CHAPTER 5

USING ADMINISTRATION TOOLS

This chapter describes the Administration tools provided with VisiBroker for Java, and includes the following sections:

Command Usage	5-2
Using the Object Activation Daemon Utilities	5-2
Listing Objects with oadutil list	5-4
Un-Registering Objects Using oadutil unreg	5-10
Reporting All Objects and Services with osfind	5-12

COMMAND USAGE

The use of the VisiBroker administration tools described in this chapter differ, depending on whether you have a Unix or a Windows environment.

You may view command line arguments accepted by a service by entering the command name as follows:

```
osagent -help
```

USING THE OBJECT ACTIVATION DAEMON UTILITIES

The oadutil commands provides a way for you to manually register, un-register, and list the object implementations available on your VisiBroker system. The oadutil commands are implemented in Java and use a command line interface. Each command is accessed by invoking the `oadutil` command, passing the type of operation to be performed as the first argument.

NOTE *An* `oadj` *process must be started on at least one host in your network before you can use the oadutil commands.*

The `oadutil` command has the following syntax:

SYNTAX
```
oadutil {list|reg|unreg} [options]
```

The options for this tool vary, depending on whether you specify `list`, `reg` or `unreg`.

Converting Interface Names to Repository IDs

When registering or un-registering an object with the OAD, the oadutil commands allow you to specify either an object's IDL interface name or it's repository id.

An interface name is converted to a repository ID as follows:

1. Prepend "IDL:" to the interface name.
2. Replace all non-leading instances of "::" with a slash ("/") character.
3. Append ":1.0" to the interface name.

For example, the IDL interface name:

```
::Module1::Module2::IntfName
```

would be converted to the following repository ID:

```
IDL:Module1/Module2/IntfName:1.0
```

The #pragma id and #pragma prefix mechanisms can be used to override the default generation of repository id's from interface names. If the #pragma id mechanism is used in user-defined IDL files to specify non-standard repository IDs, the conversion process outlined above will not work. In these cases, you must use -r repository id argument and specify the object's repository ID.

To obtain the repository id of the object implementation's most derived interface, use the method `_repository_id ()` defined on all CORBA objects.

LISTING OBJECTS WITH OADUTIL LIST

The `oadutil list` command returns all ORB object implementations registered with the Object Activation Daemon. Each OAD has its own implementation repository database where the registration information is stored.

NOTE *An* `oadj` *process must be started on at least one host in your network before you can use the oadutil list command.*

The `oadutil list` command has the following syntax:

SYNTAX
```
oadutil list [options]
```

The `oadutil list` command accepts the following command line arguments:

OPTION	DESCRIPTION
-i <interface name>	Lists the implementation information for objects of a particular IDL interface name. Only one of the following options may be specified at a particular time: -i, -r, or -s. Note: All communications with the ORB and BOA reference an object's repository id instead of the interface name. For more information about the conversion performed when specifying an interface name, see "Converting Interface Names to Repository IDs" on page 5-2.
-r <repository id>	Lists the implementation information of a specific repository id. See "Converting Interface Names to Repository IDs" on page 5-2 for details on specifying repository IDs. Only one of the following options may be specified at a particular time: -i, -r, or -s.
-s <service name>	Lists the implementation information for a specific service name. Only one of the following options may be specified at a particular time: -i, -r, or -s.
-o <object name>	Lists the implementation information for a specific object name. You can use this only if the interface or repository id is specified in the command statement. This option is not applicable when the -s argument is used.
-h <OAD host name>	Lists the implementation information for objects registered with an OAD running on a specific remote host.

OPTION	DESCRIPTION
-verbose	Turns verbose mode on, causing messages to be output to std-out.
-version	Prints the version of this tool.
-full	Lists the status of all implementations registered with the OAD.

Description

The `oadutil list` utility allows you to list all ORB object implementations registered with the Object Activation Daemon. The information for each object includes:

- Interface names of the ORB objects.
- Instance names of the object offered by that implementation.
- Full pathname of the server implementation's executable.
- Activation policy of the ORB object (shared, unshared, or per-method).
- Reference data specified when the implementation was registered with the OAD.
- List of arguments to be passed to the server at activation time.
- List of environment variables to be passed to the server at activation time.

The following is an example of a local list request, specifying an interface name and object name:

EXAMPLE
```
oadutil list –i Bank::AccountManager –o VisigenicBank
```

The following is an example of a remote list request, specifying a host IP address:

EXAMPLE
```
oadutil list –h 206.64.15.198
```

REGISTERING OBJECTS WITH OADUTIL REG

The `oadutil reg` command allows you to register one or more ORB object implementation with the Object Activation Daemon. Registered implementations will be activated automatically by the OAD when a client requests a bind to the object. Once an object implementation has been registered with the OAD, the VisiBroker Smart Agent is notified that the object is available through the OAD. The object's implementation is also added to the OAD's implementation repository.

NOTE *An `oadj` process must be started on at least one host in your network before you can use the oadutil reg command.*

If an object implementation is started manually as a persistent server, it does not need to be registered with the OAD. Objects with a local scope are not visible to the osagent or the OAD.

The `oadutil reg` command has the following syntax:

SYNTAX
```
oadutil reg [options]
```

NOTE *If an object implementation is started manually as a persistent server, it does not need to be registered with the OAD.*

The options for the oadutil reg command accepts the following command line arguments:

OPTION	DESCRIPTION
REQUIRED	
-i <interface name>	Specifies a particular IDL interface name. Only one of the following options may be specified at a particular time: -i, -r, or -s. Note: See "Converting Interface Names to Repository IDs" on page 5-2 for details on specifying repository IDs.
-r <repository id>	Specifies a particular repository id. Only one of the following options may be specified at a particular time: -i, -r, or -s.
-s <service name>	Specifies a particular service name. Only one of the following options may be specified at a particular time: -i, -r, or -s.

-o <object name>	Specifies a particular object. You can use this only if the interface name or repository id is specified in the command statement. This option is not applicable when the -s argument is used.
-cpp <file name to execute>	Specifies the full path of an executable file that must create and register an object that matches the -o/-r/-s arguments. Applications registered with the -cpp argument must be stand-alone executables.
-java <full class name>	Specifies the full name of a Java class containing a main routine. This application must create and register an Object that matches the -o/-r/-s argument. Classes registered with the -java argument will be executed with the command: vbj <full_classname>
OPTIONAL	
-host <OAD host name>	Specifies a specific remote host where the OAD is running.
-verbose	Turns verbose mode on, causing messages to be output to stdout.
-version	Prints the version of this tool.
-d <referenceData>	Specifies reference data to be passed to the server upon activation.
-a arg 1 -a arg2	Specifies the arguments to be passed to the spawned executable as command-line arguments. Arguments can be passed with multiple -a (arg) parameters. They will be propagated in order to create the spawned executable.

-e env1 -e env2	Specifies environment variables to be passed to the spawned executable. Arguments can be passed with multiple -e (env) parameters. They will be propagated in order to create the spawned executable.
-p {shared | unshared | per-method}	Specifies the activation policy of the spawned objects. The default policy is SHARED_SERVER. shared = Multiple clients of a given object share the same implementation. Only one client is activated by an OAD at a particular time. unshared = Only one client of a given implementation will bind to the acitvated server. If multiple clients wish to bind to the same object implementation, a separate server is activated for each client application. A server exits when its client application disconnects or exits. per-method = Each invocation of a method results in a new sever being activated. The server exits upon completion of the method invocation.

Examples

PERFORMING A LOCAL REGISTRATION BY SPECIFYING REPOSITORY ID

The following command will register with the OAD the VisiBroker for Java program factory_r found in the /home/developer/project1 directory. It will be activated upon request for objects of repository ID IDL:ehTest/Factory:1.0 (which corresponds to the interface name ehTest::Factory). The instance name of the object to be activated is ReentrantServer, and that name is also passed to the spawned executable as a command-line argument. This server has the unshared policy, by which it will be terminated when the requesting client breaks its connection to the spawned server.

EXAMPLE
```
prompt> oadutil reg –r IDL:ehTest/Factory:1.0 –o ReentrantServer \
–cpp /home/developer/Project1/factory_r –a ReentrantServer \
–p unshared
```

PERFORMING A LOCAL REGISTRATION BY SPECIFYING IDL INTERFACE NAME

The following command will register with the OAD, the VisiBroker for Java class `Server`. The specified Java class must be found in the OAD's CLASSPATH. In this example, the specified Java class must activate an object of repository ID IDL:Bank/AccountManager:1.0 (corresponding to the interface name IDL name Bank::AccountManager) and instance

name CreditUnion. The sever will be started with unshared policy, ensuring that it will terminate when the requesting client breaks its connection. In addition, the System Property DEBUG is passed to the spawned Java server.

NOTE *The double backslashes are necessary to avoid having the shell interpret the backslashes before passing them to the oadutil.*

EXAMPLE
```
prompt> oadutil reg –i Bank::AccountManager –o CreditUnion \
–java Server –a CreditUnion –p unshared –e DEBUG=1
```

The previous registration tells the OAD to execute the following command when spawning the requested server:

```
vbj –DDEBUG=1 Server CreditUnion
```

PERFORMING A REMOTE REGISTRATION TO AN OAD ON WINDOWS NT

To register an implementation with an OAD on a remote host, use the -h argument to oadutil reg.

The following is an example of how to perform a remote registration to an OAD on Windows NT from a UNIX shell. The double backslashes are necessary to avoid having the shell interpret the backslashes before passing them to oadutil.

EXAMPLE
```
prompt> oadutil reg –r IDL:Library:1.0 Harvard \
–cpp c:\\vbroker\\examples\\library\\libsrv.exe –p shared –h 100.64.15.198
```

Passing Environment Variables to the Object Through the OAD

When the registered class is activated by the OAD, the following environment variables can be passed to the object by the OAD:

- PATH
- CLASSPATH
- OSAGENT_PORT
- OSAGENT_ADDR

The environment of all process spawned by the OAD contains all of the environment variables specified when the implementation was registered. Definitions for PATH, CLASSPATH, OSAGENT_PORT, and OSAGENT_ADDR are all inherited from the OAD's own

environment, unless they have been explicitly defined at registration time with the -e command line option.

UN-REGISTERING OBJECTS USING OADUTIL UNREG

The `oadutil unreg` command allows you to un-register one or more object implementations registered with the Object Activation Daemon. Once an object is unregistered, it can no longer be automatically activated by the OAD if a client requests the object. Only objects that have been previously registered via the `oadutil reg` command may be unregistered with `oadutil unreg`.

If you specify only an interface name, all ORB objects associated with that interface will be un-registered. Alternatively, you may identify a specific ORB object by its interface name and object name. When you unregister an object, all processes associated with that object will be terminated.

NOTE *An* `oad` *process must be started on at least one host in your network before you can use the oadutil reg command.*

The `oadutil unreg` command has the following syntax:

SYNTAX `oadutil unreg [options]`

The options for the `oadutil unreg` command accepts the following command line arguments:

OPTION	DESCRIPTION
REQUIRED	
-i <interface name>	Specifies a particular IDL interface name. Only one of the following options may be specified at a particular time: -i, -r, or -s. Note: See "Converting Interface Names to Repository IDs" on page 5-2 for details on specifying repository IDs.
-r <repository id>	Specifies a particular repository id. Only one of the following options may be specified at a particular time: -i, -r, or -s.
-s <service name>	Specifies a particular service name. Only one of the following options may be specified at a particular time: -i, -r, or -s.

-o <object name>	Specifies a particular object name. You can use this only if the interface name or repository id is included in the command statement. This option is not applicable when the -s argument is used.

OPTIONAL

-host <host name>	Specifies the host name where the OAD is running.
-verbose	Enables verbose mode, causing messages to be output to stdout.
-version	Prints the version of this tool.

Example

The `oadutil unreg` utility un-registers one or more ORB objects from these three locations:

- the Object Activation Daemon
- the implementation repository
- the Smart Agent

The following is an example of how to use the oadutil unreg command. It unregisters the implementation of the Bank: :AccountManager named VisigenicBank from the local OAD.

EXAMPLE
```
oadutil unreg –i Bank::AccountManager –o VisigenicBank
```

REPORTING ALL OBJECTS AND SERVICES WITH OSFIND

The `osfind` command reports on all VisiBroker related objects and services which are currently available on a given network.

You can use `osfind` to determine the number of Smart Agent processes running on the network and the exact host on which they are executing. The `osfind` command also reports on all VisiBroker objects that are active on the network. You can use `osfind` to monitor the status of the network and locate stray objects during the debugging phase.

The `osfind` command has the following syntax:

SYNTAX `osfind [options]`

The following options are valid with `osfind`. If none of the -r/-a/-o options are specified, `osfind` lists all of the agents, OAD's, and implementations in your domain.

OPTION	DESCRIPTION
-b	Specifies the VisiBroker 2.0 backward-compatible osfind mechanism Note: 3.0 activated objects will be listed as manually started when this flag is set.
-r <repository ID>	Lists all implementations of the given repository ID Note: You can use regular expressions in repository IDs. Example: -r "^.*" will list all interfaces. See "Converting Interface Names to Repository IDs" on page 5-2 for details on specifying repository IDs.
-a	Lists all Smart Agents in your domain.
-o	Lists all Object Activation Daemons in your domain.
-LOCdebug <0 or 1>	If you enter a one, this turns on debugging. The default is for debugging to be turned off. If you've turned on debugging, you can turn it off by entering a zero; for example: -LOCdebug 0.
-LOCverify <0 or 1>	If you enter a zero, this turns off verification. Verification is normally turned on--The Smart Agent does an immediate update and verifies that existence of the object before it returns an object reference to the Location Service. If you turn this feature on it will be slower but will return the most accurate information. To turn verification on, enter a one; for example: -LOCverify 1. The default is off.