Release Notes for Purify 5.2 HP-UX

Contents

Changes from Previous Releases

New in This Release

- Bug fixes and compatibility with OS patches.
- This release uses a new FlexLm based licensing. Read the new installation guide before installing the product. Use **rs_install** instead of **pure_install** for the installation.
- This is the last release to support HPUX 10.01 and HPUX 10.10.

New in Purify 5.1

- HP-UX 9.x is no longer supported.
- Support for Cygnus GNUPro 98r2 compilers.

New in Purify 5.0.1

- Bug fixes and compatibility with OS patches.
- 32-bit Purify 5.0.1 may be used with 32-bit PureCoverage 5.0.1.
- This is the last release to support HP-UX 9.x. Apex Ada is no longer supported.

New in Purify 5.0

- This version supports "wide mode" applications on HPUX 11.00: programs using 64-bit pointers, compiled with the option **+DA2.0W**. Please read the Restrictions section for important information about 64-bit development.
- Bug fixes and compatibility with OS patches.

New in Purify 4.4

- Bug fixes and compatibility with OS patches.
- Supports FLEXlm based licensing when installed as part of RSDSU.
- Support for Rational's ClearQuest defect tracking tool. Please see the Restrictions and Known Issues section below for details on how to use Purify with ClearQuest or ClearDDTS.

New in Purify 4.3

- Support for Apex 3.0.0 Ada and C++ on Solaris and HPUX.

  In addition to support for code generated by the Apex 3.0.0 C++ and Ada compilers, this release provides Apex GUI integration in the form of the Purify Viewer Edit, Debug, Check-in, Check-out, and JIT debugging features.

New In Purify 4.2

- Bug fixes

Supported systems

Operating system and Hardware

Purify has been tested with HP-UX versions 10.01, 10.10, 10.20, and 11.00 from Hewlett Packard. This is the last release to support HP-UX versions 10.01 and 10.10.

Purify also supports 64-bit wide-mode programs on HPUX 11.00. A wide-mode program is one that uses 64-bit longs and pointers, built with the compiler option **+DA2.0W**.

Compilers

Purify has been tested with the following compilers:

- Bundled cc.
- ANSI cc.
- C++.
- aCC.
- GNU gcc and g++ versions, through version 2.8.1.
- Cygnus GNUpro v.98r2
- f77.

See the Restrictions and Known Issues section for more details.

Threads

Purify supports these threads packages:

- DCE threads (either **libdce** or **libcma**).
- HP-UX kernal threads.

Restrictions and Known Issues

General

- Instrumented programs now always use immediate binding for the initial load of shared libraries and for any dynamic loads using shl_load().

  This change was made due to a problem in HP dld patches from June 1998 for HPUX 11.00 and August 1998 for HP-UX 10.20, which caused the instrumented application to hang after a call to shl_load(). One known instance of this problem occurs when you use getservbyname() because it loads network protocols using shl_load().

  Linker and dld patches available after 3/99 don't have this problem. If the user has an earlier patch, an upgrade is strongly recommended.
- Purify will not run on short (14-character
-) file systems.
- Operating system revisions below 10.01 are no longer supported.
- The PA-RISC 1.0 CPU is no longer supported.
- Profile Based Optimization is not supported.
- Applications which link to over 50 large dynamic libraries may experience reduced startup times by setting the runtime option **-lazy-dld-maps=1**.

Using 32-bit vs 64-bit Purify

Purify supports both 32- and 64-bit development. "Wide" mode, or 64-bit applications are those compiled with the **+DA2.0W** option - apps using 64-bit pointers. "Narrow" mode applications are traditional 32-bit programs.

Purify ships in 2 configurations, one supporting wide mode and the other supporting narrow. Both can be installed on the same file system, but the 64-bit version can only be used on 64-bit HP-UX 11.x systems.

If both install directories are in your path, Purify will auto-select the correct wide or narrow mode version, in most situations (see below for limitations). For example, you can install two versions of Purify:

purify-5.1-beta-H1-hpux (32-bit)
purify-5.1-beta-P1-hpux (64-bit)

(Beta and proto release with **H<*n*>** in their name are 32-bit release. **P<*n*>** signifies a 64-bit release.)

Or, for a final release:

purify-5.1-hpux (32-bit)
purify-5.1-hpux64 (64-bit)

If the two install directories are in your path, then running **purify** will automatically select the correct version, based on the type of program you are linking. The same is true for Quantify.

Even if only one install directory is on your path, auto-selection will occur if both versions are properly installed: Running **pure_install** on each version will prompt you for the location of the other product.

If you already have your licenses installed and do not choose to run **pure_install**, you can set up this connection between the two install directories by running the script **pure_link_32_64** in each install directory:

```
% cd purify-5.1-hpux
% pure_link_32_64
<answer the questions>
% cd purify-5.1-hpux64
% pure_link_32_64
```

Auto-selection only works between 32- and 64-bit Purify from 5.0 onwards.

In some situations, auto-selection does not have enough context to tell which version (32 or 64-bit) you need. This is true for the options:

-help
-printhomedir
-test-license
-version

In these cases, Purify defaults to the 32-bit version unless you explicitly specify what you want, using the new **-ptr64** and **-ptr32** options:

```
% purify -ptr64 -test-license
% quantify -ptr64 -printhomedir
% purify -ptr32 -version
% quantify -ptr32 -help
```

Failure to include **-ptr<32|64>** in these cases may yield the wrong information. For example, you may get the product home directory for the 32-bit product when you wanted the 64-bit product.

These options are NOT necessary during normal instrumentation and viewing operations:

```
% purify cc -g -o foo foo.o
% quantify -view my_app.qv
```

If you attempt to use the 64-bit Purify using **-ptr64**, or by having it on your PATH first, on a non 11.x systems, execution will fail. It only runs on HP-UX 11.x and later.

Because of a defect in auto-selection, auto-selection does not occur when using **-nolink**. You must use **-ptr64** or **-ptr32** to ensure the correct version is used:

```
% quantify -ptr32(or -ptr64) -nolink ld mylib.a
```

Restrictions on the HP-UX 11.00-wide (64-bit) version of Purify

- Static data checking is not supported.
- Using Purify with old versions of the HP-UX linker may cause the install test to fail with an error from the linker saying the **+nodynhash** option is not recognized. You should obtain a newer linker from HP. This option is support by 64-bit linkers from 07-Jan-1999 and later.

  You may also work around this problem by include the following in your PUREOPTIONS environment variable:

  ```
  -force-no-dynhash=no
  ```

  The default setting of this option (to “yes”) is used to work around an HP bug in newer linkers.
- Shared-library **fini** sections (static destructors) do not run.
- **shl_unload** and dlclose are not implemented. When a program attempts to unload a shared library, the call will appear to succeed, but the library will not actually be closed or unloaded. Also, C++ “static destructors” are not run for the library.
- There is a bug in strlen in the 64-bit libc which causes the example program **hello.c** to get numerous UMR's instead of only one, as you might expect.
- The open-file summary does not include the file position for 64-bit programs.
- A breakpoint at **purify_stop_here** does not show a proper stack traceback.
- Errors involving large amounts of memory do not report properly. For instance, an FMR when copying a 17MB freed memory block would report as a 1MB FMR.
- Shared library search is not fully implemented: If you application relies on locating a library via SHLIB_PATH, Purify may not be able to find it.
- When you want to set a breakpoint on **purify_stop_here** using HP's wdb (or gdb) debugger, use this command:

  ```
  b *purify_stop_here
  ```

  When you stop at **purify_stop_here** in the debugger, the debugger's stack trace will not be correct: You won't see the stack trace for you program, only for Purify's runtime libraries.

  The easiest way to get around this is to set a temporary breakpoint at the address which is in register r15 using this wdb command:

  ```
  tb *$r15
  ```

  Then, you can use the “continue” command to stop in your program at the instruction that caused the Purify error report.

- When a Japanese locale is specified using the LANG environment variable, instrumentation of archive libraries may fail with the message:

  ```
  Error: Child process exited with status = 1.
  ```

  This is caused by an 'ar' failure in this locale. A workaround is to unsetenv LANG before instrumenting.

User Interface

- If a large number of items are selected, **Expand all** followed by **Collapse all** can crash some unpatched versions of the OpenWindows 3.0 server. This occurs if you are displaying on a SUN workstation.
- If you expand or collapse messages while the **Continue**, **Reset**, etc., buttons are displayed, the buttons may subsequently be incorrectly positioned.
- The **Edit** and **Coverage** toolbar items may be slow to respond.
- If EDITOR is set to vuepad (**/usr/vue/bin/vuepad**), the edit button will be unable to position the editor window to the correct line. This is a **vuepad** limitation. The workaround is to use a more capable editor.
- The Purify GUI menus and buttons become inaccessible if either the **NumLock** or **ScrollLock** key is activated. The workaround is to switch them off, or add the following line(s) to your **$HOME/.Xdefaults** file.

  ```
  ! Ignore the NumLock and ScrollLock keys on
  ! mouse buttons
  Purify*ignoreModifierMask: Mod3|Mod2
  ```

  This second workaround will take effect for a new Purify viewer after you restart your X-session or run a command like **xrdb -merge $HOME/.Xdefaults**.
- The **Invoke ClearDDTS** Button has been modified to bring up the ClearQuest web interface. This feature only works with Netscape Navigator.

  The site-wide URL for ClearQuest can be given during installation or set by manually editing the file **pure_clearquest_url** in your Purify home directory. A user can override the site-wide URL by setting the environment variable PURE_CLEARQUEST_URL.

  This feature is partly implemented by a shell script, (**pure_invoke_clearquest** in your Purify home directory) to allow you to tailor its operation to your needs. If you wish, you may copy and customize this script. As long as the directory containing the script appears in your search path before your Purify home directory, it will be used instead of the original script.

  If you prefer to use Purify with ClearDDTS, you can do so by setting the X resource:

  ```
  Purify*ddtsCommandString
  ```

to **xddts**, if **xddts** is in your search path, or to the full path to your **xddts** executable. **xddts** is invoked by a shell script (**pure_invoke_ddts** in your Purify home directory). If you wish to customize it, please read the section on customizing **pure_invoke_clearquest** above.

If you already have a customized **pure_invoke_ddts** script in your search path, all you need to do is set your X resource as described above, and Purify will find your customized script automatically.

The following copyright applies to portions of this ClearQuest integration code:

Compilers

- The GNU gcc extensions are not tested against Purify. Most gcc extensions will probably work fine. Known limitations at present include problems with nested functions (e.g.: making a pointer to a nested function and attempting to call through it will not work).
- Purify supports the aCC compiler from HP but there is a required patch: for HP-UX 10.01 through 10.10, use PHSS_14507. For HP-UX 10.20, use PHSS_17689. For HP-UX 10.20 you also need PHSS_17225.
- C revisions 9.61 through 9.65
- These compilers generate incorrect debugging information for some functions. Programs compiled using these compilers may not be debuggable after translation. In this case Purify may produce a message beginning “Failure relocating...” HP patch PHSS_4923 corrects this problem.
- Exception handling

  g++ 2.7.2 exceptions are not supported.
- Aggressive Loading
- Some compilers load data from memory but ignore the data that has been read. Purify will signal a UMR if the loaded data is uninitialized. In some sense this is a false error report because the uninitialized data can not affect your program.

Purify'ing X Applications

- When running a Purify'd X application, there is a potential for deadlock if your application causes Purify to generate a message while the application is holding the X lock, since Purify will be unable to generate the message, and the application is blocked until the message is delivered.

  To avoid this kind of problem, you should run your application on a different X server than the Purify UI or Purify stderr output, or you should use the **-log-file=** or **-view-file=** options to specify a file to capture messages for inspection after your application is finished.

  A convenient way to debug on two displays is to pre-start the Purify Viewer on one display ("slave"), and then start the application on the other display ("master"):

  ```
  % purify -display slave:0 -view a.out.X &
  % a.out.X -display master:0
  ```

  The two commands must be executed on the same computer, but it could be the workstation associated with either display, or altogether another computer remote from both displays. The application will connect to the already started Purify Viewer, and messages will not conflict with the X display interactions of the application under test.

Debuggers

- XDB & Softdebug
  - Use **<*purifyhome*>/purify_xdb** to invoke xdb on an instrumented program.
  - An instrumented program receives a signal 18 (death of child) during its initialization. Place "z 18 sir" in your **~/.xdbrc** file to suppress this warning. Failure to suppress the warning will sometimes cause xdb to fail.
  - XDB, when debugging a program that uses shared libraries, reads the symbol table from **/lib/dld.sl**. However, Purify has changed your program to use an instrumented version of **dld.sl**. The symptom of this mismatch is the message:

    ```
    Wait...loading shared-library map tables.
    xdb panic: Mapped addresses for dld
       overlap text segment for dld
    ```

    There is a simple workaround for this problem and we've implemented it in the shell script **<*purifyhome*>/purify_xdb**. Whenever you use xdb on an instrumented program use this script to invoke xdb.
  - Typing **Control-C** while running an instrumented program under xdb requires caution. There is a high probability that the interrupt will occur inside the code Purify has added to your application. If this is the case, you must single step or set a breakpoint and continue before you attempt to call any subroutines (the Purify API is an example).

- The XDB single stepping command, "**s**" sometimes fails to find the next source line. When this happens, usually near a subroutine call, program execution continues until the next breakpoint. The "**S**" command is always reliable.

- DDE - Distributed Debugging Environment
  - The DDE debugger at release 2.10 has been used with instrumented programs. The shell script: **<*purifyhome*>/purify_dde** implements one of the workarounds.
  - Instrumented programs get a SIGCHLD signal at program initialization. One workaround is to just hit **go** when the program stops. The following commands added to a .**dderc** file also help:

```
prop system -on
alias 'after_debug delete intercept signal SIGCHLD; \
  prop system -off; \
  breakpoint -in main -entry -exit; \
  go
```

- Attaching to a running process
  - JIT debugging may fail to attach to your application if the executable resides on an NFS file system mounted without the **nointr** option. The HP Debugger reference manual says:

    "If you get a Permission denied error message when you attach to a running process, it is likely that you are running either the debugger or the target process over an NFS link and that the relevant file system is mounted with the default intr option. You must mount the file system with the nointr option to resolve this problem. Use a command like the following to mount the file system containing the debugger:

```
mount -o nointr[,other_options] \
    system:/opt/langtools /tools
```

    Use a command like the following to mount the file system containing the target process:

```
mount -o nointr[,other_options] \
    system:/test_area /test
```

It is probably easier to create an auxiliary mount for the file system than to unmount and remount it."

Old Style Fixups

Purify does not support a type of relocation information known as "old style fixups". These were generated by HP-UX system software before release 3.0. If Purify detects old style fixups the message:

```
Object file has incompatible format
  (may be older than HPUX 3.0)
```

is generated. We have seen this problem with HP's **libsql.a** and some of Oracle's **Oracle6** libraries.

There is a simple workaround. Given a problem object module (or modules) the workaround is to have **/bin/ld** build a new object module. Suppose the old object modules are called '**foo.o**' and '**bar.o**'. Issuing the command:

```
% ld -r -o new_foo.o foo.o
% ld -r -o new_bar.o bar.o
```

or

```
% ld -r -o foo_and_bar.o foo.o bar.o
```

would generate a new object module where the old style fixups have been removed.

In the case of an archive file the following script will create a new archive given the full pathname of the original:

```
#!/bin/sh
# Remove old fixups from an archive.
# Supply original .a name as first argument.
cd /tmp
lib=new_‘basename $1‘
ar x $1
rm -f $lib
for member in ‘ar t $1‘ ; do
   ld -r -o _$member $member
   ar q $lib _$member
   rm $member _$member
done
echo Created ‘pwd‘/$lib
```

Threads

- Call chains describing when memory was malloced or freed do not always include the thread id.
- The Purify API functions purify_map_pool() and purify_map_pool_id() are not MT safe.
- Customers using unsupported threads packages should contact Rational Software technical support (**support@rational.com**) to ensure compatibility.

Unsupported Features

- SBR and SBW errors are not reported on HPUX.