Progress®



# Develop Portable ABL Applications

OpenEdge®

# Copyright

© **2019 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.

Corticon, DataDirect (and design), DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Deliver More Than Expected, Icenium, Kendo UI, Kinvey, NativeScript, OpenEdge, Powered by Progress, Progress, Progress Software Developers Network, Rollbase, SequeLink, Sitefinity (and Design), Sitefinity, SpeedScript, Stylus Studio, TeamPulse, Telerik, Telerik (and Design), Test Studio, and WebSpeed 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, DataDirect Autonomous REST Connector, DataDirect Spy, SupportLink, DevCraft, Fiddler, JustAssembly, JustDecompile, JustMock, Kinvey Chat, NativeChat, NativeScript Sidekick, OpenAccess, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, 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.

**February 2019**

**Last updated with new content:** Release 10.1A

**Updated: 2019/03/01**

# Table of Contents


**Preface....7**

Purpose....7
Using ABL documentation....8
References to ABL compiler and run-time features....8
References to ABL data types....8
Typographical conventions....8
Examples of syntax descriptions....10
Long syntax descriptions split across lines....11
Complex syntax descriptions with both required and optional elements....12
OpenEdge messages....12
Obtaining more information about OpenEdge messages....13

**Chapter 1: Naming conventions for files, tables, and variables....15**

Characters to avoid....15
Case sensitivity....16
File path specifications....16
Terminal key definitions....16

**Chapter 2: Operating system statements....17**

Using the OPSYS function....17
Standard system commands....18
OS-COMMAND....19
INPUT FROM OS-DIR....20

**Chapter 3: Coding style....21**

Using preprocessor directives....21
Using the VIEW-AS phrase....23

# Preface

For details, see the following topics:

- Purpose
- Using ABL documentation
- Typographical conventions
- Examples of syntax descriptions
- OpenEdge messages

## Purpose

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

This web paper covers the following topics:


- Naming conventions for files, tables, and variables on page 15
- Operating system statements on page 17
- Coding style on page 21

# Using ABL documentation

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 updates go to https://docs.progress.com.

## 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 documentation uses the following typographical and syntax conventions:

| Convention | Description |
|---|---|
| **Bold** | Bold typeface indicates commands or characters the user types, provides emphasis, or the names of user interface elements. |
| *Italic* | Italic typeface indicates the title of a document, or signifies new terms. |
| **SMALL, BOLD CAPITAL LETTERS** | Small, bold capital letters indicate OpenEdge key functions and generic keyboard keys; for example, **GET** and **CTRL**. |
| **KEY1+KEY2** | 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**. |
| **KEY1 KEY2** | 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**. |
| **Syntax:** | |
| `Fixed width` | A fixed-width font is used in syntax, code examples, system output, and file names. |
| *`Fixed-width italics`* | Fixed-width italics indicate variables in syntax. |
| ***`Fixed-width bold`*** | Fixed-width bold italic indicates variables in syntax with special emphasis. |
| `UPPERCASE fixed width` | ABL keywords in syntax and code examples are almost always shown in upper case. Although shown in uppercase, you can type ABL keywords in either uppercase or lowercase in a procedure or class. |
| Period (.) or colon (:) | 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. |
| **[ ]** | Large brackets indicate the items within them are optional. |
| [ ] | Small brackets are part of ABL. |
| **{ }** | Large braces indicate the items within them are required. They are used to simplify complex syntax diagrams. |
| { } | Small braces are part of ABL. For example, a called external procedure must use braces when referencing arguments passed by a calling procedure. |
| **\|** | A vertical bar indicates a choice. |
| **. . .** | Ellipses indicate repetition: you can choose one or more of the preceding items. |

# Examples of syntax descriptions

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

### Syntax

```
ACCUM aggregate expression
```

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

```
FOR EACH Customer NO-LOCK:
  DISPLAY Customer.Name.
END.
```

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

### Syntax

```
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

```
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

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

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

### Syntax

```
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**.

# 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 chapter when naming your fields, files, tables, and variables.

For details, see the following topics:

- Characters to avoid
- Case sensitivity
- File path specifications
- Terminal key definitions

## 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

For details, see the following topics:

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

## Using 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. See *OpenEdge Development: ABL Reference* for more information on the OPSYS function.

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" chapter of *OpenEdge Deployment: Managing 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 18. 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.

See *ABL Reference* for more information on the INPUT FROM statement.

# 3

# Coding style

For the best portability, this chapter describes techniques to isolate your user-interface code.

For details, see the following topics:

- Using preprocessor directives
- Using the VIEW-AS phrase

## Using 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.
```

# Using 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 on the VIEW-AS phrase, see *OpenEdge Development: ABL Reference*.