# Executing a Code Object

Only in DbVisualizer Pro

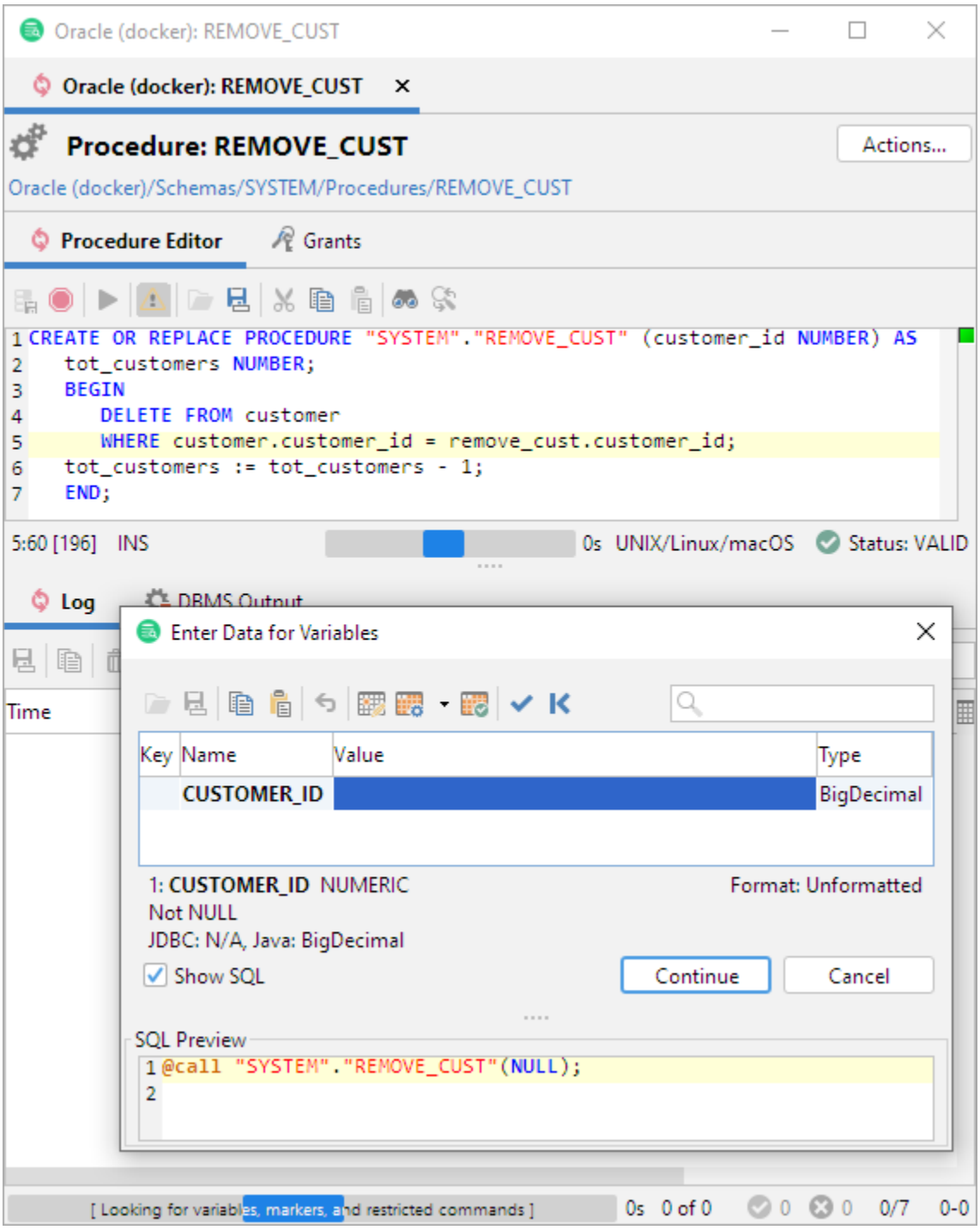
This feature is only available in the DbVisualizer Pro edition.

You can execute a code object, such as a function or stored procedure, either in the Code Editor or in the SQL Commander.



## Executing in the Code Editor

In the Code Editor, click the **Execute** button. DbVisualizer then generates a script for executing the code object, using variables for all parameters, and executes it.




Since the script contains variables, the **Variable Prompt** dialog pops up. Enter values for all parameters and click **Continue** to execute the procedure. The result is shown in the results area below the editor.

In the example shown in the figure, all parameters are input parameters but DbVisualizer also supports the execution of procedures with output parameters and functions returning a value. In this case, the generated script includes `@echo` statements to write the result in the **Log** tab. Please see below for more details.

## Executing in the SQL Commander

The scripts generated and executed by the Code Editor can also be included in a script and executed in an SQL Commander. Here's an example of a script calling a function and writing the result in the **SQL Commander Log** tab:

```
@call ${STATUS||(null)||String||noshow dir=out}$ = "HR"."GET_STATUS"(1002);
@echo STATUS: ${STATUS}$;
```

In this example, the result value from the GET_STATUS function is assigned to a variable named STATUS. Note that is has an option `dir=out`. This is a requirement for a variable that is assigned a value at runtime, whether it is used for a return value from a function call or for an output parameter in a procedure call. It also has the `noshow` option, to avoid getting prompted for a value for the variable. The value of the STATUS variable is then written to the log using the `@echo` command.

You can also use the output from one function or procedure as input to another, or even as a value in a SELECT or other SQL statement:

```
@call ${STATUS||(null)||String||noshow dir=out}$ = "HR"."GET_STATUS"(1002);
@call "HR"."UPDATE_STATUS"(1000, 2000, ${STATUS||||String||noshow dir=in}$);
```

Note that `dir=in` is specified for the STATUS variable when it is used in the UPDATE_STATUS procedure call. When you use a variable first for output and then as input with another `@call` command, you must change the direction option like this.

More formally, the `@call` command has this syntax when calling a function:

```
@call <OutVariable> = <FunctionName>(<ParamList>)
```

where the <FunctionName> may need to be fully qualified with a schema (and/or catalog/database) and the <ParamList> is a comma separated list of literal values or variables. Here's an example:

```
@call ${return_value||(null)||String||dir=out noshow}$ = get_some_value();
```

For a procedure, use this syntax:

```
@call <ProcedureName>(<ParamList)
```

where the <ProcedureName> may need to be fully qualified with a schema (and/or catalog/database) and the <ParamList> is a comma separated list of literal values or variables. Here's an example:

```
@call my_process('literal input',
                 ${var_in||(null)||String||dir=in}$,
                 ${var_out||(null)||String||dir=out noshow}$,
                 ${var_inout||'in_value'||String||dir=inout}$);
```

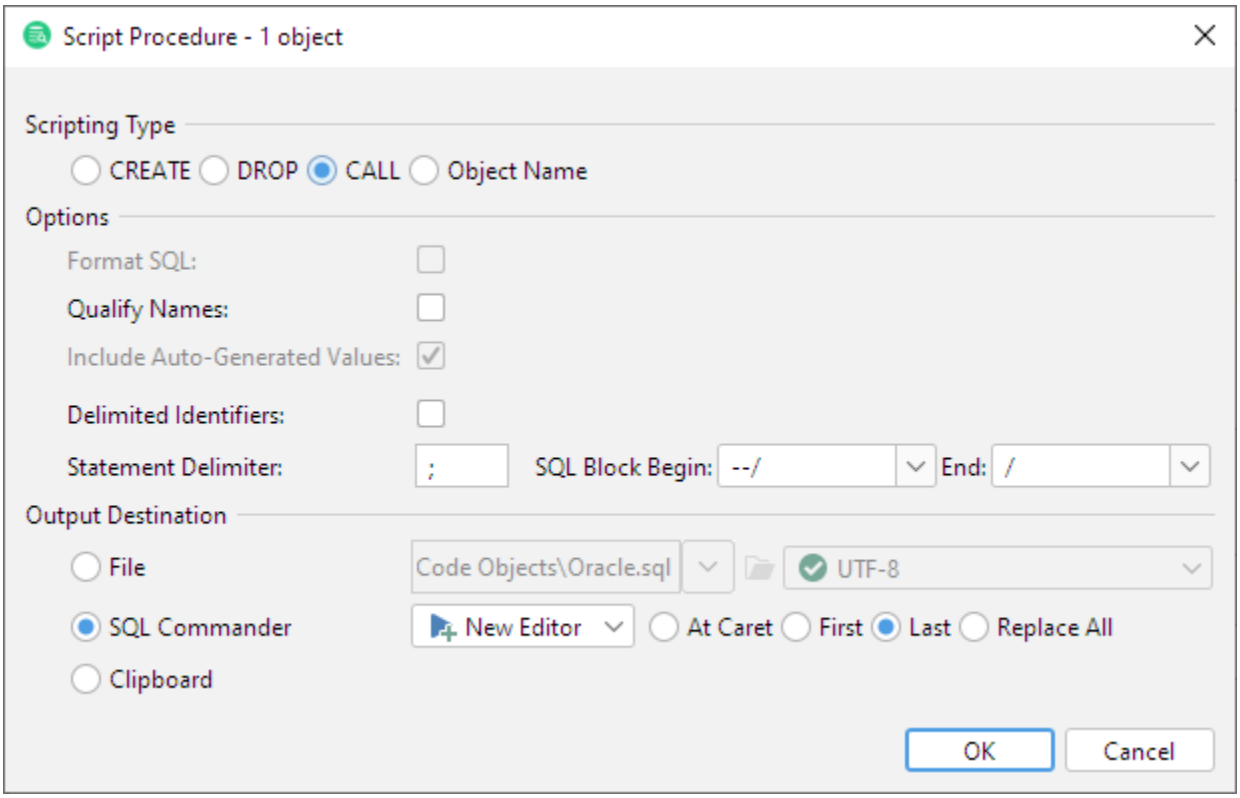
As shown in these examples, you must use the `dir` option to specify how the variable is to be used (in, out or inout) and you may use the `noshow` option to prevent being prompted for a value for an output variable.

You can use the @echo command to write the value assigned to an output variable to the log.

## Using the Script Object Dialog

Instead of writing a `@call` script by hand in an SQL Commander, you can use the Script Object (e.g. **Script Procedure** or **Script Function**) right-click menu choices for the object node in the tree.

This opens the Script Object dialog where you select that you want to generate a CALL script and can adjust settings for using delimiters and qualifiers, as well as the destination for the generated script. This is how the Script Dialog will look like when opened for a Procedure.


Script Procedure - 1 object
Scripting Type
CREATE
DROP
CALL
Object Name
Options
Format SQL:
Qualify Names:
Include Auto-Generated Values:
Delimited Identifiers:
Statement Delimiter:
;
SQL Block Begin:
--/
End:
/
Output Destination
File
Code Objects\Oracle.sql
UTF-8
SQL Commander
New Editor
At Caret
First
Last
Replace All
Clipboard
OK
Cancel