Extending a Database Profile

Only in DbVisualizer Pro

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

All database profiles must extend the **generic** database profile. The generic profile handles the very basic object types in a relational database such as **Tables**, **Columns**, **Indexes** and **Procedures**. Its implementation is based entirely on what the **JDBC** driver provide in terms of database meta data. Due to the tight connection between the generic profile and the JDBC driver, the generic profile can be used to access almost any database with a JDBC driver.

The selection on what database profile should be used is determined with the **Database Type** setting for the database connection. Some of the database types that can be picked have a dedicated database profile with extended support while others have not. For databases with no database profile available, the generic one is used. It is also possible to manually chose the generic profile in the **Connection Properties / Database Profiles** settings.

The most important area in the database profile as seen from the DbVisualizer user interface is the section describing the **database objects tree**. This is the browser or navigator showing database objects. This is also the place that connects **actions** used to operate on database objects and **views** (not database views) used to display detailed information.

This section of the users guide is mainly focused on extending an existing database profile (not the generic profile) rather than creating a completely new profile (which should extend the generic database profile).

Extending a database profile is not only about adding functionality to an existing profile but also the process of changing and removing existing definitions in any of the profiles that are extended.

Extending Commands

Extending the **Commands** and **InitCommands** elements is simple as every command should be uniquely identified. To add a **Command** just insert the new command.

```
   <Commands extends="true">
      <Command id="sample.getLoginSchema">
         <SQL>
            <![CDATA[
select '${schema}' as schema from dual
            ]]>
         </SQL>
      </Command>
   </Commands>
```

To override the definition of an existing command in the parent profile, just make sure the **id** of the new command match the id of the parent profile command. It will then be replaced.

Extending Database Objects Tree

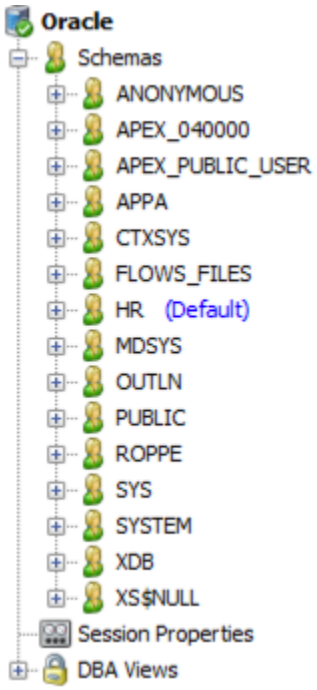
Extending or modifying the database objects tree (ObjectsTreeDef) require some attention since the modifications must match the exact object paths as defined in the parent profile. The object path is determined by the **GroupNode** and **DataNode** structure in the ObjectsTreeDef with the addition of the **type** attribute. The following is an example of the object path to the **Columns** sub node for a **Table** node:

```
GroupNode[@type='Schemas']/DataNode[@type='Schema']/GroupNode[@type='Tables']/DataNode[@type='Table']/GroupNode
[@type='Columns']
```

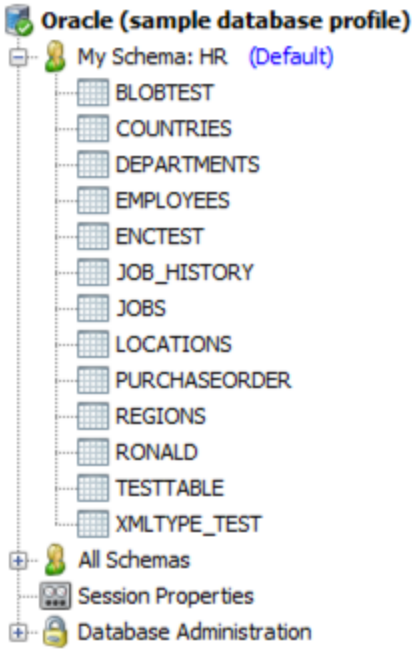
(The analyze database profile utility will report object paths in the above xpath format).

The hierarchy of GroupNode and DataNode is important when extending the database objects tree since the exact same hierarchy must be implemented in the extended profile. This also involve any conditional elements such as If/Else/ElseIf that are used in the parent profile.

Consider the following example showing the objects tree (for Oracle) with the **Schemas** node being expanded to show all schemas in the database:



Instead of showing all schema objects in the database we want to adjust so that only the default schema is displayed at the top level (below the Oracle database connection node). The default schema node should in addition only show **table** objects rather than all 20 (or so) object types being displayed in the standard Oracle database profile.



The previous screenshot shows the new **My Schema: HR** node at the top while the **Schemas** node has been renamed **All Schemas**. To accomplish the above a custom database profile has been created in the `${dbvis.prefsdir}/ext/profiles/sample-ext-oracle.xml` file with the following content required for the **ObjectsTreeDef** definition:

```
   <!--Commands used in this profile-->
   <Commands extends="true">
      <Command id="sample.getLoginSchema">
         <SQL>
            <![CDATA[
select '${schema}' as schema from dual
            ]]>
         </SQL>
      </Command>
   </Commands>

   <ObjectsTreeDef extends="true">
      <!--The following "Schema" definition shows the login schema directly below-->
      <!--the Database Connection for fast access. It is limited to only show-->
      <!--tables (by setting the "Table" DataNode to isLeaf="true")-->
      <DataNode type="Schema" label="My Schema: ${sample.getLoginSchema.SCHEMA}"
                icon="MySchema" order-before="0">
         <SetVar name="schema" value="${sample.getLoginSchema.SCHEMA}"/>
         <Command idref="sample.getLoginSchema">
            <Input name="schema" value="${#db.loginSchema}"/>
         </Command>
         <DataNode type="Table" label="${getTables.TABLE_NAME}"
                   sort="getTables.TABLE_NAME" isLeaf="true">
            <SetVar name="objectname" value="${getTables.TABLE_NAME}"/>
            <SetVar name="rowcount" value="true"/>
            <SetVar name="acceptInQB" value="true"/>
            <Command idref="oracle.getTables">
               <Input name="owner" value="${schema}"/>
               <Output id="getTables.TABLE_SCHEM" index="1"/>
               <Output id="getTables.TABLE_NAME" index="2"/>
               <Filter type="Table" name="Table">
                  <Column index="TABLE_NAME" name="Name"/>
               </Filter>
            </Command>
            <!--These are needed for the viewers defined in the parent profile-->
            <!--associated with the "Table" type-->
            <SetVar name="theTableName" value="${objectname}"/>
            <SetVar name="theParentName" value="${objectname}"/>
            <SetVar name="triggersCondition"
                    value="and table_name = '${theTableName}'"/>
         </DataNode>
      </DataNode>

      <!--Renaming the standard Schemas node to "All Schemas"-->
      <GroupNode type="Schemas" label="All Schemas"/>
   </ObjectsTreeDef>
```

In the **Commands** section there is a new **Command** that run a dummy SQL SELECT only to create a result set containing a single row/column with the value of the **${schema}** variable. The value for the **${schema}** variable is provided in the Command element for **DataNode type="Schema"** using the **${#db.loginSchema}** variable value as input. This variable is maintained by DbVisualizer and contain the login schema as specified in the connection setup. For Oracle this is the **userid**.

```
         <Command idref="sample.getLoginSchema">
            <Input name="schema" value="${#db.loginSchema}"/>
         </Command>
```

The command above is used to present the default schema as in the following **DataNode** declaration.

```
<DataNode type="Schema" label="My Schema: ${sample.getLoginSchema.SCHEMA}"
          icon="MySchema" order-before="0">
   <SetVar name="schema" value="${sample.getLoginSchema.SCHEMA}"/>
   <Command idref="sample.getLoginSchema">
      <Input name="schema" value="${#db.loginSchema}"/>
   </Command>
   <DataNode type="Table">
      ...
   </DataNode>
</DataNode>
```

The label for this Schema type is **label="My Schema: ${sample.getLoginSchema.SCHEMA}"**. The **${sample.getLoginSchema.SCHEMA}** variable name consists of two parts, the name of the command: **sample.getLoginSchema** and the column name: **SCHEMA** in the result set the produced by the command.

As a sub node to the **My Schema** node there is the **DataNode type="Table"** definition for the Table object type. The complete declaration for the Table element and its sub elements has been copied from the parent profile:

```
   <DataNode type="Table" label="${getTables.TABLE_NAME}"
             sort="getTables.TABLE_NAME" isLeaf="true">
      <SetVar name="objectname" value="${getTables.TABLE_NAME}"/>
      <SetVar name="rowcount" value="true"/>
      <SetVar name="acceptInQB" value="true"/>
      <Command idref="oracle.getTables">
         <Input name="owner" value="${schema}"/>
         <Output id="getTables.TABLE_SCHEM" index="1"/>
         <Output id="getTables.TABLE_NAME" index="2"/>
         <Filter type="Table" name="Table">
            <Column index="TABLE_NAME" name="Name"/>
         </Filter>
      </Command>
      <!--These are needed for the viewers defined in the parent profile-->
      <!--associated with the "Table" type-->
      <SetVar name="theTableName" value="${objectname}"/>
      <SetVar name="theParentName" value="${objectname}"/>
      <SetVar name="triggersCondition"
              value="and table_name = '${theTableName}'"/>
   </DataNode>
```

The Table objects in the extended profile should not show any sub nodes such as columns or triggers and these declarations are then removed in the copied DataNode for the Table object. The **isLeaf="true"** attribute specifies that there will be no child nodes.

The new **All Schemas** node in the extended profile is supposed to have the exact same content and definition as in the parent profile when it was called **Schemas**. The following definition will just rename the label of the existing node.

```
<GroupNode type="Schemas" label="All Schemas"/>
```

This example show adding a new **Role** node under all **DBA / Users / User** objects.

In the parent Oracle profile there are no child nodes below **User**. To handle this all nodes from **DBA** down to **User** must be specified (aka the object type path). The only requirement is that the type attribute is specified and that it match the type in the parent profile. In addition, this example specify the **label** attribute for some of the nodes just to show that **overridden attributes** will replace any parent equvivalent node attributes.

Only attributes for **GroupNode** and **DataNode** can be overridden. If you need to override for example a SetVar in a DataNode then all of the attributes in the DataNode and all its sub elements must be specified.

```
   <GroupNode type="DBA" label="Database Administration">
      <GroupNode type="Users">
         <!--The "User" type don't allow child nodes in the parent profile.-->
         <!--Setting isLeaf="false" is needed to override this and allow the-->
         <!--new "Role" child node-->
         <DataNode type="User" isLeaf="false">
            <!--Here comes the new child "Role" DataNode-->
            <DataNode type="Role" isLeaf="true"
                      label="${oracle.getGranteeRoles.GRANTED_ROLE} - ext">
               <SetVar name="objectname"
                       value="${oracle.getGranteeRoles.GRANTED_ROLE}"/>
               <Command idref="oracle.getGranteeRoles">
                  <Input name="grantee" value="${objectname}"/>
               </Command>
            </DataNode>
         </DataNode>
      </GroupNode>
   </GroupNode>
```

The following are specified only to redefine the position of the **Locks** and **Sessions** nodes. One of **order-before** and **order-after** attributes are used to either identify a type for which the node should be positioned before or after, or an index. The index is the fixed position or 0 which means first or a somewhat high number means last. The following will move the **Sessions** first among the **DBA** child nodes.

```
      <GroupNode type="Sessions" order-before="0"/>
      <!--The following will move the "Locks" node before "Sessions"-->
      <GroupNode type="Locks" order-before="Sessions"/>
```

Extending Actions

Extending ActionGroup and Action elements follow the same rules as for extending ObjectsTreeDef section. The following example show removing the **oracle-schema-stringsearch** action for the **Schema** object type in the parent profile. A new **ActionGroup**: **Extended Schema Actions** is added with a single new **Action**: **sample-schema-sample-action**.

```
<ObjectsActionDef extends="true">
   <ActionGroup type="Schema">
      <!--Remove action from parent profile for "Schema"-->
      <Action id="oracle-schema-stringsearch" action="drop"/>
      <!--Adds an "Extended Schema Actions" sub menu in the "Schema" actions menu-->
      <ActionGroup type="sample-schema-test" label="Extended Schema Actions">
         <!--Sample action that does nothing-->
         <Action id="sample-schema-sample-action" label="Sample Action"
                 reload="true" resetcatalogs="true" icon="remove">
            <Input label="Text Field" name="textField" style="text" editable="true"/>
            <Command>
               <SQL><![CDATA[Sample Action "${textField}"]]></SQL>
            </Command>
            <Confirm>
               Really run Sample Action "${textField}"?
            </Confirm>
            <Result>
               Sample Action "${textField}" processed!
            </Result>
         </Action>
      </ActionGroup>
   </ActionGroup>
</ObjectsActionDef
```

Extending Object Views

Extending **ActionGroup** and **Action** elements follow the same rules as for extending ObjectsTreeDef section.

```
   <ObjectsViewDef extends="true">
      <!--Schema is dropped in the Oracle profile. Redefine it here and show the-->
      <!--dictionary views. That data is really not associated with the single schema-->
      <!--defined in this profile but is a way to have it quickly accessed from-->
      <!--a single node.-->
      <ObjectView type="Schema">
         <DataView id="sample-schema-dict" label="Dictionary"
                   icon="sample-schema-dict" viewer="grid">
            <Command idref="sample.getDict"/>
            <Message>
               <![CDATA[
<html>
Simple viewer showing all dictionary tables with description. Easily accessed
by opening the Schema viewer since that is empty anyway in the
parent <b>oracle</b> profile.
</html>
               ]]>
            </Message>
         </DataView>
      </ObjectView>
   </ObjectsViewDef>
```

Remove an Element

Removing an object in the parent profile is easy, just add the **action="drop"** attribute to any of **GroupNode**, **DataNode**, **ObjectView**, **DataView**, **ActionGroup** and **Action** elements. If there are any sub elements for the object being dropped these are also removed.

Complete sample Database Profile

This document describe the different parts of a extended sample database profile for an Oracle database. Here follow the complete sample database profile.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE DatabaseProfile SYSTEM "dbvis-defs.dtd">

<DatabaseProfile desc="Sample profile extending the Oracle profile"
                 extends="oracle">

   <!--Commands used in this profile-->
   <Commands extends="true">
      <Command id="sample.getLoginSchema">
         <SQL>
            <![CDATA[
select '${schema}' as schema from dual
            ]]>
         </SQL>
      </Command>
      <Command id="sample.getDict">
         <SQL>
            <![CDATA[
select * from dict order by table_name
            ]]>
         </SQL>
      </Command>
   </Commands>

   <ObjectsActionDef extends="true">
      <ActionGroup type="Schema">
         <!--Remove action from parent profile for "Schema"-->
         <Action id="oracle-schema-stringsearch" action="drop"/>
         <!--Adds an "Extended Schema Actions" sub menu in the "Schema" actions menu-->
         <ActionGroup type="sample-schema-test" label="Extended Schema Actions">
            <!--Sample action that does nothing-->
            <Action id="sample-schema-sample-action" label="Sample Action"
                    reload="true" resetcatalogs="true" icon="remove">
               <Input label="Text Field" name="textField" style="text" editable="true"/>
               <Command>
```

```
            <SQL><![CDATA[Sample Action "${textField}"]]></SQL>
         </Command>
         <Confirm>
            Really run Sample Action "${textField}"?
         </Confirm>
         <Result>
            Sample Action "${textField}" processed!
         </Result>
      </Action>
   </ActionGroup>
</ActionGroup>
</ObjectsActionDef>

<ObjectsTreeDef extends="true">
   <!--The following "Schema" definition shows the login schema directly below-->
   <!--the Database Connection for fast access. It is limited to only show-->
   <!--tables (by setting the "Table" DataNode to isLeaf="true")-->
   <DataNode type="Schema" label="My Schema: ${sample.getLoginSchema.SCHEMA}"
             icon="MySchema" order-before="0">
      <SetVar name="schema" value="${sample.getLoginSchema.SCHEMA}"/>
      <Command idref="sample.getLoginSchema">
         <Input name="schema" value="${#db.loginSchema}"/>
      </Command>
      <DataNode type="Table" label="${getTables.TABLE_NAME}"
                sort="getTables.TABLE_NAME" isLeaf="true">
         <SetVar name="objectname" value="${getTables.TABLE_NAME}"/>
         <SetVar name="rowcount" value="true"/>
         <SetVar name="acceptInQB" value="true"/>
         <Command idref="oracle.getTables">
            <Input name="owner" value="${schema}"/>
            <Output id="getTables.TABLE_SCHEM" index="1"/>
            <Output id="getTables.TABLE_NAME" index="2"/>
            <Filter type="Table" name="Table">
               <Column index="TABLE_NAME" name="Name"/>
            </Filter>
         </Command>
         <!--These are needed for the viewers defined in the parent profile-->
         <!--associated with the "Table" type-->
         <SetVar name="theTableName" value="${objectname}"/>
         <SetVar name="theParentName" value="${objectname}"/>
         <SetVar name="triggersCondition"
                  value="and table_name = '${theTableName}'"/>
      </DataNode>
   </DataNode>

   <!--Renaming the standard Schemas node to "All Schemas"-->
   <GroupNode type="Schemas" label="All Schemas"/>

   <!--The main purpose with the following is to add a "Role" child DataNode -->
   <!--for each "User". In the parent Oracle profile there are no child-->
   <!--nodes below "User". To handle this all nodes from "DBA" down to "User"-->
   <!--must be specified (aka the object type path). The only requirement is that-->
   <!--the type attribute is specified and that it match the type in the parent profile.-->
   <!--In addition, this example specify the label attribute for some of the-->
   <!--nodes just to show that overridden attributes will replace any parent-->
   <!--equvivalent node attributes.-->
   <GroupNode type="DBA" label="Database Administration">
      <GroupNode type="Users">
         <!--The "User" type don't allow child nodes in the parent profile.-->
         <!--Setting isLeaf="false" is needed to override this and allow the-->
         <!--new "Role" child node-->
         <DataNode type="User" isLeaf="false">
            <!--Here comes the new child "Role" DataNode-->
            <DataNode type="Role" isLeaf="true"
                      label="${oracle.getGranteeRoles.GRANTED_ROLE} - ext">
               <SetVar name="objectname"
                       value="${oracle.getGranteeRoles.GRANTED_ROLE}"/>
               <Command idref="oracle.getGranteeRoles">
                  <Input name="grantee" value="${objectname}"/>
               </Command>
            </DataNode>
```

```
            </DataNode>
         </GroupNode>

         <!--The following are specified only to re-define the position of the-->
         <!--"Locks" and "Sessions" nodes. One of "order-before" and "order-after" -->
         <!--attributes are used to either identify a type for which the node should-->
         <!--be positioned before or after, or an index. The index is the fixed-->
         <!--position or 0 which means first or a somewhat high number means last.-->
         <!--The following will move the "Sessions" first among the "DBA"-->
         <!--child nodes-->
         <GroupNode type="Sessions" order-before="0"/>
         <!--The following will move the "Locks" node before "Sessions"-->
         <GroupNode type="Locks" order-before="Sessions"/>
      </GroupNode>
   </ObjectsTreeDef>

   <ObjectsViewDef extends="true">
      <!--Schema is dropped in the Oracle profile. Redefine it here and show the-->
      <!--dictionary views. That data is really not associated with the single schema-->
      <!--defined in this profile but is a way to have it quickly accessed from-->
      <!--a single node.-->
      <ObjectView type="Schema">
         <DataView id="sample-schema-dict" label="Dictionary"
                   icon="sample-schema-dict" viewer="grid">
            <Command idref="sample.getDict"/>
            <Message>
               <![CDATA[
<html>
Simple viewer showing all dictionary tables with description. Easily accessed
by opening the Schema viewer since that is empty anyway in the
parent <b>oracle</b> profile.
</html>
               ]]>
            </Message>
         </DataView>
      </ObjectView>
   </ObjectsViewDef>
</DatabaseProfile>
```