NAME

bgexec − Run programs in the background while handling Tk events. kill − Terminate program or send signal.

SYNOPSIS

blt::bgexec *varName* ?*switches*? *program* ?*arg*?...
blt::kill *processid* ?*signal*?

DESCRIPTION

The **kill** command terminates a *processid* or under unix sends a signal.

The **bgexec** command executes a *program* pipleline using the **Tcl** event-loop allowing other events to continue to be serviced. Upon completion it sets the global variable *varName* with a list of 4 status values: a text token, the process-id, the exit code, and a text message. **Bgexec** pro vides capabilities similar to the **exec** command, but with added support for callbacks, output to variables and termination.

When used with no options, the returned value from **bgexec** is the output from the *program*. But when the last *arg* is an ampersand (&) the *program* runs detached, and **bgexec** immediately returns with a list of the process ids created in the command pipeline. Detached processes can be interrupted and terminated simply by setting *varName*.

The valid *switches* are as follows:

−check *num*
Interval in ms to poll for the exiting processes. The default is 1000.

−closeonkill *millisecs*
Force close of stdin/stdout on kill after the given interval. This lets kill finalize processes, even uninterruptably sleeping ones unable to receive signals. The default is **0** for do not force close.

−command *script*
Specifies a command to call upon command completion/termination. Two extra arguments are appended before the call. The data output from the command, and the status info as set into *varName*.

−decodeerror *encodingName*
Specifies the encoding of the stderr channel. This affects only data returned to the Tcl interpreter. No translation is done on file redirection. For example if data is to be converted from Unicode for use in Tcl, you would use the "unicode" encoding. The default is that no tranlation is performed.

−decodeoutput *encodingName*
Specifies the encoding of the stdout channels. This affects only data returned to the Tcl interpreter. No translation is done on file redirection. For example if data is to be converted from Unicode for use in Tcl, you would use the "unicode" encoding. The default is that no tranlation is performed.

−echo *boolean*
Indicates if the pipeline's stderr stream should be echoed. *Note: this option is deprecated.*

−error *varName*
Specifies that a global variable *varName* is to be set with the contents of stderr after the program has completed.

−keepnewline *boolean*
Specifies that a trailing newline should be retained in the output. If *boolean* is true, the trailing newline is truncated from the output of the **−onoutput** and **−output** variables. The default value is **true**.

−killsignal *signal*
Specifies the signal to be sent to the program when terminating. This option is available only on Unix. *Signal* can either be a number (typically 1-32) or a mnemonic (such as SIGINT). If *signal* is the empty string, then no signal is sent. The default signal is **9** (SIGKILL).

−lasterror *varName*
Specifies a variable *varName* that is updated whenever data becomes available from standard error of the program. *VarName* is a global variable. Unlike the **−error** option, data is available as soon as it arrives.

−lastoutput *varName*
Specifies a variable *varName* that is updated whenever data becomes available from standard output of the program. *VarName* is a global variable. Unlike the **−output** option, data is available as soon as it arrives.

−limit *numBytes*
Limit the size of the returned data to *numBytes*, terminating the program if exceeded. The limit applies to both stdout and stderr.

−linebuffered *boolean*
Specifies that updates should be made on a line-by-line basis. Normally when new data is available **bgexec** will set the variable (**−lastoutput** and **−lasterror** options) or invoke the command (**−onoutput** and **−onerror** options) delivering all the new data currently available. If*boolean* is true, only one line at a time will be delivered. This can be useful when you want to process the output on a line-by-line basis. The default value is **false**.

−local *boolean*
When *boolean* is true, any unqualified variables or command options are treated as local to the current namespace. This is mostly useful for non-detaching (no ampersand) commands. Note that using this flag with a detached command will use variables from the current namespace, not from the current proc stack-frame.

−onerror *command*
Specifies the start of a Tcl command that will be executed whenever new data is available from standard error. The data is appended to the command as an extra argument before it is executed.

−onoutput *command*
Specifies the start of a Tcl command that will be executed whenever new data is available from standard output. The data is appended to the command as an extra argument before it is executed.

−output *varName*
Specifies a global variable *varName* to be set with the output of the program, upon completion.

−raise *boolean*
When *boolean* is **true**, a non-zero return code from a non-detached command will raise an error (.ie emulates **exec**). The default is **false** an error is generated only if one of the following occurs: invalid options are given, a redirection error, or process creation failure (eg. executable program not found). Detached commands, of course, never raise an error on a non-zero return code.

−− This marks the end of the options. The following argument will be considered the name of a program even if it starts with a dash (−).

USAGE

Invoking **bgexec** without a trailing ampersand will block and wait for result. However, other Tcl events continue to be serviced. This prevents Tcl from hanging, eg:

```
pack [text .t]
set val [blt::bgexec myStatus du -s]
```

Note that text widget .t continues to respond to events.

CALLBACKS

Here is an example that invokes the Unix **du** program with a **-command** callback.

```
proc Done {data status} {  puts "Done($status)\n$data" }

blt::bgexec myStatus  -command Done   du -s $dir &
```

When **du** has completed, the handler **Done** is called with data and status. Also, the global variable *myStatus* is set to contain the program's exit status, eg:

```
EXITED 26811 0 {child completed normally}
```

If *myStatus* is set before **du** has completed, the process will be killed. Under Unix, this sends a signal (SIGKILL by default). Under Win32, **TerminateProcess** is called.

VARIABLE

Here is another example, this time using the **-output** option to direct output to a variable.

```
global myStatus myOutput
blt::bgexec myStatus -output myOutput du -s $dir
puts "Disk usage for $dir is $myOutput"
```

Upon completion, **MyOutput** will contain the output of the program.

STDERR

Various **bgexec** options can be used to capture **stderr** separately from **stdout**.

```
global myStatus myOutput myErrs
blt::bgexec myStatus -output myOutput -error myErrs du -s $dir
```

The **−error** option is similar to **−output** in that it sets a variable when the program completes with data written to stderr.

LOCAL

By default, **bgexec** treats variable or command options as being in the global namespace. The **-local** option can change this to use the current namespace. Thus data can be collected to namespace-local variables even those inside of procs, eg.

```
proc Work {} {
  blt::bgexec myStatus -local 1 -output val -error err du -s
  puts "VAL=$val"
  puts "ERR=$err"
}
```

which collects data to local variables.

For detached processes, **-local** will cause data to aggregate to namespace variables, ie. outside the proc, eg.

```
namespace eval ::Ns {
  set pval {}
  set perr {}
  proc Work {} {
    blt::bgexec myStatus -local 1 -output pval -error perr du -s &
  }
```

```
}
```

This collects data to **::Ns::pval** and stderr to **::Ns::perr**. Similarly, proc names (eg **-onoutput**) will be relative to the current namespace.

PROGRESS

The **−output** and **−error** variables are set only after the program completes. But if a program runs for a long time, you can gather data as it becomes available using the **−onoutput** option. As new data becomes available, this command is executed, with data appended as an argument.

```
proc GetInfo { data } { puts $data }

blt::bgexec myStatus -onoutput GetInfo du -s $dir
```

The **−onerror** option performs a similar function for the stderr data stream.

ERROR HANDLING

Like **exec**, **bgexec** returns an error if the exit code of the program is non-zero. To handle this invoke **bgexec** from within a **catch**.

```
catch { blt::bgexec myStatus -output myOutput du -s $dir }
```

Detached jobs will generate an error only if the program startup failed. Otherwise the only indication is the status code set in *myStatus*.

TKWAIT

By default, **bgexec** waits for a program to finish and returns the resulting output. To detach a program simply append an ampersand (&) as the last argument on the command line, eg.

```
global myStatus myOutput
blt::bgexec myStatus -output myOutput du -s $dir &
```

Bgexec will then return immediately with the spawned process ids as the result. If needed **tkwait** can be used to wait for the program to finish:

```
global myStatus myOutput
blt::bgexec myStatus -output myOutput du -s $dir &
      ...
tkwait variable myStatus
```

Note however that using **tkwait** can be dangerous. Multiple **tkwait**/**vwait** calls must complete in the reverse order called. The BLT **busy** command can be used to try and enforce this, but a better alternative is to just use **-command** instead.

DIFFERENCES WITH EXEC

Using **bgexec** without an ampersand will not hang Tcl: events continue to be serviced by the event handler while the call blocks. Also unlike **exec**, an error will not be generated if output is appears on **stderr**. And output from **stderr** can be separately managed and collected (without having to redirect to files). Finally, **bgexec** ensures that invoked processes get properly cleaned up at termination.

DIFFERENCES WITH FILEEVENT

Since Tk 4.0, a subset of **bgexec** can be achieved using the **fileevent** command. The steps for running a program in the background are:

Execute the program with the **open** command (using the "|" syntax) and save the file handle.

```
  global fileId
```

```
set fileId [open "|du -s $dir" r]
```

Next register a Tcl code snippet with **fileevent** to be run whenever output is available on the file handle. The code snippet will read from the file handle and save the output in a variable.

```
fileevent fileId readable {
  if { [gets $fileId line] < 0 } {
      close $fileId
      set output $temp
      unset fileId temp
  } else {
      append temp $line
  }
}
```

However, **Bgexec** is simpler and less error prone than using **open** + **fileevent**. You don't have to worry about non-blocking I/O. Everything is handled for you automatically.

Moreover, **bgexec** can run programs that **fileevent** can not. **Fileevent** assumes that the when stdout is closed the program has completed. But some programs, like the Unix **compress** program, reopen stdout, fooling **fileevent** into thinking the program has terminated. In the example above, we assume that the program will write and flush its output line-by-line. However when running another program, your application can block in the **gets** command reading a partial line.

Bgexec gives you get back the exit status of the program. It also lets you reliably kill detached processes and allows you to collect data from both stdout and stderr individually. Finally, since data collection is handled in C code, **bgexec** is faster and more efficient.

SEE ALSO

busy, exec, tkwait, vwait

KEYWORDS

exec, background, busy