) for all external procedures that you include in your Open Client interface. As indicated previously, a persistent procedure is an external procedure that leaves behind an active named context when it completes execution. Other than this, and the need to manage persistent procedure context, there is no difference between non-persistent and persistent procedures within ABL.

ABL makes an external procedure persistent or non-persistent by the way you invoke the procedure, not by the procedure’s definition or content. Thus, you can arbitrarily include any OpenEdge external procedure in an Open Client interface as non-persistent or persistent. For Web services and 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 the “Session models” section on page 3–3) and object types involved.

**Encapsulating functionality**

For an Open Client proxy or a 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 as methods and those intended as ProcObjects. Any procedure designed to run persistently is typically is 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 AppServer 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 AppServer performance.

For more information on how these built-in methods work with each object type, see Chapter 4, “Programming Concepts,” 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 AppServer is configured and fundamentally determines how an Open Client and AppServer can communicate with one another. (In ProxyGen, you only need to specify the session model for Web services.)
**Session models and Open Clients**

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

For more information on session models and how they affect Open Client applications, see Chapter 1, “Overview.” For a complete definition of session models and how they affect the interaction between an AppServer and a client, see *OpenEdge Application Server: Developing AppServer Applications*.

**Specifying the session model**

In ProxyGen, you only need to specify the session model explicitly for 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 AppServer (session-managed) or an application service (session-free).

The session model is really a property of the application service, and the AppServer enables support for and determines the session model for the application service by the setting of the AppServer 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 AppServer Applications*.

**Making 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 AppServer. 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 AppServer. However you set up your PROPATH, you must ensure that the relative paths under the PROPATH component settings remain the same between the AppServer and ProxyGen.
Getting started with ProxyGen

ProxyGen is a development tool that defines and generates Java and .NET proxies and generates 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
- Using ProxyGen execution options
- Starting and using ProxyGen
- Editing an Open Client interface
- Specifying generation settings
- Saving Open Client interfaces in a project file
- Generating an Open Client proxy or Web service definition
- Running Batch ProxyGen
- Converting a project file from Progress Version 9 to OpenEdge
- Converting a preferences file from Progress Version 9 to OpenEdge

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 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 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 Web service definitions.
  - On UNIX, Batch ProxyGen can generate only Java Open Client proxies and 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 AppServer ABL procedures. You can use either ProxyGen or Batch ProxyGen to generate the actual Open Client proxy or Web service definition.

Table 3–2 shows how to complete these basic tasks with ProxyGen and Batch ProxyGen.

### Table 3–2: ProxyGen and Batch ProxyGen tasks

<table>
<thead>
<tr>
<th>Task</th>
<th>ProxyGen</th>
<th>Batch ProxyGen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editing an Open Client interface</td>
<td>X</td>
<td>–</td>
</tr>
<tr>
<td>Specifying generation settings, including client types</td>
<td>X</td>
<td>–</td>
</tr>
<tr>
<td>Saving Open Client interfaces in a project file</td>
<td>X</td>
<td>–</td>
</tr>
<tr>
<td>Generating an Open Client proxy or Web service definition</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**Using 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.

To convert a preferences file from Progress Version 9 to OpenEdge format, see the “Converting a preferences file from Progress Version 9 to OpenEdge” section on page 3–12.
Starting and using ProxyGen

Start ProxyGen on Windows by opening the **Proxy Generator** icon in the **OpenEdge** program group, as shown in Figure 3–1.

![Figure 3–1: Starting ProxyGen (Windows)](image)

The **ProxyGen** main window then opens, as shown in Figure 3–2.

![Figure 3–2: ProxyGen main window—AppObject tab](image)
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 and persistent procedures, as shown in Figure 3–3.

![ProxyGen main window—Procedures tab](image)

**Figure 3–3:** ProxyGen main window—Procedures tab

### Editing 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** in Figure 3–2). Any child nodes in the tree view (for example, **Customer** in Figure 3–2) 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 Release 10 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, including .NET, Java, and Web services.

To convert from a Version 9 .pxg file to an OpenEdge Release 10 .xpxg file, see the “Converting a project file from Progress Version 9 to OpenEdge” section on page 3–11.
Specifying 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 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 the “Specifying generation settings” section on page 3–25.

Saving 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 Web service definition. Once you save the project file, use ProxyGen or Batch ProxyGen to generate. For more information on saving project files, see the “Saving the Open Client interface definition in a project file” section on page 3–24.

Generating an Open Client proxy or Web service definition

Open Client interface generation validates the information in the project file against the AppServer r-code and then generates the specified Open Client proxies and/or Web service definition. If you make subsequent changes to the AppServer 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 the “Validating and generating an Open Client proxy or Web service definition” section on page 3–36.

If this automatic update does not produce acceptable results, you must run ProxyGen and update the Open Client interface definition manually. In addition, as AppServer 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.
Running Batch ProxyGen

Batch ProxyGen runs on supported UNIX and Windows platforms. It allows you to generate Open Client proxies and/or a 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:

```
bproxygen -pxgfile project-filename.xpxg [ -useWildCard]
```

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.

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 the “Converting a project file from Progress Version 9 to OpenEdge” section on page 3–11.

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, but that do not also define .NET proxies. 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.
Converting 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.

Converting 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 Figure 3–4.

![Figure 3–4: Opening a Progress Version 9 file in OpenEdge](image)

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.
Converting using the PXGConvert utility

You can also convert a Progress Version 9 project file using the PXGConvert command-line utility.

To run PXGConvert, enter the following command at the Windows or UNIX command prompt:

```
bpxgconvert -pxgfile Version9-project-filename.pxdg
    [-pxgfile project-filename.xdg]
```

Where:

```
Version9-project-filename
```

   The name and path of the Version 9 project file containing the Open Client interface.

```
xpxgfile project-filename.xdg
```

   Optionally specifies the new project file name and path. If this is not specified, the new project file is saved to the same location as `Version9-project-filename`, with the same root name, and the `.xdg` extension.

Converting a preferences file from Progress Version 9 to OpenEdge

The format for the preferences file saved by the Open Client Toolkit in Progress Version 9 has converted to XML in OpenEdge. You can convert a Version 9 preferences file using the PrefConvert command line utility.

To run PrefConvert, enter the following command at the Windows or UNIX command prompt:

```
bprefconvert Version9-preferences-filename
```

Where:

```
Version9-preferences-filename
```

   The name and path of the Version 9 preferences file.

The new preferences file is saved to the same location as `Version9-preferences-filename`, with the same root name, and the `.xml` extension.
Defining an Open Client interface using ProxyGen

This section describes how to define an Open Client interface.

To define an AppServer application service as an Open Client proxy or Web service definition in ProxyGen:

1. Define an AppObject for the Open Client interface, and optionally define one or more SubAppObjects.
2. Select ABL procedures (non-persistent and persistent) 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:

- Defining an AppObject or SubAppObject
- Defining procedures in AppObjects and SubAppObjects
- Saving the Open Client interface definition in a project file
- Specifying generation settings
- Validating and generating an Open Client proxy or Web service definition
- Object naming in ProxyGen
- Example

For specific information on ProxyGen options for defining and generating Open Client interfaces, refer to the ProxyGen online help.
**Defining 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 AppServer 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 the “Object naming in ProxyGen” section on page 3–41.

**Creating 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.

**Specifying 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.

- The **PROPATH** (**Propath Components** list box) is set to the locations of AppServer procedures for this Open Client interface object. (See the “Defining procedures in AppObjects and SubAppObjects” section on page 3–17.)

Optional, 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 the “Handling the Unknown value (?)” section on page 3–15.

**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.
Defining an AppObject or SubAppObject

- 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 the “**Mapping temp-table parameters in Java Open Clients**” section on page 3–15.

- 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 the “**Adding ProDataSet before-image data in Web services**” section on page 3–16.

**Handling 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.

**Note:** Web services can always pass the ABL Unknown value (?), regardless of the Open Client interface settings, because Web service parameters are all nillable and allow "nil" to be passed as a representation of the ABL Unknown value (?)

**Mapping 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*.

**Adding ProDataSet before-image data in Web services**

You can specify whether a 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.

**Specifying AppServer procedure files**

You must specify any AppServer 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 AppServer procedure files, see the “Defining procedures in AppObjects and SubAppObjects” section on page 3–17.
Defining 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 the “Setting PROPATH components” section on page 3–17.

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). See the “Customizing method and ProcObject definitions (optional)” section on page 3–20.

4. Optionally, only for ProcObjects, customize the method definitions for all specified internal procedures and user-defined functions. See the “Customizing method definitions in a ProcObject (optional)” section on page 3–22.

The following sections provide information about:

- Setting PROPATH components
- Adding and deleting procedure files
- Changing the procedure execution mode (type)
- Customizing procedure access
- Customizing method and ProcObject definitions (optional)
- Customizing method definitions in a ProcObject (optional)
- Changing the PROPATH setting for existing procedures

Setting 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 AppServer 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 AppServer PROPATH. You also can use dot (.) or some other relative path as a PROPATH component.

### Adding and deleting 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:

- **For non-persistent execution** — Choose Procedure → Add → Non-persistent from the menu bar
- **For persistent execution** — Choose Procedure → Add → Persistent from the menu bar

These choices display either the Add Non-persistent Procedures dialog or the Add Persistent Procedures dialog for your current Open Client interface object. The user interface is almost identical for the two dialog. Both dialog 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.

The difference between the two dialog is in the execution type of the procedure file that is added. For each non-persistent procedure, ProxyGen generates a method in the Open Client interface. For each persistent 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 *OpenEdge Development: .NET Open Clients*, *OpenEdge Development: Java Open Clients*, or *OpenEdge Development: Web Services*. 
After selecting and adding the respective procedure files, any non-persistent procedures appear in the **Procedures** list, and any persistent procedures appear in the **Persistent Procedures** 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.

**Changing 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 folder. This folder allows you to select and move procedures between non-persistent execution (**Non-persistent Procedures** list) and persistent execution (**Persistent Procedures** list).

**To move procedures between modes:**

1. Select one or more procedures in one list.

2. Choose the **Move to Persistent** or **Move to Non-Persistent** button or the **Procedure** → **Change Type** from the menu bar, to move the selected procedures to the opposite list.

**Note:** If you choose this function for a persistent procedure, you lose any customizations to internal procedures and user-defined functions that you applied using the **Customize** function.

**Customizing procedure access**

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

1. The AppServer 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. For more information, see the “Customizing method and ProcObject definitions (optional)” section on page 3–20.

3. The AppServer developer can use the business logic on the AppServer 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 AppServer applications can use the **EXPORT** statement to control remote procedure access at run time, see *OpenEdge Application Server: Developing AppServer Applications*.
Customizing 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 the corresponding Customize Non-Persistent Procedure or Customize Persistent Procedure dialog for your Open Client interface object. Both dialog provide:

- 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 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 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 the “Specifying AppObject and SubAppObject definitions” section on page 3–14.

• The **Use AppObject Before-Image Setting** check box (checked by default) specifies whether to include ProDataSet before-image data in a Web service. For more information, see the “Specifying AppObject and SubAppObject definitions” section on page 3–14.

• 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 ProDataGraph objects to map temp-table parameters for Java Open Clients. For more information, see the “Specifying AppObject and SubAppObject definitions” section on page 3–14. 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 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), 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)
Customizing 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, 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 or the Customize User Defined Function dialog appears, depending on the selection. This dialog 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 Web service. For more information, see the “Specifying AppObject and SubAppObject definitions” section on page 3–14.
• 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 ProDataGraph objects to map temp-table parameters for Java Open Clients. For more information, see the “Specifying AppObject and SubAppObject definitions” section on page 3–14. 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 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 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.

### Changing 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 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, refer to the online help.
Saving 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 AppServer procedures and any customizations you applied to those procedures, for each Open Client interface object definition.

Setting 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 the “Using ProxyGen execution options” section on page 3–6.

ProxyGen allows you to specify a separate directory to save generated Open Client proxies, a Web service definition, and log file. For details, see the “Specifying generation settings” section on page 3–25.

Saving 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 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 opens to the last directory you selected. ProxyGen automatically saves the project file to the current work directory each time you validate or generate.
Specifying 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 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 change the default preferences for creating a new Open Client interface, use the Preferences dialog (choose Options → Preferences from the menu bar).

  These options specify such information as an application service name for the AppServer 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 change the preferences for the current Open Client interface before generation, use the Generate dialog (choose File → Generate or the Generate toolbar button).

Both the Preferences and the Generate dialog set generation preferences in a tab folder with these tabs:

- General tab
- .NET tab
- Java tab
- Web Services tab
- Sonic Web Service tab
- Sonic Native tab

You can specify a local preferences file in the properties of the ProxyGen icon. See the “Using ProxyGen execution options” section on page 3–6.

General tab

The General tab has the following fields for choosing the clients you want to generate and specifies output settings common to all of them:

- Client Proxy / Service Definition to Generate — Select the desired type of client:
  - .NET
  - Java
  - Web Services
  - Sonic Web Services
  - Sonic Native Invocation
• **AppService** — This field allows you to specify an application service name that relies on an OpenEdge NameServer to locate an appropriate AppServer.

**AppService** can contain the name of an application service known to the NameServer or AppServer being accessed through the interface. To use the name of the AppObject in the **AppService** field, select the **Use Default** check box.

• **Output Dir** — This field specifies an output directory where ProxyGen places output files from generation, including the proxy files (.class and .dll), .NET runtime assemblies (.dll, for .NET only) Web service definition files (.wsm and .wsdl), and the activity log (AppObject.log).

To make the output directory more portable, you can use dot (.) or some other relative path as your **Output Dir** 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, 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 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 AppServer 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.

• **Verbose Logging** — Select this check box to have detailed information output to the log file.
• **Session Model** — You must set this field according to the operating mode of all AppServers that support this service interface:

  – **Managed** — The AppServers are configured for the stateless, state-aware, or state-reset operating mode.

  – **Free** — The AppServers are configured for the state-free operating mode. This is the default.

  **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*.]

**.NET tab**

The .NET tab is enabled if .NET is selected in the Client Proxy/Service Definition to Generate field on the General tab.

The .NET tab has the following fields for setting .NET proxy options:

• **Namespace** — Enter the namespace to use as the root for generating the classes in a .NET proxy.

• **DataSet Namespace** — By default, ProxyGen creates the namespace of the strongly typed DataSet or DataTable class by appending “StrongTypedNS” to the project’s Namespace. To specify a different namespace for the DataSet or DataTable class, uncheck the Use Default option.

• **Append Namespace Directories to Output Dir** — Check 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).

• **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 which allows you to set compiler and XSD settings for the .NET client proxy. For a detailed description, see the “Advanced .NET Options dialog” section on page 3–29.
- **AssemblyInfo** — General information about a .NET assembly is controlled through this set of attributes: **Title**, **Version**, **Description**, **Company**, **Product**, **Runtime**, **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 form:

\[\text{major.minor.build.revision}\]

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 \textit{build})
- **1.2.3.*** (accept default value for \textit{revision})
- **1.2.*** (accept default values for \textit{build} and \textit{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 “Open Client interface generation” section on page 3–37.

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.
Advanced .NET Options dialog

You access the Advanced .NET Options dialog from the .NET tab by clicking the Advanced button. This dialog 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 tab

The Java tab is enabled if Java is selected in the Client Proxy / Service Definition to Generate field on the General tab. The Java tab has the following fields for setting Java proxy options:

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

- **Compiler** — These radio buttons allow you to specify one of the following compilers to use for generating a Java proxy:

  - Default javac — ProxyGen uses the JavaSoft javac compiler installed by OpenEdge.

  - Custom — You specify 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.

**Caution:** The Classpath includes the ABL class libraries `progress.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.

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

For directory paths, symbolic references (indicated by <> ) often are 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>
<thead>
<tr>
<th>Symbolic Reference</th>
<th>Windows:</th>
<th>UNIX:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Default classes.zip&gt;</td>
<td><code>$DLC\jdk\lib\classes.zip</code></td>
<td><code>$JDKCP</code></td>
</tr>
<tr>
<td>&lt;Install Dir&gt;</td>
<td><code>$DLC</code></td>
<td><code>$DLC</code></td>
</tr>
<tr>
<td>&lt;Proxy Dir&gt;</td>
<td>The <strong>Output Dir</strong> value specified on the <strong>Generate</strong> dialog of ProxyGen. Progress Software Corporation recommends you always leave this symbolic reference and change the <strong>Output Dir</strong> value on the <strong>Generate</strong> dialog to alter the proxy output directory.</td>
<td></td>
</tr>
</tbody>
</table>


Web Services tab

The Web Services tab is enabled if Web Services is selected in the Client Proxy/Service Definition to Generate field on the General tab.

The Web Services tab has the following fields for setting default deployment information for a Web service definition. You also can change these values during Web service deployment.

- **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 in the Web Services tab and the Namespace field in the Sonic Web Service tab are linked. If you change the Namespace in one tab, it is automatically updated in the other tab.

- **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.
Generating Proxies and Web Service Definitions

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 Explorer or Progress 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 (Output Dir on the General tab) to avoid overwriting any WSDL file already generated for this Web service.

**WSDL Settings** — These options specify the content and formatting of the generated WSM file and optional test WSDL file:

- **PortType/Binding Suffix** — 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).

- **Service Suffix** — 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).

- **Generate Test WSDL** — If you check this box, ProxyGen generates a WSDL file with the specified style/use.
Sonic Web Service tab

The Sonic Web Service tab is enabled if Sonic Web Service is selected in the Client Proxy/Service Definition to Generate field on the General tab.

The Sonic Web Service tab has the following fields for setting default deployment information for a Sonic Web Service definition (you also can change these values during the Sonic ESB container deployment):

- **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 in the Sonic Web Service tab and the Namespace field in the Web Services tab are linked. If you change the Namespace in one tab, it is automatically updated in the other tab.

- **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.

- **AppServer 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.
  - **Resource Dir** — The path in SonicFS where you want to store the WSM and WSDL files.
– **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.

- **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.

- **Advanced** — Click this button to access the Advanced Sonic ESB Options dialog which allows you to specify Sonic Domain Manager settings. For a detailed description, see the “Advanced Sonic ESB Options dialog” section on page 3–34.

### Advanced Sonic ESB Options dialog

You access the Advanced Sonic ESB Options dialog from the Sonic Web Service tab by clicking the Advanced button. This dialog 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 (*).
Sonic Native tab

The Sonic Native tab is enabled if Sonic Native Invocation is selected in the Client Proxy/Service Definition to Generate field on the General tab.

Sonic Native builds a script file (.esboe file) for each specified procedure or function that can be imported directly into the Sonic Workbench. When a Sonic ESB process is created using the Native Invocation methodology, ABL procedures are called directly via an OpenAPI call to an OpenEdge Application Server. For more information on the Sonic Native Invocation methodology, see OpenEdge Development: Messaging and ESB.

The Sonic Native tab has the following fields for setting deployment options for a Sonic Native Invocation definition (you also can change the following values during the Sonic ESB container deployment):

- **Save to Output Directory** — Specify this option to have your .esboe files written to the directory specified on the General tab.

- **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 which allows you to specify Sonic Domain Manager settings. For a detailed description, see the “Advanced Sonic ESB Options dialog” section on page 3–34.
Validating and generating an Open Client proxy or Web service definition

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, which displays the Generate dialog, and then choose the OK button.

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 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**
- **Open Client interface generation**

**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.
Validating and generating an Open Client proxy or Web service definition

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 Web service definition can be generated. The Java proxy includes a set of .java and .class files. The .NET proxy is a single assembly (.dll file) containing the classes. The Web service definition includes a .wsm file and (optionally) a .wsdl file. In all cases, the activity log records any significant generation events and errors.
Generating .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** 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-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 the “.NET tab” section on page 3–27), .NET proxy generation causes the corresponding Open Client Runtime assemblies to be copied to the output directory (Output Dir on the General tab) for ease of deployment.

ProxyGen generation performs the following actions, writing any output to the output directory that you specify in the generate preferences (Output Dir on the General tab):

• Saves the Open Client interface definition to the project file

• Performs Open Client interface validation

• Generates a Java proxy, .NET proxy, and/or Web service definition, as specified

• For .NET proxies, provides a copy of the selected Open Client Runtime assemblies

For information on the generated proxy for a Java client, see *OpenEdge Development: Java Open Clients*. For information on the generated proxy for a .NET client, see *OpenEdge Development: .NET Open Clients*. For information on the generated Web service definition, see *OpenEdge Development: Web Services*. 

3–39
Supporting localized messages for .NET client interfaces

When ProxyGen builds the .NET proxy, it copies the required .NET runtime assemblies to the proxy generation output directory. For the message assembly, only the resources for the language matching the current local language are copied to the output directory. If you need to support a different language, you must manually copy some additional files to the output directory.

To support an alternative language or more than one language, you must copy the language resources from one of the following locations depending on your generation option:

- **Signed-only Open Client Runtime assemblies** — Copy the resource for the languages you want to support from the following directory:

  ```
  OpenEdge-install-directory\dotnet\deploy\signed
  ```

- **Strong-named Open Client Runtime assemblies** — Copy the resource for the languages you want to support from the following directory:

  ```
  OpenEdge-install-directory\dotnet\deploy\strongnamed-signed
  ```

- **Strong-named-unsigned Open Client Runtime assemblies** — Copy the resource for the languages you want to support from the following directory:

  ```
  OpenEdge-install-directory\dotnet\deploy\strongnamed
  ```

For example, if you want to support a French deployment for a signed-only environment, copy the `fr` directory from `OpenEdge-install-directory\dotnet\deploy\signed` to your proxy generation output directory.
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 the “Automatic name conversions” section on page 3–42.

  Each object must be uniquely named.

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

  For 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 AppServer procedure** — The method name is the procedure filename without the extension. Method names are case sensitive. ProxyGen also performs any automatic name conversions. See the “Automatic name conversions” section on page 3–42.

- **Methods in a ProcObject that execute an internal procedure or user-defined function defined in the persistent 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 the “Automatic name conversions” section on page 3–42.
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 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
- **Procedures** — Inventory\InvAdd.r
- **Persistent Procedures** — Inventory\p-Cust.r, 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
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.
All Open Clients use equivalent mechanisms to access an AppServer application service. In general, accessing an AppServer using an Open Client interface follows this process:

1. Connect to an AppServer that supports the required application service, by instantiating an AppObject (and calling connect for 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 AppServer.
4. Disconnect from the AppServer when it is no longer needed.

This chapter describes how to perform these tasks, the common features of the mechanisms and how they differ, in the following sections:

- Connecting to an AppServer
- Getting user input for run-time connection information
- Understanding proxy and Web service object methods
- Passing parameters
- Accessing Open Client run-time properties
- Handling errors in an Open Client application
- Accessing an AppServer directly without proxies

For more detailed information about specific clients, see OpenEdge Development: Java Open Clients, OpenEdge Development: .NET Open Clients, and OpenEdge Development: Web Services.
Connecting to an AppServer

To specify an AppServer 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 the “Accessing Open Client run-time properties” section on page 4–10. 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 AppServer operating mode configuration.

- For session-managed 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*.

For more information on session models and how they affect Open Client applications, see Chapter 1, “Overview.”

Types of connections

Open Clients support two types of AppServer connections, depending on the type of Open Client:

- Internet connections
- Intranet connections

**Internet connections**

.NET and Java Open Clients can access an AppServer over the Internet using the AppServer Internet Adapter (AIA), which is a Java servlet running on a Web server that converts HTTP requests from Open Clients into direct requests from the AIA to the AppServer and returns the results back to the Open Client from the AppServer. As such, the AIA allows any AppServer client to access the AppServer from anywhere on the Internet.

Web service clients access the AppServer over the Internet using the Web Services Adapter (WSA), which is also a Java servlet running on a Web server that converts HTTP requests from Web service clients into direct requests from the WSA to the AppServer and returns the results back to the Web service client from the AppServer.
The difference between .NET/Java Open Clients and Web service clients accessing the AppServer over the Internet is in how the AppServer interface appears to the client. For .NET/Java Open Clients, the interface maps in “native mode” directly to the application service supported by the AppServer. For Web service clients, the interface maps to an industry-standard Web service interface, which in turn maps directly to the application service supported by the AppServer. Each type of client connects to the same functionality, regardless of the type of interface.

Intranet connections

Only .NET and Java Open Clients can access an AppServer over an intranet, using an OpenEdge®-proprietary AppServer protocol to transport requests. For this type of connection, the Open Client interface maps to the application service in a “native mode” supported by the Open Client proxy using a connection that directly accesses the AppServer. As such, the interface to an AppServer, whether it is over the Internet or over the intranet, appears identical to the Open Client; only the protocol used to make the connection is different between the two.

Note: The performance of Internet connections to the AppServer are inherently slower than for intranet connections because of the Web server needed to support the Internet.

Secure connections

Open Clients support two types of secure connections to an AppServer, depending on the type of Open Client:

- Secure Internet connections using the AppServer Internet Adapter (AIA) or Web Services Adapter (WSA)
- Secure intranet connections to an SSL-enabled AppServer

The following sections briefly describe how secure AppServer connections work with Open Clients. For more information on AppServer support for secure connections, see OpenEdge Application Server: Developing AppServer Applications. For more information on support for secure connections in OpenEdge, see OpenEdge Getting Started: Core Business Services.

Secure Internet connections

.NET/Java Open Clients and Web service clients can access the AppServer over secure Internet connections using HTTPS. The Secure Sockets Layer (SSL) provides the security by tunneling HTTP requests, with data privacy provided by symmetric key encryption and authentication between the client and server (in this case the Web server) provided using public key (digital) certificates. The requirement for using HTTPS connections is to have access to a root digital certificate that corresponds to a certificate that is stored on the server. For more information on managing root digital certificates for Open Clients, see Chapter 2, “Configuration.”

Secure Internet connections do not by themselves secure the entire route to the AppServer. OpenEdge can secure the connection beyond the Web server by supporting SSL connections from the AIA and WSA to an SSL-enabled AppServer over the intranet. For more information on AIA and WSA connections to an SSL-enabled AppServer, see OpenEdge Application Server: Developing AppServer Applications.
Secure intranet connections

.NET and Java Open Clients can access the AppServer over secure intranet connections using OpenEdge SSL. SSL provides the security by tunneling AppServer protocol requests, where data privacy is provided by symmetric key encryption and authentication between the client and server (in this case the SSL-enabled AppServer) is provided by public key (digital) certificates. The requirement for using SSL-enabled AppServer connections is that the client must have access to a root digital certificate that corresponds to a digital certificate stored on the server. For more information on managing root digital certificates for Open Clients, see Chapter 2, “Configuration.”

A secure intranet connection to an SSL-enabled AppServer from an Open Client secures the entire connection between the client and the AppServer.

Specifying the AppServer connection

The connection information includes character-string parameters that are passed to the AppServer. Typically you pass the following parameters:

- **URL** — AppServer connection information, depending on the connection type:
  - **Intranet connection (Java and .NET only)** — AppServer protocol or SSL-enabled AppServer protocol for a secure intranet connection, specified using an AppServer URL that identifies the application service
  - **Internet connection (Java and .NET only using the AIA)** — HTTP protocol or HTTPS protocol for a secure Internet connection, specified using an AppServer URL that identifies the application service
  - **Internet connection (Web services only, using a WSA)** — HTTP protocol or HTTPS protocol for a secure Internet connection, specified using a Web service URL that identifies the Web service and that is specified during deployment of the Web service to the WSA

- **User-ID and password** — AppServer user identification and password

- **AppServer Information** — Application-specific information

In general, all connection parameters are optional, depending on the AppServer and Web service configuration and application requirements. For any parameter you do not specify, OpenEdge uses standard default values; however, you must ensure any defaults are valid for your application environment. For more information on the AppServer URL format, see the information on connecting to an AppServer using a URL in *OpenEdge Application Server: Developing AppServer Applications*. For more information on Web service URLs, see *OpenEdge Development: Web Services*.

Depending on the connection protocol, Open Client type, and various network conditions, your application might need to obtain information from the user in order to complete a connection. The following section describes some of these requirements for user information.
Getting 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 “Accessing Open Client run-time properties” section on page 4–10.

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 an SSL intranet connection, to authenticate the identity of an AppServer. Every Web server that hosts the AIA and supports the HTTPS protocol and every SSL-enabled AppServer 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 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 AppServer 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: .NET Open Clients, 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.
Understanding proxy and Web service object methods

Proxy objects provide methods that allow you to manage proxy objects, create other proxy objects, and access AppServer procedures. ProxyGen generates proxy objects for Java and .NET. The actual proxies for a 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 AppServer.
  
  For more information about connecting to an AppServer, see the “Connecting to an AppServer” section on page 4–2. For more information about establishing a connection for a specific type of Open Client, see *OpenEdge Development: Java Open Clients*, *OpenEdge Development: .NET Open Clients*, or *OpenEdge Development: Web Services*.

- **Common methods** — Manages a proxy object and its AppServer connection. The common methods available depend on the client type. For details, see *OpenEdge Development: Java Open Clients*, *OpenEdge Development: .NET Open Clients*, or *OpenEdge Development: Web Services*.

- **Class factory methods** — Allows you to create SubAppObjects and ProcObjects. Executing a SubAppObject class factory method does not access the AppServer. Executing a ProcObject class factory method executes the corresponding persistent procedure on the AppServer.

  ProxyGen generates class factory methods using standard naming conventions for each type of Open Client. For more information, see Chapter 3, “Generating Proxies and Web Service Definitions.”

  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 AppServer. For more information, see the “Remote ABL methods” section on page 4–7.

Table 4–1 shows which types of methods are available for each type of proxy object.

### Table 4–1: Types of methods available for different proxy objects

<table>
<thead>
<tr>
<th>Type of method</th>
<th>AppObject</th>
<th>SubAppObject</th>
<th>ProcObject</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>X</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Common</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Class factory:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SubAppObject class factory</td>
<td>X</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>ProcObject class factory</td>
<td>X</td>
<td>X</td>
<td>–</td>
</tr>
<tr>
<td>SDOResultSet class factory</td>
<td>X</td>
<td>X</td>
<td>–</td>
</tr>
<tr>
<td>Remote ABL</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Remote ABL methods

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

Note: If the ABL persistent 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 OpenEdge Application Server: Developing AppServer Applications.

Identifying generated methods

ProxyGen generates method names using automatic conversions and conventions. For more information, see Chapter 3, “Generating Proxies and Web Service Definitions.” 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.

Passing 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 OpenEdge Development: Java Open Clients, OpenEdge Development: .NET Open Clients, and OpenEdge Development: Web Services.

Handling 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:

<table>
<thead>
<tr>
<th>Java method</th>
<th>_getProcReturnString</th>
</tr>
</thead>
<tbody>
<tr>
<td>.NET method</td>
<td>_GetProcReturnString</td>
</tr>
</tbody>
</table>

This method returns the current value of RETURN-VALUE as set by the RETURN statement most recently executed on the AppServer 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 OpenEdge Development: ABL Reference, and for how they function in an AppServer application, see OpenEdge Application Server: Developing AppServer 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: Error Handling.*
Passing parameters

The Open Client passes parameters to the AppServer 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 OpenEdge Development: Java Open Clients, OpenEdge Development: .NET Open Clients, and OpenEdge Development: Web Services.
Accessing 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 OpenEdge Development: Java Open Clients, OpenEdge Development: .NET Open Clients, and OpenEdge Development: Web Services.
Handling errors in an Open Client application

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

- **Client-side errors**
- **Server-side errors before remote ABL method execution**
- **Server-side errors during remote ABL execution**

A client-side error is an error that occurs on the client machine before the client sends a particular request to the AppServer. A server-side error is an error that occurs on the AppServer machine, either before the AppServer 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 OpenEdge Development: Java Open Clients, OpenEdge Development: .NET Open Clients, and OpenEdge Development: 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. 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 AppServer, the request is cancelled with no effect on the AppServer. In general, client-side errors have no effect on the AppServer to which the client is connected. However, the AppServer 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 AppServer 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 Web server, or the AIA is not installed and 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 AppServer 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 AppServer writes error messages for these conditions to the server log file.

- **STOP condition** — Any unhandled STOP condition raised on the AppServer 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 AppServer 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.
Accessing an AppServer directly without proxies

For .NET and Java Open Clients, OpenEdge provides a means for you to access application services directly on an AppServer 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, 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 AppServer using the OpenAPI. For more information, see Chapter 1, “Overview” and previous sections in this chapter. However, if you have a basic understanding of how ABL provides application services on the AppServer 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 basic information on ABL, see the sections on ABL procedures in Chapter 1, “Overview.” For information on the AppServer, see OpenEdge Application Server: Developing AppServer Applications.

Table 4–2 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 4–2: OpenAPI Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenAppObject</td>
<td>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.</td>
</tr>
<tr>
<td>OpenProcObject</td>
<td>For instantiating and accessing a remote ABL persistent 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.</td>
</tr>
<tr>
<td>ParamArray</td>
<td>Array for passing parameters to an ABL procedure or user-defined function.</td>
</tr>
<tr>
<td>ParamArrayMode</td>
<td>Constants for specifying the mode of an ABL procedure or user-defined function parameter (INPUT, INPUT-OUTPUT, or OUTPUT).</td>
</tr>
<tr>
<td>Parameter</td>
<td>Constants for specifying the ABL data type of a procedure or user-defined function parameter, return type, or temp-table field.</td>
</tr>
</tbody>
</table>
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:

- *OpenEdge Development: .NET Open Clients* for the .NET OpenAPI
- *OpenEdge Development: Java Open Clients* for the Java OpenAPI
Index

A
ABL internal procedures 3–22
ABL procedures 1–8, 3–17
customizing access 3–19
ABL user-defined functions 3–22
Accessing the AppServer 1–5
Activity log file 3–36
Adding procedure files 3–18
AppObjects 1–8
  defining in ProxyGen 3–14
  Open Client interface 3–2
  specifying ABL procedures and functions 3–17
AppServer
  accessing with Open Clients 1–5
  connecting, general programming 4–2
  connections from Open Clients 4–4
  default connection values 4–4
  interface access to procedures 3–4
  network connection types 4–2
  Open Client architecture 1–2
AppServers
  connection information 4–4
Architecture, Open Client 1–2
Authentication
  server digital certificates 4–5

B
Building applications, overview 1–4

C
Class factory methods
  definition 1–10
  using in an Open Client 4–6
Client-side errors
  client-proxy interactions 4–11
  definition 4–11
Common methods 4–6
Configuration requirements 2–1
Connecting to an AppServer
  See AppServer
  Connection information
    for AppServers 4–4
Connection methods 4–6
Connections
  Internet 4–2
  intranet 4–3
  secure 4–3
  user input 4–5
Creating
  AppObjects 3–14
  ProcObjects 3–17
  SubAppObjects 3–14
Customizing access to ABL procedures 3–19
<table>
<thead>
<tr>
<th>D</th>
<th>J</th>
</tr>
</thead>
</table>
| Default values  
connecting to an AppServer 4–4 | Java Open Clients  
root digital certificate packages 2–3 |
| Defining an interface 3–13 | Java proxy  
generating 3–36  
generation rules 3–37  
temp-table parameter mapping  
AppObjects and SubAppObjects 3–15  
external procedures 3–21  
internal procedures and UDFs 3–22 |
| Deleting procedure files 3–18 | Java tab 3–29 |
| Digital certificates, See Root digital certificates |  |
| Distribution packages, Open Client Runtime 2–2 |  |

<table>
<thead>
<tr>
<th>E</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Encapsulating functionality 3–3</td>
<td></td>
</tr>
<tr>
<td>ERROR condition handling 4–12</td>
<td></td>
</tr>
</tbody>
</table>
| Error handling 4–11  
ProxyGen validation and generation 3–36 |  |

<table>
<thead>
<tr>
<th>G</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>General tab 3–25</td>
<td></td>
</tr>
<tr>
<td>Generating proxies 3–36</td>
<td></td>
</tr>
</tbody>
</table>
| Generation  
Open Client interface 3–37 |  |
| Generation rules 3–37 |  |
| Generation settings  
specifying 3–9, 3–25 |  |
| Getting started with ProxyGen 3–5 |  |

<table>
<thead>
<tr>
<th>H</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchy of proxy objects 1–10</td>
<td></td>
</tr>
</tbody>
</table>
| HTTPS  
digital certificates 2–3 |  |

<table>
<thead>
<tr>
<th>I</th>
<th></th>
</tr>
</thead>
</table>
| Internal procedures  
mapping in ProxyGen 3–22 |  |
| Internet connections 4–2  
secure 4–3 |  |
| Intranet connections 4–3  
secure 4–4 |  |

<table>
<thead>
<tr>
<th>L</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical connections 1–6</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>M</th>
<th></th>
</tr>
</thead>
</table>
| Methods  
class factory 4–6  
common 4–6  
connection 4–6  
customizing for ProcObjects 3–22  
customizing in ProxyGen 3–20  
defining in ProxyGen 3–18  
passing parameters 4–9  
remote ABL 4–6, 4–7 |  |

<table>
<thead>
<tr>
<th>N</th>
<th></th>
</tr>
</thead>
</table>
| Naming conventions  
automatic conversions 3–42  
proxy object 3–41  
restrictions and side effects 3–43 |  |
| .NET Open Client  
localized messages 3–40  
Runtime assemblies 3–38 |  |
| .NET Open Clients  
root digital certificates 2–4 |  |
| .NET proxy  
generating 3–36  
generation rules 3–38  
validation rules 3–36 |  |
| .NET tab 3–27 |  |
| Non-persistent procedures  
adding and deleting in ProxyGen 3–18  
definition 1–8 |  |
Index

O
Object model 1–7
Open Client
applications
building and running 1–4
requirements 2–2
specifying an AppServer connection 4–2
AppServer connections 4–4
architecture 1–2
generating proxies 3–36
interface 3–2
object model 1–7
Open Client interface
.NET generation 3–38
AppServer procedure access 3–4
defining 3–13
definition example 3–44
encapsulation 3–3
generating 3–9
generation 3–37
generation and validation 3–36
naming conventions 3–41
objects 3–2
saving in project file 3–9
specifying generation settings 3–9
validation 3–36
Open Client Proxy Generator, See ProxyGen
Open Client Runtime distribution packages 2–2
OpenAPI
description 4–13
supporting classes 4–13
OpenEdge installation
root digital certificates 2–3

P
Parameter passing in Open Clients 4–9
Persistent procedures
adding and deleting in ProxyGen 3–18
definition 1–8
Physical connections 1–6
PrefConvert utility 3–12
Preferences
file
conversion 3–12
ProxyGen 3–25
Procedure files
adding and deleting 3–18
methods 3–20
Procedures
customizing access 3–19
ProcObjects 1–9
compared to procedures 3–3
customizing 3–20
defining in ProxyGen 3–17
Open Client interface 3–2
ProDataSets
including before-image data in Web service 3–20
including before-image data in Web services 3–15, 3–16
Programming Open Clients
AppServer connections 4–2
error handling 4–11
error handling. See Error handling
method types 4–6
OpenAPI without proxies 4–13
passing parameters 4–9
run-time properties 4–10
Progress Version 9
preferences file conversion 3–12
project file conversion 3–11
Project
file
conversion 3–11
definition 3–8
saving 3–24
work directory 3–24
PROPATH
accessing AppServer procedures 3–4
AppObjects and SubAppObjects 3–14
changing existing setting 3–23
Proxies
adding and deleting procedures 3–18
defining 3–13
definition 1–10
generating 3–36
generation and validation 3–36
methods 4–6
naming conventions 3–41
objects 1–7
hierarchy 1–10
releasing 1–10
types 1–10
parameter passing 4–9
run-time properties 4–10
Index

ProxyGen
  .NET tab 3–27
  activity log 3–36
  basic tasks 3–6
  converting project files 3–11
  creating a ProcObject 3–17
  creating a SubAppObject 3–14
  creating an AppObject 3–14
  defining interfaces 3–13
  General tab 3–25
  generating and validating proxies 3–36
  getting started 3–5
  GUI execution options 3–6
  Java tab 3–29
  main window 3–7, 3–8
  overview 3–5
  preferences 3–25
  saving the project file 3–24
  specifying generation settings 3–25
  starting 3–7
    batch 3–10
  Web Services tab 3–31

Public key certificates, See Root digital certificates

.pxs See Project

PXGConvert utility 3–12

S

Secure connections 4–3

Secure Sockets Layer (SSL)
  digital certificates 2–3
  Internet connections 4–3
  intranet connections 4–4

Servers digital certificates, validating 4–5

Servers, digital certificates 4–5

Server-side errors
  client-server interactions 4–11
    definition 4–11

Session models 1–5
  configuration 1–5
  connection management 1–6
  programming 1–6
  specifying for Open Clients 3–4

Session-free model 1–5
  AppObjects 1–8
  ProcObjects 1–9
  SmartDataObjects 1–9
  SubAppObjects 1–8

Session-managed model 1–5
  AppObjects 1–8
  SubAppObjects 1–8

Signed-only .NET assemblies 3–38, 3–39
  localized messages 3–40

Specifying an AppServer connection
  Open Client application 4–2

SSL, See Secure Sockets Layer (SSL)

Starting
  Batch ProxyGen 3–10
  ProxyGen 3–7

STOP condition handling 4–12

Strong-named .NET assemblies 3–38
  localized messages 3–40

structured error handling 4–8

SubAppObjects 1–8
  defining in ProxyGen 3–14
    definition 1–7
  Open Client interface 3–2
  specifying ABL procedures and functions 3–17

System requirements
  building applications 2–2
  ProxyGen 2–2
T
Temp-table parameters passed in Java proxies 3–15
Thread control 4–10
Tracing 4–10
Types
  of Open Clients 1–1
  of proxy objects 1–10

U
UNIX
  Batch ProxyGen 3–10
Unknown value (?)
  enabling for function return values 3–22
  enabling for parameters and return values
    of Sub/AppObjects 3–14
  enabling for procedure parameters
    external 3–20
    internal 3–23
  example for managing 3–44
  passed as a parameter in proxy objects 3–15
  proxy validation rules 3–37
  setting for procedure file methods and
    ProcObjects 3–22
User input for connections 4–5
User-defined functions
  mapping in ProxyGen 3–22

V
Validation
  Open Client interface 3–36
  rules 3–36
  server digital certificates 4–5

W
Web service clients
  root digital certificates 2–4
Web service interface
  defining 3–13
Web services
  including ProDataSet before-image data 3–15, 3–16, 3–20
    methods 4–6
    run-time properties 4–10
Web Services tab 3–31
Windows
  Batch ProxyGen 3–10
Work directory
  project 3–24