XML element - Commands

Only in DbVisualizer Pro

ⓘ This document and the Database Profile Framework in general is appropriate only when using the licensed DbVisualizer Pro edition.

The **Commands** element is a simple grouping element for **Command** elements.

- XML element - Command
  - Result Set
    - XML element - Input
    - XML element - Output

```
<Commands extends="true">

  <Command id="profile.xxx">
    ...
  </Command>

</Commands>
```

The **extends="true"** attribute specifies that the list of commands will extend the list of commands defined in the profile being extended.

XML element - Command

The main purpose with the **Command** element is to run a single **SQL statement** or a **script** of SQL statements. In most cases, the script should return a result set with 0 or multiple rows with the exception for actions which not necessarily need to return a result set, e.g., a "drop" action). The following show the command element, its attributes and valid sub elements.

```
<Command     id="sybase-ase.getLogins"
         method="dynamic"
       parsesql="true"
continueonerror="false">

   <SQL>
      ...
   </SQL>

   <Input>
      ...
   </Input>

   <Output>
      ...
   </Output>

   <Filter>
      ...
   </Filter>

</Command>
```

Attribute	Description
id	The command element is identified with a unique **id** attribute. This id is referred in ObjectsTreeDef, ObjectsViewDef and ObjectsActionDef definitions using the **idref** attribute. The id naming convention for command elements is to prefix with the name of the profile and a dot. Example **oracle.xxx**, **sqlserver.xxx**, and so on.
method	**dynamic** this is the default value and define that the SQL is a dynamic SQL statement as opposed to setting it to JDBC which defines that the SQL is really a JDBC meta data call rather than SQL, - jdbc See description for dynamic, - runBeforeConditionsEval this value is only considered if the command is define in the InitCommands section.

parsesql	The default behavior is that the SQL may contains multiple SQL statements each delimited by a semi colon (':'). Set this attribute to false to disable multiple SQL statements.
	If the SQL contain multiple SQL statements only one may produce a result set.
continueon error	This attribute is only valid in combination with **parseSQL="true"** and if the SQL contain multiple SQL statements. If one of SQLs fail execution will continue with the next. The **default value is false**.

The following command queries login information in Sybase ASE.

```
<Command id="sybase-ase.getLogins">
    <SQL>
        <![CDATA[
SELECT name "Name", suid "SUID", dbname "Default Database", fullname "Full Name",
language "Default Language", totcpu "CPU Time", totio "I/O Time", pwdate "Password Set"
FROM master.dbo.syslogins ORDER BY 1
        ]]>
    </SQL>
</Command>
```

The id for this command is **sybase-ase.getLogins**. The reason for prefixing the id with the name of the profile is that profiles can be extended and id's need to be unique.

This SQL example show a command with a **SELECT** statement using **column aliases**. If no aliases are specified the column names should be used to refer the data.
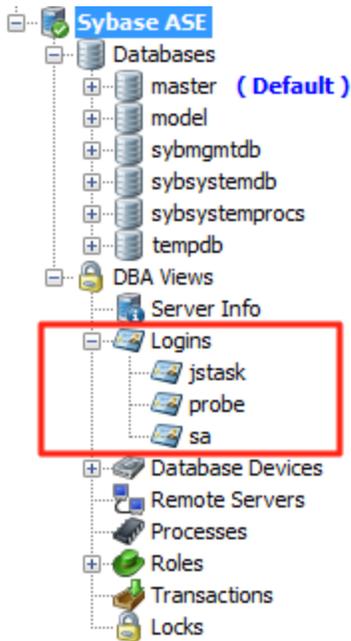
Result Set

This is the result set for the previous query:

Name	SUID	Default Database	Full Name	Default Language	CPU Time	I/O Time	Password Set
jstask	3	master	(null)	(null)	0	10	2009-12-22 09:53:50
probe	2	subsystemdb	(null)	(null)	0	0	2009-12-22 08:37:35
sa	1	master	(null)	(null)	182	168723	2009-12-22 08:36:54

How DbVisualizer handle the result set depends on whether the command is executed as a request in the database objects tree (ObjectsTreeDef) or in the object view (ObjectsViewDef). If executed in the database objects tree, each row in the result set will be represented by a node in the tree. If executed in the object view, it is the viewer component that decide how the result will be displayed.

Another important difference between the database objects tree and the object view is that the tree is a hierarchical structure of objects while the object view presents information about a specific object. An object that is inserted in the database objects tree is a 1..1 mapping to a row from the result set. The end user will see these objects (nodes) by some descriptive label, as defined in the ObjectsTreeDef. All data for the row from the original result set is stored with the object in the tree and may be used in the **label**, **variables**, **conditions**, etc. This is not the case in the ObjectsViewDef.

The following example put some light on this. Consider the previous result set and that it is used to create objects in the database objects tree. The end user will see the following in DbVisualizer. (The label for each row is the name column in the result set.):

Each of the **jstask**, **probe** and **sa** nodes have all their respective data from the result set associated with the nodes. The data is referenced as **commandId .columnName**, i.e., **sybase-ase.getLogins.Name**, **sybase-ase.getLogins.Default Database,** etc. All associated data for the **sa** node in the example is listed next:

```
sybase-ase.getLogins.Name = sa
sybase-ase.getLogins.suid = 1
sybase-ase.getLogins.Default Database = master
sybase-ase.getLogins.Full Name = (null)
sybase-ase.getLogins.Default Language = (null)
sybase-ase.getLogins.CPU Time = 182
sybase-ase.getLogins.I/O Time = 168716
sybase-ase.getLogins.Password Set = 2009-12-22 08:36:54.576
```

The DataNode element definition presenting **jstask**, **probe** and **sa** nodes in the previous screenshot use the associated data for the **label** as follows:

```
<DataNode type="Login" label="${sybasease.getLogins.Name}" isLeaf="true">
   <SetVar name="objectname" value="${sybasease.getLogins.Name}"/>
   <Command idref="sybasease.getLogins">
      <Output id="sybasease.getLogins.Name" index="1"/>
      <Output id="sybasease.getLogins.suid" index="2"/>
   </Command>
</DataNode>
```

XML element - Input
The **Input** sub element for a Command is only used when a command is being referred with the **idref** attribute in any of ObjectsActionDef, ObjectsTreeDef or ObjectsViewDef. It has no effect specifying it for a Command in the Commands section.

There are two types of commands, with and without dynamic input. The difference is that dynamic commands accepts input data that is typically used to form the **WHERE** clause in SELECT SQLs. The previous example illustrates a static SELECT statement (without dynamic data).

To allow for dynamic input, just add variables at the positions (can be anywhere) in the SQL statement that should be replaced with dynamic values. The following is an extension of the previous example that allows for dynamic input.

```
<Command id="sybase-ase.getLogins">
   <SQL>
      <![CDATA[
SELECT name "Name", suid "SUID", dbname "Default Database", fullname "Full Name",
language "Default Language", totcpu "CPU Time", totio "I/O Time", pwdate "Password Set"
FROM master.dbo.syslogins WHERE name = '${name}' and suid = '${suid}' ORDER BY 1
      ]]>
   </SQL>
</Command>
```

This example add two input variables: **${name}** and **${suid}**. Values for these variables should then be supplied wherever the command is referred for execution via the Input element.

The following is an example from the ObjectsTreeDef that specify the **Input** sub elements to **map values** to the variables defined in the SQL.

```
<GroupNode type="Logins" label="Logins">
  <DataNode type="Login" label="${sybase-ase.getLogins.Name} isLeaf="true">
    <SetVar name="objectname" value="${sybase-ase.getLogins.Name}">
    <Command idref="sybase-ase.getLogins">
      <Input name="name" value="sa">
      <Input name="suid" value="${sybase-ase.getProcesses.suid}">
    </Command>
  </DataNode>
</GroupNode>
```

(Note that the Command element refer the command via the **idref** attribute which is then matched with the corresponding **id** for the Command).

The **${name}** variable in the SQL will be replaced with string **sa**.

The value for the **${suid}** variable will in this case get the value of another variable, **sybase-ase.getProcesses.suid**. So where is this variable defined? As explained in the Result Set section, all the data for a row in the result set is associated with the corresponding node in the database objects tree. In addition, it is possible to use all the data kept by the node and even its parent nodes (as presented in the objects tree) in the input to commands. So to evaluate the **${sybase-ase.getProcesses.suid}** variable, DbVisualizer first look for the variable in the current node. If it doesn't exist, it continues to look through the parent nodes until it reaches the root, which is the **Connections** node in the objects tree. If the variable is not found, it will be set to the string representation for null, which is **(null)** by default. Whenever a matching variable is found, DbVisualizer use its value and stops searching.

XML element - Output

As mentioned earlier, a specific column value in a result set row is referenced by the name of the column and then prefixed by the command id. Sometimes this is not desirable and the **Output** definition can be used to change this behavior. The following identifies a column in the result set by its **index number**, starting from 1, and then force its name to be set to the **value** of the **id** attribute.

```
<Output id="sybase-ase.getLogins.Name" index="1">
<Output id="sybase-ase.getLogins.suid" index="2">
```

(The index attribute accepts either the name of the column or index number in the result set starting from the first column at index 1).

The Output element can also be used to **alter the structure** of columns in the result set by **adding**, **renaming** or **removing** columns.

```
<Output modelaction="add" index="THIS_IS_A_NEW_COLUMN" value="Rattle and Hum">
<Output modelaction="rename" index="2" name="PHONE">
<Output modelaction="drop" index="MOBILE_PHONE">
<Output modelaction="removeisnullrows" index="4">
<Output modelaction="removerowsifequalto" index="ORDINAL_POSITION" value="0"/>
```

modelaction attribute	Description
add	Adds a new column to all rows. The value attribute accepts variables using the ${...} notation
rename	Renames a column
drop	Drops the specified column
removeisnullrows	Removes the row if the value in the specified column is null
removerowsifequalto	Removes the row if the data in the specified column is equal to the specified value

The rename operation is primarily used when building a custom command that is supposed to be used by a viewer that requires predefined input by specific column names. Read more in the ObjectsViewDef section.