Funciones básicas

Índice de contenido

En gvSIG existe una clara jerarquía entre los distintos documentos que pueden formar un proyecto. De este modo, existe un *"Proyecto"* que a su vez tiene *Documentos* por ejemplo, *"Vistas"*, *"Tablas"*, *"Mapas"*, algunos de estos documentos a su vez pueden tener colecciones de otras cosas, por ejemplo una *Vista* puede tener múltiples capas, es decir, podría decirse, que los documentos que forman un proyecto puede decirse que tiene forma de árbol.

Una de las ventajas de la extensión de scripting en gvSIG-desktop 2.0 es que está completamente integrada en la plataforma, lo que permite interactuar directamente con los distintos documentos que usa gvSIG, sin embargo, puede ser complicado acceder a un elemento concreto del proyecto, sobretodo si hay varios del mismo tipo. Para facilitar la obtención de estos elementos existen funciones que nos permiten acceder fácil y rápidamente a los documentos activos del proyecto. Estas funciones son *currentProject*, *currentView*, *currentTable* y *currentLayer* que retornan respectivamente, el objeto *Proyecto*, la *Vista* activa del proyecto, la *Tabla* activa o la *Capa* activa de la vista.

1 El objeto Proyecto (Project). Función currentProject

El objeto que está más alto en la jerarquía es el *Proyecto*, y en gvSIG únicamente puede haber uno cargado a la vez. Podemos obtener el proyecto que tenemos cargado mediante la función *currentProject*.

La sintaxis es muy sencilla:

```
project = currentProject()
```

El objeto *project* tiene los siguientes métodos principales:

```
- getView( [name] )
- getTable( [name] )
- getProjectionCode()
```

- getView(name): Devuelve un documento de tipo *Vista* o None. Si se invoca con el parámetro name, busca entre las vistas y devuelve aquella cuyo nombre, en la Tabla de Contenidos (Table of Contents, TOC), coincida. Si no hay una vista cargada en el proyecto, devuelve None.
    - string name (opcional): Nombre de la vista en el gestor de proyactos
- getTable(name): Devuelve un documento de tipo *Table* o None. Si se invoca con el parámetro name, busca entre las tablas y devuelve aquella cuyo nombre, en el gestor de proyectos, coincida. Si no hay una tabla cargada en el proyecto, devuelve None.
    - string name (opcional) : Nombre de la Tabla
- getProjectionCode(): Devuelve un string con el código de la proyección del proyecto

Acceso a las vistas del proyecto

Como hemos mencionado, los distintos elementos que constituyen un proyecto están organizados de forma jerárquica en forma de árbol. De este modo, para obtener un elemento concreto debemos

preguntar a la rama superior por ese elemento.

Por ejemplo, supongamos que queremos obtener una vista distinta de la que tenemos activa en el proyecto. Para ello lo primero que debemos hacer es obtener el objeto que contiene las vistas, es decir el proyecto, y a continuación, una vez que tenemos el proyecto podemos pedir que nos de una vista concreta.

Como hemos visto anteriormente código para obtener una vista del proyecto sería:

```
project = currentProject()
vista = project.getView('nombreDeLaVista')
```

El método *getView* del proyecto nos permite obtener la vista que esté activa o una concreta que especifiquemos mediante su nombre. En el caso de que no encuentre una vista que coincida con el nombre devolverá *None,* siendo *nombreDeLaVista* una cadena de texto que coincida con el nombre de la vista con la que deseamos trabajar.

> *Observa que si invocamos el método sin pasar como parámetro el nombre de la vista que queramos, el comportamiento es el mismo que si invocamos a la función currentView, que se puede invocar sin necesidad de obtener el proyecto previamente.*

2 El objeto vista (View). Función currentView

Esta función nos permite obtener directamente la vista activa de nuestro proyecto, sin necesidad de estar buscando entre las vistas que tenga el proyecto cargadas.

La sintaxis es:

```
vista = currentView()
```

Los métodos principales del objeto *Vista* son:

```
- getLayer([name])
- getLayers()
- getGraphicsLayer()
```

- getLayer(name): Devuelve un documento de tipo *Layer* o None. Si se invoca con el parámetro name, busca entre las capas cargadas en la *Vista* y devuelve aquella cuyo nombre en el TOC, coincida. Si no hay una capa cargada en el proyecto, se produce un error.
    - string name (opcional): Nombre de la capa
- getLayers(): Devuelve una colección con las capas que haya en la *Vista*.
- getGraphicsLayer(): Devuelve una capa especial de la vista que se sitúa sobre ella
- getProjectionCode(): Devuelve un string con el código de la proyección de la vista

La capa especial *Graphics layer*

Graphics layer es una capa especial asociada a una vista de gvSIG. Esta capa vendría a ser como un acetato que se pinta sobre las capas que componen la vista. La capa *Graphics* se puede utilizar para pintar *cosas* sobre el resto y se accede mediante el método *getGraphicsLayer* de la vista. Los objetos graphics que se insertan en esta capa especial se agrupan mediante identificadores del grupo (groupId).

La sintaxis es:

```
vista = currentView()
vista.getGraphicsLayer()
```

Los métodos principales del objeto graphicsLayer son:

```
- removeGraphics(groupId)
- addSymbol(symbol)
- addGraphic(groupId, geom, idsym, label)
- getTypeVectorLayer()
```

- removeGraphics(groupId): Borra los objetos graphics que haya en el objeto graphicsLayer especificado mediante el parámetro groupId.
    - groupId, string: identifica el grupo al que se van a eliminar los objetos
- addSymbol(symbol): inserta el símbolo especificado como parámetro en el objeto graphicsLayer y devuelve un identificador del símbolo.
    - symbol, Symbol: Símbolo que se inserará en el graphicsLayer
- addGraphic(groupId, geom, idsym, label): Añade un graphic a la capa.
    - groupId, string: identifica el grupo al que se va a añadir el graphic
    - geom, Geometry: Geometría que se va a usar para añadir el graphic
    - idsym, int: Identificador, dentro del graphicsLayer, del símbolo que se va a usar (se obtiene del método addSymbol)
    - label, string: Etiqueta que se le añade al símbolo.
- getTypeVectorLayer(): Devuelve el tipo de geometría de la capa graphics.

Un uso correcto del objeto graphicsLayer implica que lo primero que debemos hacer es borrar los graphics que haya dentro del groupId que vayamos a utilizar. Después insertaremos los símbolos que queramos utilizar y por último añadiremos los graphics a la capa.

El siguiente código define una función que recibe como parámetros una geometría y un símbolo y los añade a la graphicsLayer de la vista activa.

```
def insertSymbol(geometry, symbol):
  graphicsLayer = currentView().getGraphicsLayer()
  graphicsLayer.removeGraphics("ejemplo")
  idSimbolo = graphicsLayer.addSymbol(symbol)
  graphicsLayer.addGraphic("ejemplo", geometry, idSimbolo, "Etiqueta")
```

Acceso a las capas de la vista

Como hemos visto podemos obtener la capa activa de la vista mediante la función *currentLayer*, pero si deseamos obtener una capa distinta debemos solicitarsela a la vista mediante el método *getLayer* pasándole como parámetro el nombre de la capa que deseamos obtener.

La sintaxis es:

```
vista.getLayer([ *nombreDeLaCapa* ])
```

Siendo *nombreDeLaCapa* una cadena de texto que coincida con el nombre de la capa que deseamos obtener.

Si invocamos el método sin parámetros, devolverá la capa activa, si hay

alguna, y None en caso contrario. Es decir, tiene el mismo comportamiento que la función currentLayer

Es posible también obtener una colección con todas las capas que tiene cargadas la vista a través del método *getLayers*.

La sintaxis es

```
layers = vista.getLayers()
```

Esta colección nos permite recorrer todas las capas de la vista y operar con ellas.

3 Obtener el documento Table activo . Función currentTable

Devuelve la *Tabla* activa del proyecto

La sintaxis es

```
table = currentTable()
```

4 Obtener la capa activa. Función currentLayer

Esta función permite obtener directamente la capa activa de nuestra vista.

La sintaxis es

```
layer = currentLayer()
```

5 Funciones de simbología. Creación de símbolos básicos

Estas funciones devuelven un símbolo básico que podemos instertar en el objeto graphicsLayer.

Estas funciones son:

```
- simplePointSymbol(color)
- simpleLineSymbol(color)
- simplePoligonSymbol(color)
```

- simplePointSymbol(color): Devuelve un símbolo básico de tipo punto del color especificado como parámetro.
- simpleLineSymbol(color): Devuelve un símbolo básico de tipo linea del color especificado como parámetro.
- simplePoligonSymbol(color): Devuelve un símbolo básico de tipo polígono del color especificado como parámetro.

El parámetro color puede ser o un string que se corresponde con el nombre de un color o un objeto java.awt.Color :

- color, string: Nombre del color. Los valores admitidos son: 'black', 'blue', 'cyan', 'darkGray', 'gray', 'green', 'lightGray', 'magenta', 'orange', 'pink', 'red', 'white', 'yellow'

6 Otras funciones

Función getCRS

Devuelve un objeto Projection a partir de un código. Si el código no es válido, devuelve None.

```
projection = getCRS(name)
```

- getCRS(name):
  - name, string: Código de la proyección que queremos obtener, por ejemplo: 'EPSG:23030', 'CRS:84'

Función getDataFolder

Devuelve la ruta en la que se encuentra la carpeta de datos definida en las preferencias de gvSIG (Preferencias/General/Carpetas). Si la carpeta no está definida devuelve None. La sintaxis es

```
path = getDataFolder()
```

Función getColorFromRGB

Devuelve un objeto java.awt.Color a partir de los valores rojo, verde, azul y alpha en un rango 0-255.

```
color = getColorFromRGB(r, g, b ,a)
```

- getColorFromRGB(r, g, b [,a]):
  - r, int: valor del rojo (0-255)
  - g, int: valor del verde (0-255)
  - b, int: valor del azul (0-255)
  - a, int (opcional): valor del alpha (0-255)

Función getDefaultColor

Devuelve el objeto java.awt.Color que se corresponde con el color por defecto del relleno definido en las preferencias de gvSIG (preferencias/Simbología).

```
color = getDefaultColor()
```

gvSIG Association

Plaza Don Juan de Villarrasa 14-5,

46001, Valencia (Spain)

Registro Nacional de Asociaciones: 596206

e-mail : info@gvsig.com

Web: www.gvsig.com

Web del proyecto: http://www.gvsig.org

Documentación realizada por Víctor Acevedo.

Listas de Distribución

Existen tres listas de distribución con el objeto de facilitar la comunicación entre todos los interesados en el proyecto gvSIG. Las dos primeras, la de usuarios y la de desarrolladores, están principalmente orientadas a la comunidad de habla hispana, siendo el castellano el idioma preferente a utilizar en las mismas. La tercera de ellas, lista internacional, está orientada principalmente al resto de comunidades y la lengua preferente a utilizar es la inglesa.

- **Lista de usuarios**. Aquí podéis hacer llegar vuestra opinión sobre el funcionamiento: qué cosas os gustaría que se desarrollaran, dudas en el uso de gvSIG y todo aquello que penséis que tiene cabida en una lista de usuarios. El enlace para la suscripción a la lista de usuarios es:
  http://listserv.gva.es/mailman/listinfo/gvsig_usuarios

- **Lista de desarrolladores**. Está orientada para todos los interesados en conocer cómo está desarrollado el gvSIG. El enlace para la suscripción a esta lista esa:
  http://listserv.gva.es/mailman/listinfo/gvsig_desarrolladores

- **Lista internacional**. Está orientada tanto para usuarios como para desarrolladores de habla no hispana. El idioma a utilizar es preferentemente inglés. El enlace para la suscripción a esta lista es:
  http://listserv.gva.es/mailman/listinfo/gvsig_internacional