

# Multi-Process Debug Target Server (rdbgd)

Documentation »

ACCESS Linux Platform Native Development »

Development Tools »

Multi-Process Debug Target Server (rdbgd)

The term "rdbgd" (pronounced "R-debug-D") is the name of the multi-process Debug Target Server used by ACCESS Linux Platform. rdbgd is a server which runs on the target to provide process information, start gdb servers, download and launch applications, and the like. It is used primarily by ACCESS-provided Eclipse plug-ins, although you can connect to it and interact with it using telnet.

rdbgd is part of the root filesystem and is launched automatically at target startup.

rdbgd Protocol (Commands) ^TOP^

The following sections detail the elements of the rdbgd protocol. After connecting to rdbgd using telnet, you can enter these protocol elements manually, treating them as commands.

cd ^TOP^

## Purpose

Sets the current working directory in the context of the rdbgd process using chdir. This is the starting point for path searches for pathnames not beginning with '/.

## Syntax

```
cd /working/directory/target/path
```

/working/directory/target/path is the desired working directory.

## Output

```
cd errno
```

errno is the error returned by chdir.

close ^TOP^

#### Purpose

Closes the current rdbgd client connection.

#### Syntax

close

#### Output

There is no output string. The connection is simply closed.

connect ^TOP^

#### Purpose

Attach a gdbserver to an already running process.

#### Syntax

connect pid

pid is the numeric process ID of the target application.

#### Output

connect errno port

errno is the numeric failure code (0 on success) and port is the number of the TCP port on which the new gdbserver is listening.

install ^TOP^

### Purpose

Transfer a file to the target system.

### Syntax

```
install /path/to/destination/on/target N username groupname mode
```

N is the size of the file in bytes, username and groupname are the names of the file's intended owner and group identity, and mode is the numeric permissions mask to be set on the destination file via `chmod()`. The N bytes of file data must immediately follow the command string's trailing LF character.

### Output

```
install errno
```

where `errno` is the numeric failure code (0 on success).

kill ^TOP^

### Purpose

Deliver a kill signal to a process.

### Syntax

```
kill signal pid
```

signal is the number of the signal to deliver and pid is the numeric process ID of the target application.

### Output

kill `errno`

`errno` is the numeric failure code (0 on success). Note that this reflects the success of signal delivery, not a guarantee that any specific result has occurred. In particular, even if delivering a signal 9 (SIGKILL), the target application is not guaranteed to have been terminated.

launch `^TOP^`

### Purpose

Execute a program under `gdbserver`.

### Syntax

launch [-terminal] /path/to/program/on/target [arg1 arg2 ...]

The optional command-line arguments to the program are separated by whitespace (which is stripped when invoking the program) and where the `-terminal` option creates a port from which `stdin` is read and to which `stdout` is written for the executed program.

### Output

launch `errno gdbserverPort [terminalPort]`

`errno` is the numeric failure code (0 on success), and `gdbserverPort` is the number of the TCP port on which the new `gdbserver` is listening. An optional `terminalPort` value is passed back if the `-terminal` option was specified. This is the number of the TCP port to which `stdout`, `stdin` and `stderr` are redirected.

list `^TOP^`

### Purpose

Fetch a list of processes.

## Syntax

list

Output

list N

followed by N lines of proc[pid]/stat output, one line per process.

N is the number of running processes.

package ^TOP^

Purpose

Interact with an ACCESS Linux Platform bundle.

Syntax

package install /path/on/target/to/package.bar

package run com.access.PackageName

package debug com.access.PackageName

package register {/path/on/target/to/package.bar | /package/directory/on/target}

package unregister com.access.PackageName

package delete com.access.PackageName

package names

package exit [com.access.PackageName]

For package install and package register, /path/on/target/to/package.bar is the target path to an ACCESS Linux Platform bundle. Use the regular install command to get it there if necessary. com.access.PackageName is the installed name of an ACCESS Linux Platform bundle.

Output

Output string for install, run, register, unregister, or delete:

package errno

errno is the numeric failure code (0 on success)

Output string for package debug:

package errno pid

errno is the numeric failure code (0 on success), and pid is the process ID of the ACCESS Linux Platform application waiting to be debugged.

Output string for package names:

package errno  
... N lines of bundle names

errno is the numeric failure code (0 on success). Following this are the bundle names, one on each line.

printenv ^TOP^

Purpose

Prints the environment variables in the context of the rdbgd process.

Syntax

printenv

Output

printenv errno

...N lines of "ENV=value" output

run ^TOP^

Purpose

Execute a program independently of gdb/gdbserver.

Syntax

run [-terminal] /path/to/program/on/target [arg1 arg2 ...]

The optional command-line arguments to the program are separated by whitespace (which is stripped when invoking the program). The -terminal option creates a port from which stdin is read and to which stdout is written for the executed program.

Output

run errno [terminalPort]

errno is the numeric failure code (0 on success). An optional terminalPort value is passed back if the -terminal option was specified. This is the number of the TCP port to which stdout, stdin and stderr are redirected.

setenv ^TOP^

Purpose

Sets the specified environment variable in the context of the running rdbg executable, using putenv.

Syntax

setenv var value

var is the variable to be set and value is its new value.

#### Output

setenv errno

errno is the error returned by putenv.

version ^TOP^

#### Purpose

Returns the protocol version of rdbgd.

#### Syntax

version

#### Output

version N

N is the number of the protocol version supported by this rdbgd.

Basic rdbgd Tasks ^TOP^

Launching rdbgd ^TOP^

You do not need to explicitly launch rdbgd; it is automatically launched on the target or in the Simulator at bootup.

## Installing and Debugging an Application ^TOP^

First, make sure that the rdbgd tool is running on the target. The target could be your native system.

Telnet to the rdbgd server from the host:

```
telnet 192.168.2.101 5038
```

where the 192.168.2.101 is replaced by the IP address of your target. The port should be 5038, since that is the default rdbgd port (unless you changed it on startup).

Install the application on the target by copying it over to the appropriate place in the location of the rootfs on the host. (Although you could use an rdbgd command, it is much easier to simply copy it over).

Launch a gdbserver session using the rdbgd connection:

```
launch /path/on/target/to/executable
```

For instance, to launch the built-in calculator application, you would use:

```
launch /opt/alp/bundles/com.access.apps.calc/  
libalp_calc.so
```

The output looks like this:

```
launch 0 5023
```

This output tells you that the launch was successful and that the gdbserver is listening on port 5023.

Launch gdb and connect to the gdbserver at the returned port.

Now you can debug.

### Connecting to an Existing Process (ARM) ^TOP^

Launch your program on the device.

Find out its process ID with the ps command.

First Host Session:

```
telnet 192.168.2.101 5038
```

When connected to rdbgd:

```
connect <PID from step 2, above>
```

Output:

```
connect <errno> <port>
```

Second Host Session:

<your ARM GDB> <host path to executable>

In GDB:  
target remote 192.168.2.101:<port from the  
connect command, above>

You are now connected to the target process.

{moscomment}