NAME

debugger_init, debugger_eval, debugger_break − debugging embedded scripts and debugging remotely.

SYNOPSIS

debugger_init ?*host* ?*port*??

debugger_eval ?*switches*? *script*

debugger_attached

debugger_break ?*str*?

INTRODUCTION

The Tcl procedures defined here allow the Ajuba Debugger to be used to debug remote, embedded and CGI applications. In order for your application to establish and maintain communication with the debugger, you must modify your application to source the initdebug.tcl file and call the **debugger_init** and **debugger_eval** procedures.

COMMAND PROCEDURES

The following procedures are provided by the debugger library:

debugger_init ?*hostname* ?*port*??

Establish the connection with the debugger that is currently running on *hostname* and listening on *port*. By default *hostname* is **localhost** and the port is **2576**. After the connection has been made, the debugger will instrument any files that are sourced into the interpreter with the **source** command, or any commands that appear in the arg list of the **debugger_eval** command. The command returns 1 if the connection was successful and returns 0 if the connection failed.

debugger_eval ?*switches*? *script*

The **debugger_eval** command instruments and invokes the specified *script*. The **debugger_eval** command allows a program to explicitly instrument a block of code that might not otherwise be instrumented by the debugger. If the script is not currently connected to the debugger, **debugger_eval** simply evaluates the script argument.

If the initial arguments to **debugger_eval** start with −, then they are treated as switches. The following switches are currently supported:

−**name** *name* Associate a *name* with the script. This causes the debugger to remember breakpoint information as if the script were sourced from a file of the given *name*. This feature can be useful in remote debugger situations, or when evaluating blocks of dynamically generated code that are used multiple times. By creating a unique *name* for each block, the user can set breakpoints in the block that persist across invocations.

− − Marks the end of switches. The argument following this one will be treated as *script* even if it starts with a −.

debugger_attached

The **debugger_attached** returns 1 if the script is currently connected to the debugger. Otherwise it returns 0.

debugger_break ?*str*?

The **debugger_break** command will cause a break to occur when executed. The effect is similar to the effect of a break−point on the line containing the **debugger_break** command (the only difference is that *str* is evaulated before the break occurs). When the break occurs a dialog is presented in the debugger's GUI. If *str* is given (and not empty) the value of *str* is presented in the dialog box. If the script is not currently connected to the debugger, **debugger_break** acts as a no−op.

EXAMPLES

The example code below demonstrates the simplest way to establish a remote connection and debug an entire script remotely. The connection is established between the local machine and **remoteMachine** via port **2576**. At this point it is assumed that the debugger is running on **remomoteMachine** and is listening on port **2576**. See the User's Guide or online help system for more information on how to specify the port that the debugger listens on. The file **main.tcl** is then sourced, which will cause the contents of the file, and any subsequent sourced files, to become instrumented (unless the preferences set in the debugger indicate otherwise.)

```
source initdebug.tcl if {[debugger_init remoteMachine 2576] == 0} {
   return "cannot communicate with remoteMachine on port 2576" } source main.tcl
```

The next example shows how to control exactly which commands become instrumented. Establish the connection exactly like the previous example. The commands that create the variables x, y and z will not be instrumented and the debugger will not step through theses lines. The commands that create the variables a, b and c are inside the **debugger_eval**. This causes these commands to be instrumented and the debugger will step through these commands.

```
source initdebug.tcl if {[debugger_init remoteMachine 2576] == 0} {
   return "cannot communicate with remoteMachine on port 2576" } set x 1 set y 2 set z 3 debugger_eval {
   set a [expr {$x + 1}]
   set b [expr {$y + 1}]
   set c [expr {$z + 1}] }
```

This example is especially relevant when debugging embedded scripts. Simply add the first two lines to the beginning of the script and wrap the existing script in a call to **debugger_eval**.

KEYWORDS

remote debugging, debugger_init, debugger_eval, instrument, attach, detach