Class sun.tools.debug.RemoteDebugger

```
java.lang.Object
   |
   +----sun.tools.debug.RemoteDebugger
```

---

public class **RemoteDebugger**
extends Object

The RemoteDebugger class defines a client interface to the Java debugging classes. It is used to instantiate a connection with the Java interpreter being debugged.

Version:
1.0 26 June 1995
Author:
Thomas Ball

Constructor Index

o **RemoteDebugger**(String, String, DebuggerCallback, boolean)
Create a remote debugger.

Method Index

o **close**(boolean)
Close the connection to the remote debugging agent.
o **findClass**(String)
Find a specified class.
o **freeMemory**()
Report the free memory available to the Java interpreter being debugged.
o **gc**()
Free all unreferenced objects.
o **get**(Integer)
Get an object from the remote object cache.
o **getExceptionCatchList**()
Return the list of the exceptions the debugger will stop on.
o **getSourcePath**()

Return the source file path the Agent is currently using.

- **itrace**(boolean)
  Turn on/off instruction tracing.
- **listBreakpoints**()
  Return a list of the breakpoints which are currently set.
- **listClasses**()
  List the currently known classes.
- **listThreadGroups**(RemoteThreadGroup)
  List threadgroups
- **run**(int, String[])
  Load and run a runnable Java class, with any optional parameters.
- **setSourcePath**(String)
  Specify the list of paths to use when searching for a source file.
- **totalMemory**()
  Report the total memory usage of the Java interpreter being debugged.
- **trace**(boolean)
  Turn on/off method call tracing.

Constructors

- **RemoteDebugger**

```
public RemoteDebugger(String host,
                      String password,
                      DebuggerCallback client,
                      boolean verbose) throws Exception
```

Create a remote debugger.

Parameters:

host – the name of the system where a debuggable Java instance is running (default is localhost)
password – the password reported by the debuggable Java instance
client – the object to which notification messages are sent (it must support the DebuggerCallback interface)
verbose – turn on internal debugger message text

Methods

- **close**

```
public void close(boolean closeRemoteAgent)
```

Close the connection to the remote debugging agent.

Parameters:

closeRemoteAgent – terminate the Java interpreter being debugged

- **get**

```
public RemoteObject get(Integer id)
```

Get an object from the remote object cache.
Parameters:
id – the remote object's id
Returns:
the specified RemoteObject, or null if not cached.

o **listClasses**

```
public RemoteClass[] listClasses() throws Exception
```

List the currently known classes.

o **findClass**

```
public RemoteClass findClass(String name) throws Exception
```

Find a specified class. If the class isn't already known by the remote debugger, the lookup request will be passed to the Java interpreter being debugged. NOTE: Substrings, such as "String" for "java.lang.String" will return with the first match, and will not be successfully found if the request is passed to the remote interpreter.
Parameters:
name – the name (or a substring of the name)of the class
Returns:
the specified (Remote)Class, or null if not found.

o **listThreadGroups**

```
public RemoteThreadGroup[] listThreadGroups(RemoteThreadGroup tg) throws Exception
```

List threadgroups
Parameters:
tg – the threadgroup which hold the groups to be listed, or null for all threadgroups

o **gc**

```
public void gc() throws Exception
```

Free all unreferenced objects. The remote debugger maintains a copy of each object it has examined, so that references won't become invalidated by the garbage collector of the Java interpreter being debugged.

o **trace**

```
public void trace(boolean traceOn) throws Exception
```

Turn on/off method call tracing. When turned on, each method call is reported to the stdout of the Java interpreter being debugged. This output is not captured in any way by the remote

debugger.

Parameters:

traceOn – turn tracing on or off

o **itrace**

```
public void itrace(boolean traceOn) throws Exception
```

Turn on/off instruction tracing. When turned on, each Java instruction is reported to the stdout of the Java interpreter being debugged. This output is not captured in any way by the remote debugger.

Parameters:

traceOn – turn tracing on or off

o **totalMemory**

```
public int totalMemory() throws Exception
```

Report the total memory usage of the Java interpreter being debugged.

o **freeMemory**

```
public int freeMemory() throws Exception
```

Report the free memory available to the Java interpreter being debugged.

o **run**

```
public RemoteThreadGroup run(int argc,
                             String argv[]) throws Exception
```

Load and run a runnable Java class, with any optional parameters. The class is started inside a new threadgroup in the Java interpreter being debugged. NOTE: Although it is possible to run multiple runnable classes from the same Java interpreter, there is no guarantee that all applets will work cleanly with each other. For example, two applets may want exclusive access to the same shared resource,such as a specific port.

Parameters:

argc – the number of parameters

argv – the array of parameters: the class to be run is first, followed by any optional parameters used by that class.

Returns:

the new ThreadGroup the class is running in, or null on error

o **listBreakpoints**

```
public String[] listBreakpoints() throws Exception
```

Return a list of the breakpoints which are currently set.

Returns:

an array of Strings of the form "class_name:line_number".

o **getExceptionCatchList**

```
public String[] getExceptionCatchList() throws Exception
```

Return the list of the exceptions the debugger will stop on.
Returns:
an array of exception class names, which may be zero−length.

o **getSourcePath**

```
public String getSourcePath() throws Exception
```

Return the source file path the Agent is currently using.
Returns:
a string consisting of a list of colon−delineated paths.

o **setSourcePath**

```
public void setSourcePath(String pathList) throws Exception
```

Specify the list of paths to use when searching for a source file.
Parameters:
pathList − a string consisting of a list of colon−delineated paths.