Obtaining information from descriptive data. Class Schema

Content

1.Tabular data sources

As already indicated there are two objects that are considered "tabular data sources" and whose behavior in terms of managing their collections of data are identical. These objects are Layer and Tables. The first represents the View *layer* and the second document the table type. Both objects contain records and the main difference between them as the data they contain is a record of a layer containing a field 'GEOMETRY'.

We have seen how to access objects and Layer Table both via functions currentTable () and currentLayer () and by the Project object method getTable and Object View getLayer by the method (), respectively.

Here we will focus on getting the definition or structure of the data and to obtain the records of collections.

2. The definition of the data. Schema object.

Defining tabular data from a source object is represented by the *Schema*. This object contains the definition of the data structure and the methods needed for it's management.

The main methods are

```
- modify()
- append(name, type[, size][, default][, precision])
- get(name[, default])
- getAttrNames()
- getCopy()
```

- modify (): Sets the edit mode to the data structure. Before adding or modify a property it's necessary to activate the edit mode.
- append (name, type [, size] [, default] [, precision]): Adds a property to the definition of the data. If the data definition is not in edit mode by using this method it will activate.
    - name, string: Name of the property
    - type, string: data type contained. The most common supported values are:
        - "BOOLEAN"
        - "DOUBLE"
        - "FLOAT"
        - "INTEGER"
        - "LONG"
        - "STRING"
        - "GEOMETRY"

      There are more usable types of data available, go check the documentation of the DataTypes.
    - size, int (optional): Size of the property. Determines the maximum size of data type string. Also used in the representation of the data.

    - default, object (optional): Default value of the property if you do not enter another.
    - Accuracy, int (optional): Sets the number of decimal data type, if this is numeric and has a decimal support.
- get (name, default): Returns the attribute descriptor (FeatureAttributeDescriptor) whose name is passed as a parameter if it exists in the definition. Otherwise returns the object that is passed as a parameter (default) or None in case it has been omitted.
    - name, string: name of the property
    - default, object (optional): default value returned if the property does not exist in the data definition.
- getAttrNames (): Returns a list of property names
- getCopy (): Returns a copy of the data definition

Get data definition from a data source

If we get the definition of a tabular data from a source, it is sufficient to invoke the method *getSchema* of the source.

```
schemaTable = currentTable().getSchema()
schemaLayer = currentLayer().getSchema()
```

These lines of code can not be together because only one of them will be the active element.

3. Change a data definition

Adding fields to a data definition. The append method

The *append* method of a data definition allows us to add a new attribute to the definition of data and defines its properties. If the data definition is not in edit mode by invoking this method it will directily activate.

```
schema.append(name, type[, size][, default][, precision])
```

- name, string: Name of the property
- type, string: data type contained. The most commont values supported are:
    - "BOOLEAN"
    - "DOUBLE"
    - "FLOAT"
    - "INTEGER"
    - "LONG"
    - "STRING"
    - "GEOMETRY"

There are more usable types of data available, go check the documentation of the DataTypes.

- size, int (optional): Size of the property. Determines the maximum size of the data type string. Also used in the representation of the data.

The default size attribute is 1, so it is advisable to indicate the size of the field.

- default, object (optional): Default value of the property if you do not enter another value.
- Accuracy, int (optional): Sets the number of decimal data type, in the case it is numeric and has decimal support.

The default value precision attribute is 4, so if you need a higher precision it is necessary to indicate it.

For example, if we want to add to a data definition a field called "ID" of type LONG

```
import  gvsig

def main():
    schema = gvsig.currentLayer().getSchema()
    schema.append("ID", "LONG", size=10)
    gvsig.currentLayer().edit()
    gvsig.currentLayer().updateSchema(schema)
    gvsig.currentLayer().commit()
```

The data container must be in edit mode, and for the changes to make effect in the data container the method updateSchema must be update it.

A complete example, we will add a new field to a layer that has 2 attributes called, *field1* and GEOMETRY

```
Jython Completion Shell
Jython 2.5.2 (Release_2_5_2:7206, Mar 2 2011, 23:12:06)
[Java HotSpot(TM) Server VM (Sun Microsystems Inc.)] on java1.6.0_26
>>>
>>> from gvsig import *
>>> schema = currentLayer().getSchema()
>>> schema.getAttrNames()
[u'campo1', u'GEOMETRY']
>>> schema.append('ID','INTEGER')
org.gvsig.fmap.dal.feature.impl.DefaultEditableFeatureAttributeDescriptor
@4f4458
>>> #The field you just added is in the data definition
>>> schema.getAttrNames()
[u'campo1', u'GEOMETRY', u'ID']
>>> #The field you just added is not in the Data Definition of
>>> #the layer
>>> currentLayer().getSchema().getAttrNames()
[u'campo1', u'GEOMETRY']
```

```
>>> #An error occurs because the layer is not in editing mode
>>> currentLayer().updateSchema(schema)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
  File "....
>>> currentLayer().edit()
>>> currentLayer().updateSchema(schema)
>>> currentLayer().getSchema().getAttrNames()
[u'campo1', u'GEOMETRY', u'ID']
>>> currentLayer().commit()
>>>
```

Remove fields from a data definition. The remove method.

This method allows us to remove one of the attributes of the data source. Unlike the method *append* the data definition must be in edit mode before invoking the remove method.

The data container must be in edit mode to make changes

The syntax is:

```
schema.remove(name)
```

- name, string: Name of the property that we want to eliminate

For example, we want to remove a field called "field1".

```
import  gvsig

def main():
    schema = gvsig.currentLayer().getSchema()
    schema.modify()     #The schema must be in editing mode
    schema.remove("campo1")
    gvsig.currentLayer().updateSchema(schema)
    gvsig.currentLayer().commit()
```

Following with the previous examples in the terminal, we want to remove the attribute named *field1*.

```
>>> currentLayer().isEditing()
False
>>> currentLayer().getSchema().getAttrNames()
[u'campo1', u'ID', u'GEOMETRY']
>>> #Can not delete the field because the schema is not in editing mode
>>> schema = currentLayer().getSchema()
>>> schema.remove('campo1')
False
>>> schema.modify()
>>> #Removes the field because the schema is in edition mode
>>> schema.remove('campo1')
org.gvsig.fmap.dal.feature.impl.DefaultEditableFeatureAttributeDescriptor
@c863dd
>>> #The layer does not reflect any changes
>>> currentLayer().getSchema().getAttrNames()
```

```
[u'campo1', u'ID', u'GEOMETRY']
>>> currentLayer().edit()
>>> #Layer updated
>>> currentLayer().updateSchema(schema)
>>> currentLayer().getSchema().getAttrNames()
[u'ID', u'GEOMETRY']
>>> #Editing the layer finished
>>> currentLayer().commit()
```

4. The object FeatureAttributeDescriptor

We have seen that the *get* Schema Object method returns an object FeatureAttributeDescriptor, this object contains information about one of the attributes of phenomena, such as the name, data type supported, or its accuracy.

The methods used are

```
- getDataTypeName()
- getGeomType()
- getIndex()
- getPrecision()
- getDefaultValue()
```

- getDataTypeName(): Returns the name of the attribute data type.
- getGeomType(): Returns the attribute type of the geometry, if it is a geometry.
- getIndex(): Returns the relative position of the attribute within the phenomenon..
- getPrecision(): For those attributes supports decimal data type, returns the maximum number of decimal places.
- GetDefaultValue (): Returns the default value of the property

gvSIG Association

Plaza Don Juan de Villarrasa 14-5,

46001, Valencia (Spain)

Registro Nacional de Asociaciones (National Register of Associations:): 596206

e-mail : info@gvsig.com

Web: www.gvsig.com

Project website: http://www.gvsig.org

Documentation made by Víctor Acevedo. Translated by Elisabet Adeva

Distribution Lists

There are three mailing lists in order to facilitate communication between all stakeholders in gvSIG project. The first two are for users and developers, mainly oriented to the Hispanic community, with the use of the Castilian lenguage preferred. The third list, is a international list, aimed at other communities where is preferd the use of English.

- **Members List.** . Here you can leave your opinions on the operation of the software: what things would you like to be developed, doubts related to the use of gvSIG and anything you believe should be in discussion and accommodates in the list of users. The link to subscribe to the user list is the following:
  http://listserv.gva.es/mailman/listinfo/gvsig_usuarios

- **Developers list.** It is geared/oriented to anyone interested in knowing how gvSIG is developed. The link to subscribe to this list is:
  http://listserv.gva.es/mailman/listinfo/gvsig_desarrolladores

- **International list.** It is intended for both users and developers witch do not speak Spanish. The preferably language used is English. The link to subscribe to this list is:
  http://listserv.gva.es/mailman/listinfo/gvsig_internacional