OPENSTEP 4.1 Release Notes: The GNU Source-Level Debugger

This file contains information about **gdb**, the GNU Debugger. For more information, see the **gdb** reference in **/NextLibrary/Documentation/NextDev/DevTools/Debugger.pdf**.

The **gdb** debugger for OPENSTEP Enterprise is based on the version 4.15.1 release from GNU. This brings with it many, if not most of the features of debugging on Mach, although there are inevitably some differences.

Debugging Objective-C: Differences from Mach gdb

The PDO version of **gdb** is more of a multi-language debugger than the older **gdb** on Mach (especially the older version of **gdb** available with previous releases of NEXTSTEP). This **gdb** has separate features for many different languages, including Objective-C. It attempts to guess the source language by looking at the extension of the source file name (**.m** or **.M** for Objective-C). To find out what **gdb**'s current language is, type **show language**. To force the current language to Objective-C, type **set language objective-c**.

Calling Methods from gdb

To call a method in your program from **gdb**, use the **print**, **set**, or **call** commands with an argument that looks just like a method call in Objective-C. For example:

```
(gdb) print [myClass showValue: 12]
```

If the method comes from a category, you must include the category name, as shown here:

```
(gdb) print [myClass(myCategory) showValue: 12]
```

Listing and Setting Breakpoints on Methods

To refer to a method in a **list** or **break** command, you can supply the full class and method name, including a + or **-** to indicate a class method or instance method. If there is a category name, you must give that too:

```
(gdb) list +[myClass init]
(gdb) break -[myClass(myCategory) showValue]
```

You can also set breakpoints or list a method just by giving a selector. If the selector is implemented by more than one class, **gdb** will list the corresponding methods and ask you to choose one or more:

```
(gdb) break init
[0] cancel
[1] all
[2] -[Change init] at Change.m:20
[3] -[DrawApp init] at DrawApp.m:130
[4] -[Graphic init] at Graphic.m:139
>
```

You would then enter your choice or choices at the ">" prompt.

Getting Information about Classes and Methods

gdb for PDO has the **info classes** and **info selectors** commands. These commands accept the same regular expression language as **gdb**'s **info type** and **info function** commands (that is, the UNIX-style regular expression language). This is a change from the Mach **gdb**, where **info classes** and **info selectors** accept a slightly different regular expression language. For instance, to learn about class names beginning with "NS" (using the "^" character to designate “beginning with”):

```
(gdb) info classes ^NS
```

To learn about selectors, you can use the **info selectors** command. To find every selector containing the string "withObject:" you could enter:

```
(gdb) info selector withObject:
```

To learn about methods, you can use the **info function** command, which also takes a regular expression. Since the square bracket characters ('[' and ']') have special significance in regular expressions, you can quote them with a backward slash to prevent their being treated as special characters. To list all the methods of a class, you might say:

```
(gdb) info function \[MyClass
```

To list all the methods whose selector ends with "count:", you might say:

```
(gdb) info function count:\]
```

If you want to know about a specific method of a specific class, but you aren't sure if it belongs to a category, you can use the ".*" wildcard sequence to stand for any number of any characters:

```
(gdb) info function MyClass.*mySelector:
```

Known Problems

Problems with Underscores

gdb doesn't correctly handle methods for which either the class, the category, or the selector has underscores in the name, except when the underscore is the first character of the class or selector name.

Thread Support

gdb has no support for debugging multiple threads on either Solaris or HP-UX.

Random SIGTRAPs

gdb occasionally throws random SIGTRAPs. If you verify that the instruction at the $PC is not a trap instruction, you should be able to continue by saying "signal 0" (that's a zero). This tells **gdb** to continue the child without passing any signal to it.