Progress®



Develop Portable ABL Applications

OpenEdge®

Copyright

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

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

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

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

July 2021

Product version: Progress OpenEdge 12.4

Updated: 2021/06/28

Table of Contents

Naming conventions for files, tables, and variables.....9

Operating system statements.....11

Use the OPSYS function.....11
Standard system commands.....12
OS-COMMAND.....13
INPUT FROM OS-DIR.....14

Code style.....15

Preface

Purpose

OpenEdge lets you build one application that runs in multiple environments. This set of topics discuss some of the coding issues involved with writing OpenEdge applications that are portable across system environments, interfaces, and databases.

- Naming conventions for files, tables, and variables on page 9
- Operating system statements on page 11
- Code style on page 15

Audience

This guide is intended for ABL developers who are interested in writing portable applications.

Organization

Naming conventions for files, tables, and variables on page 9

Operating system statements on page 11

Code style on page 15

Documentation conventions

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

Purpose

OpenEdge lets you build one application that runs in multiple environments. This set of topics discuss some of the coding issues involved with writing OpenEdge applications that are portable across system environments, interfaces, and databases.

- Naming conventions for files, tables, and variables on page 9
- Operating system statements on page 11
- Code style on page 15

Audience

This guide is intended for ABL developers who are interested in writing portable applications.

Organization

Naming conventions for files, tables, and variables on page 9

Operating system statements on page 11

Code style on page 15

Documentation conventions

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

1

Naming conventions for files, tables, and variables

Different operating systems have different restrictions and naming conventions. To ensure portability across platforms, use the conventions described in this topic when naming your fields, files, tables, and variables.

Characters to avoid

Several characters have special meanings or are invalid in one or more supported operating systems. Avoid using these characters:

```
/ \ " ' * ; | ? [ ] ( ) ~ ! $ { } < >
```

Do not use a hyphen as the first character of a filename. Do not use spaces in filenames; spaces delimit filenames.

Case sensitivity

Although operating system filenames are not case sensitive in Windows, they are on UNIX. On systems where operating system filenames are case sensitive, filenames and command names in uppercase are different from those in lowercase. By convention, lowercase is used for most UNIX filenames.

To ensure portability among operating systems, use lowercase when specifying a procedure name in a RUN statement, and make sure your procedure files have lowercase names on UNIX.

File path specifications

You can use UNIX pathname syntax in OpenEdge procedures in Windows. OpenEdge automatically converts the UNIX pathname syntax to Windows pathname syntax. This can help maintain portability among operating systems.

Use forward slashes (/) as separators in specifying file paths when using the `RUN`, `INPUT FROM`, or `OUTPUT TO` statements. Although OpenEdge running in Windows supports both forward slashes (/) and backslashes (\), OpenEdge running on UNIX supports only forward slashes. OpenEdge automatically converts forward slashes in pathnames to backslashes. Therefore, use forward slashes for the best portability across operating systems.

OpenEdge can only reference UNIX pathnames of up to 60 characters and Windows pathnames of up to 255 characters.

Use environment variables and relative paths instead of hard coding full paths. This allows you to make global changes more easily and creates more meaningful names. This also helps OpenEdge applications appear customized for each environment.

Terminal key definitions

The operating system you port to might expect certain terminal key mappings that do not exist in the environment where you developed your application.

Alternatively, your code might rely on unique key mappings or operating-system-specific terminal control information stored in the user-interface environment files: the PROTERMCAP file on UNIX and the registry or a `progress.ini` file in Windows. For more information on environments, see *Manage ABL Applications*.

Consider the following portability issues:

- If you are planning to port your applications from one operating system to another, use the tilde (~) as an escape character in your application. While versions of OpenEdge running on UNIX interpret both the backslash (\) and the tilde as an escape character, versions of OpenEdge running in Windows interpret only the tilde as an escape character.
- Wherever possible, use spaces rather than tabs. Spaces are consistent in size regardless of the operating system, but tabs might vary.
- Use `KEYFUNCTION` in code logic, which is portable across platforms. Use `KEYLABEL` in messages to the user.

2

Operating system statements

The next set of topics explain how you can use operating system statements in portable ABL applications:

- Use the OPSYS function on page 11
- Standard system commands on page 12
- OS-COMMAND on page 13
- INPUT FROM OS-DIR on page 14

For details, see the following topics:

- Use the OPSYS function
- Standard system commands
- OS-COMMAND
- INPUT FROM OS-DIR

Use the OPSYS function

To avoid the limitations of operating-system-specific statements, use the OPSYS function to determine the appropriate operating system. The OPSYS function allows you to build one application that can run on more than one operating system. The OPSYS function identifies the operating system being used so that a single version of a procedure can work differently on different operating systems.

For example, the following procedure produces a listing of the files in your current directory. The OPSYS function determines the operating system you are running OpenEdge on and uses the appropriate operating system command to produce the directory listing:

```
CASE OPSYS:
  WHEN "unix" THEN OS-COMMAND ls.
  WHEN "win32" THEN OS-COMMAND dir.
  OTHERWISE MESSAGE OPSYS "is an unsupported operating system".
END CASE.
```

To use operating system commands from within OpenEdge, use either the operating system statement for the operating system you are running on or the OS-COMMAND statement. For example, use the OS-COMMAND statement on a Windows system and on a UNIX system. If you include the operating system statement in a procedure on a system other than the one named, the procedure compiles but does not run if the flow of control passes through that operating system statement. OpenEdge might report an error when it tries to process an operating system statement on a system other than the one named. For more information, see "OPSYS function" in the *ABL Reference* guide.

You can also use the built-in preprocessor directive, {&OPSYS}, which expands to a character string that contains the name of the operating system that the file is being compiled on.

The preprocessor is a component of the OpenEdge Compiler. You control the preprocessor by placing preprocessor directives throughout your source code.

The {&OPSYS} preprocessor directive allows you to write code that is conditionally compiled, while the OPSYS function is a run-time function. For more information on preprocessor names, see the "Preprocessor" topics in *Manage ABL Applications*.

Standard system commands

Where possible, use standard system commands instead of variants unique to certain hardware manufacturers. The following table lists the OS statements that promote portability by avoiding limitations of operating-system-specific commands. These statements allow you to create a single version of a procedure that works differently on varying operating systems.

Table 1: OS statements

OS statement	Description
OS-APPEND	Performs a system call to execute the operating system command that appends two files.
OS-COMMAND	Performs a system call to execute an operating system command that you specify.
OS-COPY	Performs a system call to execute the operating system command that copies a file.
OS-CREATE-DIR	Performs a system call to execute the operating system command that creates a new directory.
OS-DELETE	Performs a system call to execute the operating system command that deletes a file or directory. You can delete one or more files, a directory, or an entire directory branch.
OS-DRIVES	Performs a system call to execute the operating system command that returns a comma-separated list of the available drives in Windows.
OS-ERROR	Returns an OpenEdge error code that indicates whether an execution error occurred during the last OS-APPEND, OS-COMMAND, OS-COPY, OS-CREATE-DIR, OS-DELETE, or OS-RENAME statement.
OS-GETENV	Performs a system call to execute the operating system command that returns a string containing the value of the specified environment variable in the environment in which OpenEdge is running.
OS-RENAME	Performs a system call to execute the operating system command that renames a file or directory.

See *ABL Reference* for more information on these statements.

OS-COMMAND

Use the OS-COMMAND statement to execute an operating system statement that you cannot execute using the OS statements listed in Standard system commands on page 12. The OS-COMMAND statement provides a generic, operating-system-independent way to escape to the current operating system, which lets you:

- Execute a OpenEdge or operating system command that has the same syntax on two or more different operating systems.
- Start an operating system shell.
- Execute an operating system statement that a user enters.

The arguments to OS-COMMAND must be appropriate for the current operating system. Therefore, where possible, read these arguments at run time from the user, database table, or environment variables rather than hard coding them. The following procedure prompts the user for an operating system command, then uses the OS-COMMAND statement to execute the command:

```
DEFINE VARIABLE comm-line AS CHARACTER FORMAT "x(70)".
REPEAT:
     UPDATE comm-line.
     OS-COMMAND VALUE(comm-line).
END.
```

The OS-COMMAND statement eliminates the need to use the OPSYS function to determine the operating system. However, if you cannot use the OS-COMMAND statement, use the OPSYS function to determine the operating system you are running on, and use conditional logic to execute the appropriate code using one of the operating-system-specific escape statements.

The NO-WAIT option of OS-COMMAND is valid only in multi-tasking environments. This option causes OpenEdge to pass control to the statement following the OS-COMMAND, without waiting for the operating system command to terminate. If you are using the OS-COMMAND statement to run an independent Windows application, use the NO-WAIT option.

For more information, see the OS-COMMAND Statement reference entry in *ABL Reference*.

INPUT FROM OS-DIR

You generally use the INPUT FROM statement to read the contents of an operating system file; however, you can also read a list of the files in a directory using the OS-DIR option of the INPUT FROM statement. The INPUT FROM statement specifies a new input source. Using INPUT FROM OS-DIR indicates that you want your input to be the filenames found in the directory you specify. If that directory is not a valid directory or you do not have permission to read it, an error condition occurs. Otherwise, OpenEdge generates the directory list and sends it to the calling program through the INPUT stream. An INPUT CLOSE statement discards any unread filenames from the list.

The following example uses the OS-GETENV function to find the path of the DLC directory, then uses the OS-DIR option of INPUT FROM to read the contents of the directory:

```
DEFINE VARIABLE search-dir AS CHARACTER.
DEFINE VARIABLE file-name AS CHARACTER FORMAT "x(25)" LABEL "File".
DEFINE VARIABLE attr-list AS CHARACTER FORMAT "x(4)" LABEL "Attributes".
search-dir = OS-GETENV("DLC").
INPUT FROM OS-DIR(search-dir).
REPEAT:
   SET file-name ^ attr-list
      WITH WIDTH 70 USE-TEXT TITLE "Contents of " + search-dir.
END.
INPUT CLOSE.
```

When you use the OS-DIR option, the UNBUFFERED option is ignored. OS-DIR always buffers exactly one filename at a time. When you try to read beyond the last filename in the list, OpenEdge generates the ENDKEY condition.For more information, see "INPUT FROM statement" in the *ABL Reference*.

3

Code style

This topic describes two techniques to isolate your user-interface code:

- Use preprocessor directives
- Use the VIEW-AS phrase

Use preprocessor directives

OpenEdge provides a language preprocessor that allows you to write applications that are easy to read, modify, and transport to other operating systems. The preprocessor is a component of the OpenEdge Compiler. Before the Compiler analyzes your source code and creates r-code, the preprocessor examines your source code and performs text substitutions.

You control the preprocessor by placing preprocessor directives throughout your source code. A preprocessor directive is a statement that begins with an ampersand (&) and is meaningful only to the preprocessor.

The preprocessor recognizes a built-in preprocessor constant that identifies the window system where a file is being compiled, called &WINDOW-SYSTEM. The possible values include "MS-WIN*nn*" and "TTY." You can use this constant to direct the preprocessor to determine which code to run on each window system, as follows:

```
FORM
  fld1
  fld2
&IF "{&WINDOW-SYSTEM}" = "MS-WIN97" &THEN
  fld3 AT 20
&ELSE
  fld3 AT 15
&ENDIF
   WITH FRAME XYZ.
```

The SESSION system handle also has a WINDOW-SYSTEM attribute. Your application can use this to test the current window system while it is running:

```
IF SESSION:WINDOW-SYSTEM = "MS-WIN97" THEN
   .
   .
   .
```

Both the &WINDOW-SYSTEM preprocessor constant and the SESSION handle WINDOW-SYSTEM attribute perform the same basic function. The &WINDOW-SYSTEM preprocessor constant allows you to write code that is conditionally compiled, while the WINDOW-SYSTEM attribute is a run-time function.

The WINDOW-SYSTEM attribute evaluates as follows:

- If Windows nn is running, and the Windows nn user interface is running, this attribute evaluates to MS-WINnn. Otherwise, if the Windows nn user interface is not running, it evaluates to MS-WINDOWS.
- If the application is not running in a Windows environment, this attribute evaluates to TTY.

OpenEdge supports an override option that enables applications that require a WINDOW-SYSTEM attribute to return the value of MS-WINDOWS for all Microsoft operating systems. To establish this override capability, define the Window System key in the Startup Section of the current environment, which might be in the registry or an initialization file. If the Window System key is located, the WINDOW-SYSTEM attribute returns the value associated with the Window System key on all platforms.

You can also specify offsets and ROW and COLUMN specifications using preprocessor constants, then define the constants separately for different environments. OpenEdge allows you to specify fractional character units so that you can specify precise locations for objects in a graphical environment. In a character environment, the ROW and COLUMN values are truncated to integer values, as follows:

```
&IF "{&WINDOW-SYSTEM}" = "TTY" &THEN
&GLOBAL-DEFINE COL3 20
&ELSE
&GLOBAL-DEFINE COL3 15
&ENDIF
FORM
  fld1
  fld2
  fld3 AT {&COL3}
  WITH FRAME XYZ.
```

Use the VIEW-AS phrase

If possible, use the VIEW-AS phrase with the DEFINE VARIABLE statement instead of with screen I/O statements. The VIEW-AS phrase describes how a field or variable is represented on the screen. You can use the VIEW-AS phrase as a modifier to the DEFINE VARIABLE statement or as a modifier to a screen I/O statement, such as DISPLAY.

With the DEFINE VARIABLE statement, the VIEW-AS phrase specifies the default type of widget for the variable being defined. Use the VIEW-AS phrase when defining a variable so that code that manipulates the variable, such as an UPDATE statement, does not have to specify the representation. This helps you isolate your user-interface code.

For more information, see "VIEW-AS phrase" in the *ABL Reference*.