POSSE pbuild Utility

Contents

1 Overview

The POSSE pbuild utility, commonly referred to simply as pbuild or the pbuild script, compiles Progress 4GL code into either graphical or character UI mode r-code procedure libraries. By default, all 4GL code is compiled from the POSSE source repository into a UI-specific subdirectory in the POSSE root directory as defined by the environment variable $POSSE. It can also be run against one or more specific components.

ADE POSSE source code resides in the src source code directory; the value of $POSSE is the root directory where src resides. The pbuild utility compiles source code in the $POSSE/src, and places the r-code it generates into the $POSSE/gui and the $POSSE/tty subdirectories.

The POSSE pbuild utility compiles tty on UNIX, and tty and gui on Windows platforms.

2 Minimum Run Time Requirements

The following list identifies the minimum runtime requirements to run the pbuild script:

For Windows:

- Any supported Progress 9.1B platform.
- Progress ProVision v9.1B.
- Unix shell environment such as Cygwin (www.cygwin.com) or MKS Toolkit (www.mks.com).

For Unix:

- Any supported Progress 9.1B platform.
- Progress 4GL Development System v9.1B.

In addition, for all platforms:

- The environment variable $DLC must be defined, and point to the directory where Progress 9.1B is installed.
- The environment variable $POSSE must be defined, and point to the POSSE root directory where the POSSE src subdirectory exists.
- **Note:** On Windows, the typical installation of ProVision does not install the character mode client. You must run install with custom or complete to compile character mode applications on Windows.

3 How to Run pbuild

To run pubuild, you must perform two main steps:

1. Add pbuild to your PATH
2. Run pbuild from the command line

The following information provides details about each of these steps.

3.1 Adding pbuild to your PATH

The pbuild utility resides in $POSSE/src/adebuild. You can run pbuild by adding the directory $POSSE/src/adebuild to your shell $PATH variable.

For example, to add $POSSE/src/adebuild to your PATH, you might issue the following shell command:

For Unix or Windows with CygWin:

```
PATH="$PATH:$POSSE/src/adebuild" export PATH
```

For Windows with MKS Toolkit:

```
PATH="$PATH;$POSSE/src/adebuild" export PATH
```

3.2 Running pbuild from the Command Line

This section describes how to initiate the pbuild from the command line, and provides example information about this activity.

SYNTAX

The following command syntax box and associated table identify the command line information you enter in a shell window (either Unix, Cygwin, or MKS Toolkit) to initiate pbuild:

```
pbuild [uimode=<gui|tty>] [apps="app1 app2 ..."]
```

pbuild

Identifies that you intend to run the pbuild utility.

uimode

Identifies the user interface mode for which you intend to run the create r-code. The options are

- gui - graphical user interface r-code is created (Windows only, default)
- tty - character user interface r-code is created (UNIX default)

apps

Identifies the application component list, an optional space-separated list of POSSE component subdirectories to compile. The default is to compile all applicable components for the specified uimode value (either tty or gui).

Application Component Examples. The following table identifies and describes some example values you can enter to identify application components.

If the application (apps) value is …	Then…
wrapper	All 4GL procedures in the top-level directory $POSSE/src are compiled. These “wrapper” 4GL procedures are used to start individual POSSE ADE components.
prodict	All 4GL procedures under the $POSSE/src/prodict directory are compiled.
adedict prodict	All 4GL procedures associated with both the adedict and prodict components are compiled.

3.2.1 Command Line Examples

The following table suggests and briefly describes two examples of the type of information you can enter in the command line to run pbuild.

If you enter this example value(s)…	Then…
Pbuild	All POSSE code is compiled into r-code and/or procedure libraries for the platform-default user interface mode (that is gui on Windows, and tty on UNIX.)
pbuild uimode=tty apps=prodict	The POSSE data dictionary component prodict is compiled in character mode.
pbuild uimode=gui apps="adecomm adedict prodict"	The POSSE data dictionary (prodict) and data administration (adedict) components, as well as the common code (adecomm) component are compiled in gui mode.

3.3 Using Pbuild to Compile the ICF module

The pbuild utility can be used to compile the icf module in one of two configurations: single source and split source. At the present time (August 2001 timeframe), the icf module is a separate project from the rest of “posseall,” and as an additional complication it requires a specialized adm2 module (called “icf adm2” for the purpose of distinction) that resides on the icf_10 branch. At some point in the not-so-distant future, the icf module will most likely be incorporated into posseall and the “icf adm2” module on the icf_10 branch will be merged with the posseall codebase. Until then, the pbuild utility can still be used to compile the icf and “icf adm2” with a few extra setup steps.

3.3.1 Single Source Configuration

In this configuration, all of the source directories (i.e., posseall, icf, “icf adm2”) are installed under the same src directory. This is most likely the way the structure will be in the future, when icf and “icf adm2” are merged into the posseall codebase. If you set up your workspace in this manner, you must replace the “posseall adm2” directory with the “icf adm2” directory (from the icf_10 source branch). In this case, the pbuild script can be used in the normal manner as described in this document.

3.3.2 Split Source Configuration

The document “ICF Development with WinCVS” describes how to set up a split source environment, with the posseall modules under one directory and the icf and “icf adm2” modules under another. This allows the user to maintain a single workspace for both standard posseall development and also icf development. In a single source environment, both the posseall and icf are installed under the same src directory, and the “icf adm2” module must also be installed there on top of (replacing) the “posseall adm2”.

One important deviation from the “ICF Development with WinCVS” document is that, in order to use pbuild to build the icf and “icf adm2” in this configuration, you must place the icf module beneath the src directory (i.e., at the same level as the “icf adm2” directory).

Also, in addition to setting $POSSE to point to the directory above the src directory where the posseall modules reside, you must set a second environment variable prior to running the pbuild script: $POSSEICF must be set to point to the directory above where the src directory containing the icf and “icf adm2” modules are located.

Lastly, there is one more thing that must be done to properly build the icf modules in a split source configuration: since the “icf adm2” module is not a true application module (it will eventually be merged back into posseall), it is not part of the default list of applications known to pbuild. It must be explicitly specified in one of two ways:

1. If all applications are to be compiled, the following syntax will instruct the pbuild script to look for and compile the adm2 application that is located beneath $POSSEICF/src:

   Pbuild apps=”+icfadm2”

2. If only some applications are to be compiled, the “icf adm2” application can be specified just as any other application, e.g.,

   Pbuild apps=”prodict adm2 icf icfadm2”

Please refer to the document “ICF Development with WinCVS” for additional instructions for setting up an ICF development workspace, but note that the pbuild script can be used to build the rcode rather than manually using the Application Compiler tool.

4 Build Activities that Comprise the Pbuild Script

The following list outlines the sequence of activities that pbuild performs during the build process:

1. Convert command-line parameters into environment variables.
2. Setup environment variable constants for each platform.
3. Set PROPATH to: current directory + POSSE source directory + target e4gl directory.
4. Setup default Progress client based upon uimode value.
5. Setup default Unix/Windows TTY/GUI application directory lists.
6. Create temporary directory to do compiles in (pbuild.dir, under the working directory).
7. Create gui or tty destination directory if it does not exist.
8. Display significant environment variable values for user.
9. Generate Progress 4GL code from HTML-rich embedded speedscript code.
10. Create an empty “dictdb” database.

11. For each selected application directory:
    - Recursively find list of directories and create empty target directory.
    - Perform these component-specific tasks, as needed:
        - Create application-specific databases and/or schema.
        - Update PROPATH if needed.
    - Create a Progress 4GL procedure consisting of COMPILE statements for each file that needs to be compiled.
        - For prodict, intersperse DataServer schema create/remove procedure calls before and after DataServer 4GL code compilation.
    - Run the Progress client using standard startup parameters, database "dictdb", any application-specific databases, and the 4GL procedure created in the previous bullet.
    - Perform these component-specific post-compile tasks:
        - Remove .r files if compiled r-code is not required.
        - Package .r files into Progress procedure libraries.
        - Install .r or other component data files into target directory.
        - Create translation databases and install them with r-code.
        - Install icons with GUI r-code.
            - Cleanup temporary directory.

4.1 Can I Modify pbuild?

Pbuild is customizable to be run with each tool. Additionally, you may need to modify it if you add a new component (that is, add code to existing components, or add code for a new component)

Progress Software Company recommends that you become familiar with pbuild and its original, "as delivered" functionality first, before you add new components. When you add new components, you may need to make adjustments to accommodate new behaviors.

5 Related Documentation

The following Open Source documents provide additional information about the pbuild utility:

- *ENV How to be a POSSE Programmer*
- *ADE Overview*