A
DBMS_JAVA Package

This chapter provides a description of the `DBMS_JAVA` package. The functions and procedures in this package provide an entry point for accessing RDBMS functionality from Java.

A.1 hotload_module

```
procedure hotload_module(module VARCHAR2)
```

Hotloads the module with the specified name in the current schema. If packages are later added to or removed from this module, then the module must be reloaded or re-hotloaded. Adds a `module-data` database schema data object in the same schema as the `module-info` of the module, and marks the `module-info` as being hotloaded.

A.2 hotload_module

```
procedure hotload_module(module VARCHAR2, schema VARCHAR2)
```

Hotloads the module with the specified name in the specified schema. If packages are later added to or removed from this module, then the module must be reloaded or re-hotloaded. Adds a `module-data` database schema data object in the same schema as the `module-info` of the module, and marks the `module-info` as being hotloaded.

A.3 unhotload_module

```
procedure unhotload_module(module VARCHAR2)
```

Removes the hotloaded `module-data` of the module with the specified name in the current schema, and marks the `module-info` as no longer being hotloaded.

A.4 unhotload_module

```
procedure unhotload_module(module VARCHAR2, schema VARCHAR2)
```

Removes the hotloaded `module-data` of the module with the specified name from the specified schema, and marks the `module-info` as no longer being hotloaded.

A.5 jar_digest_bytes

```
function jar_digest_bytes(jarname VARCHAR2,
                                        schema VARCHAR2,
                                        algorithm VARCHAR2) return RAW
```

Returns the JAR-digest of the specified JAR file in the specified schema, which is calculated according to the specified algorithm, as a `RAW`. The supported algorithms are the ones provided by `java.security.MessageDigest`, and includes the `MD5`, `SHA-1`, and `SHA-256` algorithms.

A.6 jar_digest

```
function jar_digest (jarname VARCHAR2, schema VARCHAR2, algorithm VARCHAR2)
  return VARCHAR2
```

Returns the JAR-digest of the specified JAR file in the specified schema, which is calculated according to the specified algorithm, as a `VARCHAR2`. The supported algorithms are the ones provided by `java.security.MessageDigest`, and includes the `MD5`, `SHA-1`, and `SHA-256` algorithms.

A.7 longname

```
FUNCTION longname (shortname VARCHAR2) RETURN VARCHAR2
```

Returns the fully qualified name of the specified Java schema object. Because Java classes and methods can have names exceeding the maximum SQL identifier length, Oracle JVM uses abbreviated names internally for SQL access. This function returns the original Java name for any truncated name. An example of this function is to display the fully qualified name of classes that are invalid:

```
SELECT dbms_java.longname (object_name) FROM user_objects
WHERE object_type = 'JAVA CLASS' AND status = 'INVALID';
```

> **See Also:**
>
> "Shortened Class Names"

A.8 shortname

```
FUNCTION shortname (longname VARCHAR2) RETURN VARCHAR2
```

You can specify a full name to the database by using the `shortname()` routine of the `DBMS_JAVA` package, which takes a full name as input and returns the corresponding short name. This is useful when verifying that your classes loaded by querying the `USER_OBJECTS` view.

> **See Also:**
>
> "Shortened Class Names"

A.9 get_compiler_option

```
FUNCTION get_compiler_option(name VARCHAR2, optionName VARCHAR2) RETURN VARCHAR2
```

Returns the value of the option specified through the `optionName` parameter. It is one of the functions used to control the options of the Java compiler supplied with Oracle Database.

A.10 set_compiler_option

```
PROCEDURE set_compiler_option(name VARCHAR2, optionName VARCHAR2, value VARCHAR2)
```

Is used to set the options of the Java compiler supplied with Oracle Database.

A.11 reset_compiler_option

```
PROCEDURE reset_compiler_option(name VARCHAR2, optionName VARCHAR2)
```

Is used to reset the specified compiler option to the default value.

A.12 resolver

```
FUNCTION resolver (name VARCHAR2, owner VARCHAR2, type VARCHAR2) RETURN VARCHAR2
```

Returns the resolver specification for the object specified in `name` and in the schema specified in `owner`, where the object is of the type specified in `type`. The caller must have `EXECUTE` privilege and have access to the given object to use this function.

The `name` parameter is the short name of the object.

The value of `type` can be either `SOURCE` or `CLASS`.

If there is an error, then `NULL` is returned. If the underlying object has changed, then `ObjectTypeChangedException` is thrown.

You can call this function as follows:

```
SELECT dbms_java.resolver('tst', 'HR', 'CLASS') FROM DUAL;
```

This would return:

```
DBMS_JAVA.RESOLVER('TST','HR','CLASS')
----------------------------------------
((* HR)(* PUBLIC))
```

A.13 derivedFrom

```
FUNCTION derivedFrom (name VARCHAR2, owner VARCHAR2, type VARCHAR2) RETURN VARCHAR2
```

Returns the source name of the object specified in `name` of the type specified in `type` and in the schema specified in `owner`. The caller must have `EXECUTE` privilege and have access to the given object to use this function.

The `name` parameter, as well as the returned source name, is the short name of the object.

The value of `type` can be either `SOURCE` or `CLASS`.

If there is an error, then `NULL` is returned. If the underlying object has changed, then `ObjectTypeChangedException` is thrown.

The returned value will be `NULL` if the object was not compiled in Oracle JVM.

You can call this function as follows:

```
SELECT dbms_java.derivedFrom('tst', 'HR', 'CLASS') FROM DUAL;
```

This would return:

```
DBMS_JAVA.DERIVEDFROM('TST','HR','CLASS')
-----------------------------------------
tst
```

A.14 fixed_in_instance

```
FUNCTION fixed_in_instance (name VARCHAR2, owner VARCHAR2, type VARCHAR2) RETURN NUMBER
```

Returns the permanently kept status for object specified in `name` of the type specified in `type` and in the schema specified in `owner`. The caller must have `EXECUTE` privilege and have access to the given object to use this function.

The `name` parameter is the short name for the object.

The value of `type` can be either of `RESOURCE`, `SOURCE`, `CLASS`, or `SHARED_DATA`.

The number returned is either `0`, indicating the status is not kept, or `1`, indicating the status is kept.

You can call this function as follows:

```
SELECT dbms_java.fixed_in_instance('tst', 'HR', 'CLASS') FROM DUAL;
```

This would return:

```
DBMS_JAVA.FIXED_IN_INSTANCE('TST','HR','CLASS')
-----------------------------------------
0
```

Consider the following statement:

```
SELECT dbms_java.fixed_in_instance('java/lang/String', 'SYS', 'CLASS') FROM DUAL;
```

This would return:

```
DBMS_JAVA.FIXED_IN_INSTANCE('JAVA/LANG/STRING','SYS','CLASS')
-------------------------------------------------------------
1
```

A.15 set_output

```
PROCEDURE set_output (buffersize NUMBER)
```

Redirects the output of Java stored procedures and triggers to the `DBMS_OUTPUT` package.

> **See Also:**
>
> "Redirecting the Output"

A.16 export_source

```
PROCEDURE export_source(name VARCHAR2, schema VARCHAR2, src BLOB)

PROCEDURE export_source(name VARCHAR2, src BLOB)

PROCEDURE export_source(name VARCHAR2, src CLOB)

PROCEDURE export_source(name varchar2, schema varchar2, src CLOB)
```

Are used to export a Java source schema object to a `BLOB` or `CLOB` in the same database. If the schema parameter is not specified, then the current schema is used. The internal representation of the source uses the UTF8 format.

A.17 export_class

```
PROCEDURE export_class(name VARCHAR2, schema VARCHAR2, src BLOB)

PROCEDURE export_class(name VARCHAR2, src BLOB)
```

Are used to export Java class schema objects to a `BLOB` in the same database. If the schema parameter is not specified, then the current schema is used.

A.18 export_resource

```
PROCEDURE export_resource(name VARCHAR2, schema VARCHAR2, src BLOB)

PROCEDURE export_resource(name VARCHAR2, src BLOB)

PROCEDURE export_resource(name VARCHAR2, schema VARCHAR2, src CLOB)

PROCEDURE export_resource(name VARCHAR2, src CLOB)
```

Are used to export the resource schema object, described by the name parameter in the current schema, to a `CLOB` or `BLOB` in the same database. If the schema parameter is specified, then that schema is used for object lookup.

A.19 loadjava

```
PROCEDURE loadjava(options VARCHAR2)
PROCEDURE loadjava(options VARCHAR2, resolver VARCHAR2)
PROCEDURE loadjava(options VARCHAR2, resolver VARCHAR2, status NUMBER)
```

Enable you to load classes in to the database using a call, rather than through the `loadjava` command-line tool. You can call this procedure within your Java application as follows:

```
CALL dbms_java.loadjava('... options...');
```

The options are identical to those specified on the command line. Each option should be separated by a space. Do not separate the options with a comma. The only exception to this is the `loadjava -resolver` option, which contains spaces. For `-resolver`, specify all other options first, separate these options by a comma, and then specify the `-resolver` options, as follows:

```
CALL dbms_java.loadjava('... options...', 'resolver_options');
```

Do not specify the `-thin`, `-oci`, `-user`, and `-password` options, because they relate to the database connection. The output is directed to `System.err`. The output typically goes to a trace file, but can be redirected.

> **See Also:**
>
> "The loadjava Tool"

A.20 dropjava

```
PROCEDURE dropjava(options VARCHAR2)
```

Enables you to drop classes within the database using a call, rather than through the `dropjava` command-line tool. You can call this procedure within your Java application as follows:

```
CALL dbms_java.dropjava('... options...');
```

> **See Also:**
>
> "The dropjava Tool"

A.21 grant_permission

```
PROCEDURE grant_permission(grantee VARCHAR2, permission_type VARCHAR2, permission_name
VARCHAR2,
permission_action VARCHAR2)
```

The result of a call to `grant_permission` is an active row in the policy table granting the permission as specified by `permission_type`, `permission_name`, and `permission_action` to `grantee`. If an enabled row matching these parameters already exists, then the table is unmodified. If the row exists but is disabled, then it is enabled. If no matching row exists, then one row is inserted. Parameter descriptions:

- `grantee` is the name of a schema or role
- `permission_type` is the fully qualified name of a class that extends `java.lang.security.Permission`
- `permission_name` is the name of the Permission
- `permission_action` is the action of the Permission

> **See Also:**
>
> "Fine-Grain Definition for Each Permission"

A.22 grant_permission

```
PROCEDURE grant_permission(grantee VARCHAR2, permission_type VARCHAR2, permission_name
VARCHAR2,
permission_action VARCHAR2, key OUT NUMBER)
```

Adds a new policy table row granting the permission as determined by the parameters.

Parameter descriptions:

- `grantee` is the name of a schema or role
- `permission_type` is the fully qualified name of a class that extends `java.lang.security.Permission`
- `permission_name` is the name of the Permission
- `permission_action` is the action of the Permission
- `key` is the key of the newly inserted row that grants the Permission. This value is -1, if an error occurs.

> **See Also:**
>
> "Fine-Grain Definition for Each Permission"

A.23 restrict_permission

```
PROCEDURE restrict_permission(grantee VARCHAR2, permission_type VARCHAR2,
permission_name VARCHAR2,
permission_action VARCHAR2)
```

Results in an active row in the policy table restricting the permission as specified by `permission_type`, `permission_name`, and `permission_action` to `grantee`. If a restricting row matching these parameters already exists then the table is unmodified. If no matching row exists then one is inserted.

Parameter descriptions:

- `grantee` is the name of a schema or role
- `permission_type` is the fully qualified name of a class that extends `java.lang.security.Permission`
- `permission_name` is the name of the Permission
- `permission_action` is the action of the Permission

> **See Also:**
>
> "Fine-Grain Definition for Each Permission"

A.24 restrict_permission

```
PROCEDURE restrict_permission(grantee VARCHAR2, permission_type VARCHAR2,
permission_name VARCHAR2,
permission_action VARCHAR2, key OUT NUMBER)
```

Adds a new policy table row restricting the permission as determined by the parameters.

Parameter descriptions:

- `grantee` is the name of a schema or role
- `permission_type` is the fully qualified name of a class that extends `java.lang.security.Permission`
- `permission_name` is the name of the Permission
- `permission_action` is the action of the Permission
- `key` is the key of the newly inserted row that grants the Permission. This value is -1, if an error occurs.

> **See Also:**
>
> "Fine-Grain Definition for Each Permission"

A.25 grant_policy_permission

```
PROCEDURE grant_policy_permission(grantee VARCHAR2, permission_schema VARCHAR2,
permission_type VARCHAR2,
permission_name VARCHAR2)
```

A specialized version of the `grant_permission(grantee VARCHAR2, permission_type VARCHAR2, permission_name VARCHAR2, permission_action VARCHAR2)` procedure for granting `PolicyTablePermission` permissions.

Parameter descriptions:

- `grantee` is the name of a schema or role
- `permission_schema` is the schema of the permission
- `permission_type` is the fully qualified name of a class that extends `java.lang.security.Permission`
- `permission_name` is the name of the Permission, which can be a glob asterisk ('*')

> **See Also:**
>
> "Acquiring Administrative Permission to Update Policy Table"

A.26 grant_policy_permission

```
PROCEDURE grant_policy_permission(grantee VARCHAR2, permission_schema VARCHAR2,
permission_type VARCHAR2,
permission_name VARCHAR2, key OUT NUMBER)
```

A specialized version of the `grant_permission(grantee VARCHAR2, permission_type VARCHAR2, permission_name VARCHAR2, permission_action VARCHAR2, key OUT NUMBER)` procedure for granting `PolicyTablePermission` permissions. Parameter descriptions:

- `grantee` is the name of a schema or role
- `permission_schema` is the schema of the permission
- `permission_type` is the fully qualified name of a class that extends `java.lang.security.Permission`
- `permission_name` is the name of the Permission, which can be a glob asterisk ('*')
- `key` is the key of the newly inserted row that grants the Permission. This value is -1, if an error occurs.

> **See Also:**
>
> "Acquiring Administrative Permission to Update Policy Table"

A.27 revoke_permission

```
PROCEDURE revoke_permission(permission_schema VARCHAR2, permission_type VARCHAR2,
permission_name VARCHAR2,
permission_action VARCHAR2)
```

Disables every active permission in the policy table that matches the parameters. The result is the same as calling the "disable_permission" procedure on every matching row. The rows are not deleted and kept in the table and can be activated by a call to "grant_permission" procedure with parameters matching those in the `revoke_permission` procedure.

Parameter descriptions:

- `permission_schema` is the name of a schema or role
- `permission_type` is the fully qualified name of a class that extends `java.lang.security.Permission`
- `permission_name` is the name of the Permission
- `permission_action` is the action of the Permission

> **See Also:**
>
> "Enabling or Disabling Permissions"

A.28 disable_permission

```
PROCEDURE disable_permission(key NUMBER)
```

Disables existing policy table row matching the specified key. The row remains in the table as an `INACTIVE` row. No error is reported if the key does not identify a row. The `disable_permission` procedure checks user permissions for policy table access and may throw a `SecurityException`.

> **See Also:**
>
> "Enabling or Disabling Permissions"

A.29 enable_permission

```
PROCEDURE enable_permission(key NUMBER)
```

Enables the existing policy table row matching the specified key. No error is reported if the key does not identify a row. The `enable_permission` procedure checks user permissions for policy table access and may throw a `SecurityException`.

> **See Also:**
>
> "Enabling or Disabling Permissions"

A.30 delete_permission

```
PROCEDURE delete_permission(key NUMBER)
```

Removes the existing policy table row matching the specified key. Before deleting the row, you must disable it as mentioned in "disable_permission". The `delete_permission` procedure has no effect if the row is still active or if `key` matches no rows.

> **See Also:**
>
> "Enabling or Disabling Permissions"

A.31 set_preference

```
PROCEDURE set_preference(user VARCHAR2, type VARCHAR2, abspath VARCHAR2, key VARCHAR2,
value VARCHAR2)
```

Inserts or updates a row in the `SYS:java$prefs$` table as follows:

```
CALL dbms_java.set_preference('HR', 'U', '/my/package/method/three', 'windowsize',
'22:32');
```

The `user` parameter specifies the name of the schema to which the preference should be attached. If the logged in schema is not `SYS`, then `user` must specify the current logged in schema or the `INSERT` will fail. The `type` parameter can take either the value `U`, indicating user preference, or `S`, indicating system preference. The `abspath` parameter specifies the absolute path for the preference. `key` is the preference key used for the lookup, and `value` is the value of the preference key.

A.32 runjava

```
FUNCTION runjava(cmdline VARCHAR2) RETURN VARCHAR2
```

Takes the Java command line as its only argument and runs it in Oracle JVM.

> **See Also:**
>
> "About Using the Command-Line Interface"

A.33 runjava_in_current_session

```
FUNCTION runjava_in_current_session(cmdline VARCHAR2) RETURN VARCHAR2
```

Same as the `runjava` function, except that it does not clear Java state remaining from previous use of Java in the session, prior to executing the current command line.

> **See Also:**
>
> "About Using the Command-Line Interface"

A.34 set_property

```
FUNCTION set_property(name VARCHAR2, value VARCHAR2) RETURN VARCHAR2
```

Establishes a value for a system property that is then used for the duration of the current RDBMS session, whenever a Java session is initialized.

> **Note:**
>
> In order to execute the `SET_PROPERTY` function, a user must have write permission on `SYS:java.util.PropertyPermission` for the property name. You can grant this permission using the following command:
>
> ```
> call dbms_java.grant_permission( '<user_name>',
> 'SYS:java.util.PropertyPermission', '<property_name>', 'write' );
> ```

> **See Also:**
>
> "About Setting System Properties"

A.35 get_property

```
FUNCTION get_property(name VARCHAR2) RETURN VARCHAR2
```

Returns any value previously established by `set_property`.

> **See Also:**
>
> "About Setting System Properties"

A.36 remove_property

```
FUNCTION remove_property(name VARCHAR2) RETURN VARCHAR2
```

Removes any value previously established by `set_property`.

> **Note:**
>
> In order to execute the `remove_property` function, a user must have write permission on `SYS:java.util.PropertyPermission` for the property name. You can grant this permission using the following command:
>
> ```
> call dbms_java.grant_permission( '<user_name>',
> 'SYS:java.util.PropertyPermission', '<property_name>', 'write' );
> ```

> **See Also:**
>
> "About Setting System Properties"

A.37 show_property

```
FUNCTION show_property(name VARCHAR2) RETURN VARCHAR2
```

Displays a message of the form `name = value` for the input name, or for all established property bindings, if name is null.

> **See Also:**
>
> "About Setting System Properties"

A.38 set_output_to_sql

```
FUNCTION set_output_to_sql (id VARCHAR2,
stmt VARCHAR2,
bindings VARCHAR2,
no_newline_stmt VARCHAR2 default null,
no_newline_bindings VARCHAR2 default null,
newline_only_stmt VARCHAR2 default null,
newline_only_bindings VARCHAR2 default null,
maximum_line_segment_length NUMBER default 0,
allow_replace NUMBER default 1,
from_stdout NUMBER default 1,
from_stderr NUMBER default 1,
include_newlines NUMBER default 0,
eager NUMBER default 0) RETURN VARCHAR2
```

Defines a named output specification that constitutes an instruction for executing a SQL statement, whenever output to the default `System.out` and `System.err` streams occurs.

Valid commands for SQL statement arguments start with one of the following case-insensitive keywords, followed by a space or tab:

- `SELECT`
- `INSERT`
- `DELETE`
- `UPDATE`
- `CALL`

> **See Also:**
>
> "About Redirecting Output on the Server"

A.39 remove_output_to_sql

```
FUNCTION remove_output_to_sql (id VARCHAR2) RETURN VARCHAR2
```

Deletes a specification created by `set_output_to_sql`.

> **See Also:**
>
> "About Redirecting Output on the Server"

A.40 enable_output_to_sql

```
FUNCTION enable_output_to_sql (id VARCHAR2) RETURN VARCHAR2
```

Reenables a specification created by `set_output_to_sql` and subsequently disabled by `disable_output_to_sql`.

> **See Also:**
>
> "About Redirecting Output on the Server"

A.41 disable_output_to_sql

```
FUNCTION disable_output_to_sql (id VARCHAR2) RETURN VARCHAR2
```

Disables a specification created by `set_output_to_sql`.

> **See Also:**
>
> "About Redirecting Output on the Server"

A.42 query_output_to_sql

```
FUNCTION query_output_to_sql (id VARCHAR2) RETURN VARCHAR2
```

Returns a message describing a specification created by `set_output_to_sql`.

> **See Also:**
>
> "About Redirecting Output on the Server"

A.43 set_output_to_java

```
FUNCTION set_output_to_java (id VARCHAR2,
class_name VARCHAR2,
class_schema VARCHAR2,
method VARCHAR2,
```

```
bindings VARCHAR2,
no_newline_method VARCHAR2 default null,
no_newline_bindings VARCHAR2 default null,
newline_only_method VARCHAR2 default null,
newline_only_bindings VARCHAR2 default null,
maximum_line_segment_length NUMBER default 0,
allow_replace NUMBER default 1,
from_stdout NUMBER default 1,
from_stderr NUMBER default 1,
include_newlines NUMBER default 0,
eager NUMBER default 0,
initialization_statement VARCHAR2 default null,
finalization_statement VARCHAR2 default null)RETURN VARCHAR2
```

Defines a named output specification that constitutes an instruction for executing a Java method whenever output to the default `System.out` and `System.err` streams occurs.

Valid commands for SQL statement arguments start with one of the following case-insensitive keywords, followed by a space or tab:

- `SELECT`
- `INSERT`
- `DELETE`
- `UPDATE`
- `CALL`

> **See Also:**
>
> "About Redirecting Output on the Server"

A.44 remove_output_to_java

```
FUNCTION remove_output_to_java (id VARCHAR2) RETURN VARCHAR2
```

Deletes a specification created by `set_output_to_java`.

> **See Also:**
>
> "About Redirecting Output on the Server"

A.45 enable_output_to_java

```
FUNCTION enable_output_to_java (id VARCHAR2) RETURN VARCHAR2
```

Reenables a specification created by `set_output_to_java` and subsequently disabled by `disable_output_to_java`.

> **See Also:**
>
> "About Redirecting Output on the Server"

A.46 disable_output_to_java

```
FUNCTION disable_output_to_java (id VARCHAR2) RETURN VARCHAR2
```

Disables a specification created by `set_output_to_java`.

> **See Also:**
>
> "About Redirecting Output on the Server"

A.47 query_output_to_java

```
FUNCTION query_output_to_java (id VARCHAR2) RETURN VARCHAR2
```

Returns a message describing a specification created by `set_output_to_java`.

> **See Also:**
>
> "About Redirecting Output on the Server"

A.48 set_output_to_file

```
FUNCTION set_output_to_file (id VARCHAR2,
file_path VARCHAR2,
allow_replace NUMBER default 1,
from_stdout NUMBER default 1,
from_stderr NUMBER default 1) RETURN VARCHAR2
```

Defines a named output specification that constitutes an instruction to capture any output sent to the default `System.out` and

`System.err` streams and append it to a specified file.

> **See Also:**
>
> "About Redirecting Output on the Server"

A.49 remove_output_to_file

```
FUNCTION remove_output_to_file (id VARCHAR2) RETURN VARCHAR2
```

Deletes a specification created by `set_output_to_file`.

> **See Also:**
>
> "About Redirecting Output on the Server"

A.50 enable_output_to_file

```
FUNCTION enable_output_to_file (id VARCHAR2) RETURN VARCHAR2
```

Reenables a specification created by `set_output_to_file` and subsequently disabled by `disable_output_to_file`.

> **See Also:**
>
> "About Redirecting Output on the Server"

A.51 disable_output_to_file

```
FUNCTION disable_output_to_file (id VARCHAR2) RETURN VARCHAR2
```

Disables a specification created by `set_output_to_file`.

> **See Also:**
>
> "About Redirecting Output on the Server"

A.52 query_output_to_file

```
FUNCTION query_output_to_file (id VARCHAR2) RETURN VARCHAR2
```

Returns a message describing a specification created by `set_output_to_file`.

> **See Also:**
>
> "About Redirecting Output on the Server"

A.53 enable_output_to_trc

```
PROCEDURE enable_output_to_trc
```

Reenables printing the output to `System.out` and `System.err` in the `.trc` file that was disabled by the `disable_output_to_trc` procedure.

> **See Also:**
>
> "About Redirecting Output on the Server"

A.54 disable_output_to_trc

```
PROCEDURE disable_output_to_trc
```

Prevents output to `System.out` and `System.err` from appearing in the `.trc` file.

> **See Also:**
>
> "About Redirecting Output on the Server"

A.55 query_output_to_trc

```
FUNCTION query_output_to_trc RETURN VARCHAR2
```

Returns a value indicating whether printing output to `System.out` and `System.err` in the `.trc` file is currently enabled.

> **See Also:**
>
> "About Redirecting Output on the Server"

A.56 endsession

```
FUNCTION endsession RETURN VARCHAR2
```

Clears any Java session state remaining from previous execution of Java in the current RDBMS session.

> **See Also:**
>
> "Two-Tier Duration for Java Session State"

A.57 endsession_and_related_state

```
FUNCTION endsession_and_related_state RETURN VARCHAR2
```

Clears any Java session state remaining from previous execution of Java in the current RDBMS session and all supporting data related to running Java.

> **See Also:**
>
> "Two-Tier Duration for Java Session State"

A.58 set_native_compiler_option

```
PROCEDURE set_native_compiler_option(optionName VARCHAR2,
value VARCHAR2)
```

Sets a native-compiler option to the specified value for the current schema.

> **See Also:**
>
> "Oracle JVM Just-in-Time Compiler (JIT)"

A.59 unset_native_compiler_option

```
PROCEDURE unset_native_compiler_option(optionName VARCHAR2,
value VARCHAR2)
```

Unsets a native-compiler option/value pair for the current schema.

> **See Also:**
>
> "Oracle JVM Just-in-Time Compiler (JIT)"

A.60 compile_class

```
FUNCTION compile_class(classname VARCHAR2) RETURN NUMBER
```

Compiles all methods defined by the class that is identified by *classname* in the current schema.

> **See Also:**
>
> "Oracle JVM Just-in-Time Compiler (JIT)"

A.61 uncompile_class

```
FUNCTION uncompile_class(classname VARCHAR2,
permanentp NUMBER default 0) RETURN NUMBER
```

Uncompiles all methods defined by the class that is identified by *classname* in the current schema.

> **See Also:**
>
> "Oracle JVM Just-in-Time Compiler (JIT)"

A.62 compile_method

```
FUNCTION compile_method(classname  VARCHAR2,
methodname VARCHAR2,
methodsig  VARCHAR2) RETURN NUMBER
```

Compiles the method specified by *name* and *Java type* signatures defined by the class, which is identified by *classname* in the current schema.

> **See Also:**
>
> "Oracle JVM Just-in-Time Compiler (JIT)"

A.63 uncompile_method

```
FUNCTION uncompile_method(classname  VARCHAR2,
methodname VARCHAR2,
methodsig  VARCHAR2,
permanentp NUMBER default 0) RETURN NUMBER
```

Uncompiles the method specified by the *name* and *Java type* signatures defined by the class that is identified by *classname* in the current schema.

> **See Also:**
>
> "Oracle JVM Just-in-Time Compiler (JIT)"

A.64 start_jmx_agent

```
PROCEDURE start_jmx_agent(port VARCHAR2 default NULL,
ssl  VARCHAR2 default NULL,
auth VARCHAR2 default NULL)
```

Starts the JMX agent in a specific session. Generally, the agent remains active for the duration of the session.

> **See Also:**
>
> "Managing Your Applications Using JMX"

A.65 set_runtime_exec_credentials

```
PROCEDURE set_runtime_exec_credentials(dbuser VARCHAR2,
osuser VARCHAR2,
ospass VARCHAR2)
```

where, `dbuser` is the name of a database user or a schema name and `osuser`, `ospass` are OS account credentials.

Associates the database user/schema `dbuser` with the `osuser/ospass` operating system (OS) credential pair. This association is encrypted and stored in a table owned by the `SYS` user. Once the new and valid association is established, every new OS process forked by the `java.lang.Runtime.exec` methods or every `ProceessBuilder` invoked by `dbuser` to run an OS command runs as the UID `osuser`, and not as the OS ID of the Oracle process. That is, the UID bits of the forked process are set to UID `osuser`.

> **Note:**
>
> DBAs and security administrators can use this procedure to tighten security of Java applications deployed to Oracle Database. By specifying lesser-privileged accounts, a DBA can limit the power and access rights of spawned processes as appropriate. You must be the `SYS` user to use the `set_runtime_exec_credentials` procedure, otherwise the `ORA-01031: insufficient privileges` error is raised. Use of invalid account credentials results in an `IOException`, when a new process is created.

Following examples show how to use this procedure:

Example 1

The following command binds user/schema `DBUSER` to credentials `osuser`/`ospass`:

```
dbms_java.set_runtime_exec_credentials('DBUSER', 'osuser', 'ospass');
```

Example 2

Either of the following commands unbinds the association of `DBUSER` and credentials `osuser`/`ospass`:

```
dbms_java.set_runtime_exec_credentials('DBUSER', '', '');

dbms_java.set_runtime_exec_credentials('DBUSER', null, null);
```

> **Note:**
>
> To use the `set_runtime_exec_credentials` procedure, you must configure the Oracle `jssu` facility to setuid `root` during oracle product installation, otherwise the `process spawn via jssu failed...` IOException may be raised at process creation time.

> **See Also:**
>
> "Secure Use of Runtime.exec Functionality in Oracle Database"