GENERALITAT VALENCIANA
CONSELLERIA D'INFRAESTRUCTURES I TRANSPORT

IVER
TECNOLOGÍAS DE LA INFORMACIÓN

Guía de scripting Version 2 gvSIG 1.0

IVER - OFICINAS CENTRALES EN VALENCIA

C/ Salamanca nº 50-52 , 46005-VALENCIA

Telf: 902 25 25 40 - Fax: 96 316 27 16

E-Mail dac@iver.es www.iver.es

Conselleria de Infraestructuras y Transporte

C/ Blasco Ibáñez Nº 50 , 46010 VALENCIA

E-Mail gvsig@gva.es

Web del proyecto: http://www.gvsig.gva.es

Índice de contenido

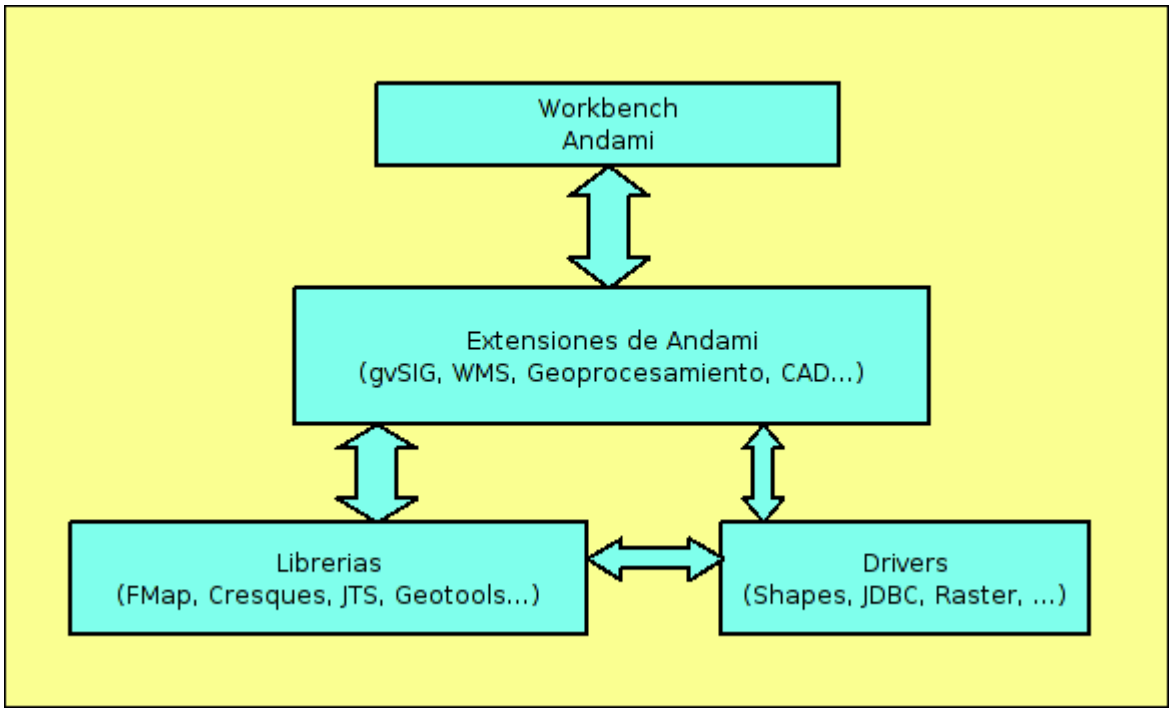
1 Introducción

El proyecto gvSIG se presenta como un framework que junto con el motor de scripting permiten dotarlo de nuevas funcionalidades sin un conocimiento amplio del core de la aplicación.

La aplicación gvSIG está construida a modo de capas que se integran usando unos mecanismos ya definidos llamados **extensiones**, a su vez, cada extensión puede definir sus propios puntos de extensión. Este modelo, permite a los desarrolladores añadir gran variedad de funcionalidades a la base de gvSIG, de forma que los artefactos de cada herramienta, como pueden ser los distintos tipos de capas, o botones, se presentan al usuario desde una plataforma común.

Los desarrolladores de extensiones también se benefician de esta arquitectura. El framework base de gvSIG les proporciona una serie de servicios de los cuales ellos no tienen que preocuparse, pudiéndose centrar en las tareas especificas de su extensión.

La figura siguiente presenta una visión simplificada del esto.



La plataforma de gvSIG esta conformada en su núcleo por tres subsistemas:

- **Andami**. Representa el esqueleto en el que se sustenta gvSIG. Es como un armazón en el que van encajando las distintas extensiones que conformarán la aplicación. Además de esto se encarga de presentar un entorno de usuario MDI, así como de dotar de aspecto a las ventanas de la aplicación.

- **Fmap**. Es el corazón SIG de la plataforma. Incluye todas las clases necesarias para manejar objetos SIG, así como drivers y adaptadores para utilizar los formatos más usados para el almacenamiento de los datos cartográficos. Dentro de esta librería encontramos clases para leer y escribir los formatos soportados, dibujar los mapas a las escalas adecuadas, asignar leyendas, definir simbologías, realizar búsquedas, consultas, análisis, etc.

- **gvSIG extensión**. Se trata de una extensión que contiene la parte de interface de usuario que presenta los datos geográficos manejados por **Fmap**. En este subsistema encontraremos las clases que implementan la mayor parte de cuadros de diálogo que utiliza la aplicación final, así como las clases de soporte a esos cuadros de diálogo. Por ejemplo, aquí se encuentran los formularios para asignar leyendas, crear mapas y vistas, definir escalas, etc.

1.1 Extensiones de Andami

Las extensiones de Andami se definen mediante un fichero xml que debe estar en una subcarpeta del directorio de gvSIG bin/gvSIG/extensiones. En este fichero de configuración se indican las clases que debe cargar Andami al arrancar la aplicación y se especifican también las opciones de menú que deben mostrarse así como los botones que aparecen en la barra de herramientas.

1.2 La librería de GUI para scripting

Thinlet es una librería gráfica de utilidades GUI. Separa la presentación gráfica y los métodos de la aplicación. Para la definición de la interface gráfica se utiliza un fichero XML con formato XUL.

Existe una herramienta llamada ThinG que permite diseñar el interface gráfico.

Para más información sobre Thinlet:

http://thinlet.sourceforge.net/

http://sjobic.club.fr/thinlet/scriptablethinlet/index.html

http://thing.sourceforge.net/

2 Ejemplos

gvSIG soporta varios lenguajes de programación para realizar los script.

Los ejemplos están realizados utilizando el lenguaje de programación Python versión 2.1 en su implementación para la maquina virtual de java (Jython).

Puede encontrarse información de Jython en www.jython.org

Puede encontrarse información de Python en www.python.org

2.1 Centrar la vista en un punto.

Vamos crear una extensión que nos permita especificar unas coordenadas para que se realice un centrado de la vista a las coordenadas introducidas y dibuje un punto en esa ubicación.

Para ello:

- Crearemos una carpeta “bin/gvSIG/extensiones/centrarVistaSobreUnPunto” en el directorio donde esté instalado gvSIG.

- Dentro de la carpeta crearemos un fichero llamado config.xml con el contenido:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<plugin-config>
	<libraries library-dir="../org.gvsig.scripting"/>
	<depends plugin-name="org.gvsig.scripting"/>
	<resourceBundle name="text"/>
	<extensions>
		<extension class-name="org.gvsig.scripting.ScriptingExtension"
			description="Extension de soporte para Scripts de usuario."
			active="true">

			<menu text="Archivo/Scripting/Centrar vista en un punto"
			  tooltip="Centrar la vista en un punto"
				action-command=""
				position="55"
				/>
		</extension>
	</extensions>
	</plugin-config>
```

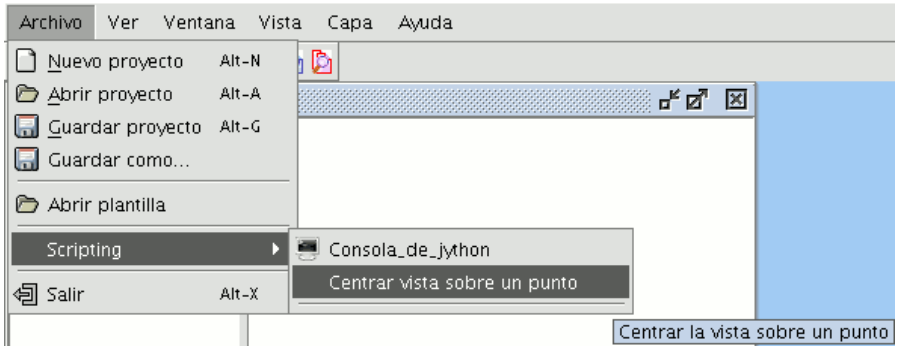
En el XML podemos encontrar los siguientes tags:

- *libraries library-dir*: Indica el directorio donde está alojadas las librerías de

scripting.

- *depends plugin-name*: Indica que la extensión necesita de otro plugin para funcionar.
- *resourceBundle name*: Indica donde está el fichero de traducciones.
- *extension class-name:* Es la clase que implementa el scripting dentro de gvSIG, el valor para esta tag debe ser ”org.gvsig.scripting.ScriptingExtension”.
- *description*: Descripción de la extensión.
- *active*: Indica si la extensión esta activa o no, debe estar siempre a true.
- *menu text:* En este tag se define donde y como se añade la entrada a la barra de menús. En este caso se creará en el menú Archivo/Scripting una entrada nueva con el nombre “Centrar la vista en un punto”. Propiedades que deben definirse en este tag son:
  - *tooltip:* Esta marca permite definir un texto que se mostrará cuando el puntero del ratón se posicione en la entrada del menú.
  - *action-command*: Admite dos tipos de acciones; show muestra una ventana, y run ejecuta un script. Tanto una como otra admiten varios parámetros que se explican más adelante.
  - *icon*: establece la ruta donde se encuentra el icono asociado a la entrada de menú que estamos creando. La ruta es relativa al directorio de la extensión. Si la ruta no es correcta la aplicación gvSIG no podrá ser iniciada.
  - *position*: Establece la posición en la que debe cargarse la entrada de menú en la barra de menús. Se pueden definir las posiciones entre 1 y 99 siendo 1 la primera en el menú y 99 la última. Si dos entradas del menú tienen la misma posición sólo se cargará una de ellas.

Una vez creado el fichero arrancamos gvSIG y comprobamos que se ha añadido la entrada en el menú.

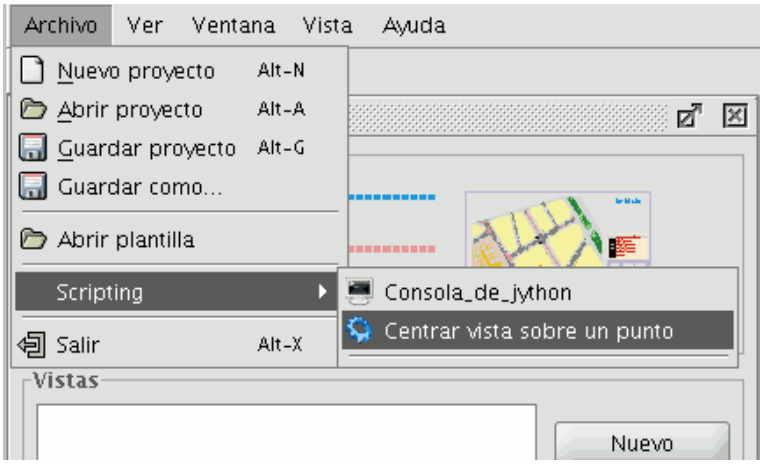


- En la carpeta que hemos creado anteriormente crearemos un directorio images y copiaremos en él un fichero que será el que utilicemos como icono. Este fichero se encuentra en “bin/gvSIG/extensiones/org.gvsig.scripting/images/default.png”.

- Añadimos en el config.xml un tag nuevo a la entrada del menú.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<plugin-config>
	<libraries library-dir="../org.gvsig.scripting"/>
	<depends plugin-name="org.gvsig.scripting"/>
	<resourceBundle name="text"/>
	<extensions>
		<extension class-name="org.gvsig.scripting.ScriptingExtension"
			description="Extension de soporte para Scripts de usuario."
			active="true">

			<menu text="Archivo/Scripting/Centrar vista en un punto"
			  tooltip="Centrar la vista en un punto"
				action-command=""
				icon="images/default.png"
				position="55"
				/>
		</extension>
	</extensions>
</plugin-config>
```

- Arrancamos la aplicación y vemos que aparece en el menú la opción junto con el icono.



- Crear el fichero centrarVistaSobreUnPunto.xml junto al config.xml recién creado, con el siguiente contenido.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<panel columns="3" gap="3">
    <label colspan="3" text="Coordenadas para centrar la vista"/>
    <label colspan="2" halign="right" text="x:"/>
    <textfield name="txtX"/>
    <label colspan="2" halign="right" text="y:"/>
    <textfield name="txtY"/>
    <panel colspan="3" gap="2" halign="right">
       <button halign="right" name="botAplicar" text="Aplicar"/>
       <button halign="right" name="botCerrar" text="Cerrar"/>
    </panel>
</panel>
```

  Este fichero centrarVistaSobreUnPunto.xml es un fichero que define una ventana que se va a mostrar cuando se pulse sobre la entrada del menú que hemos añadido.

- Modificamos config.xml y le añadimos la instrucción "**show**" al tag action-command.

  La instrucción **show** recibe como parámetros:

  filename: Indica donde está definida la ventana que debe mostrarse.

  language: Indica el lenguaje de script que se ha utilizado en el script. gvSIG en la actualidad soporta varios lenguajes de script aunque sólo está probado jython.

  title: Establece el título con el que se mostrará la ventana.

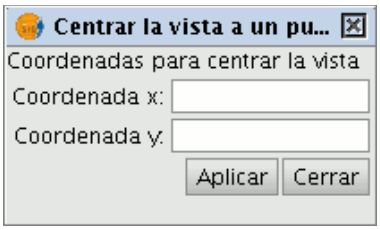
width: Ancho con el que se mostrará la ventana inicialmente.

height: Alto con el que se mostrará la ventana inicialmente.

El config.xml tendrá el siguiente contenido una vez añadido el tag.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<plugin-config>
	<libraries library-dir="../org.gvsig.scripting"/>
	<depends plugin-name="org.gvsig.scripting"/>
		<resourceBundle name="text"/>
	<extensions>
		<extension class-name="org.gvsig.scripting.ScriptingExtension"
			description="Extension de soporte para Scripts de usuario."
			active="true">
			<menu text="Archivo/Scripting/Centrar vista en un punto"
			  tooltip="Centrar la vista en un punto"
			      action-command="show(
fileName='gvSIG/extensiones/centrarVistaSobreUnPunto/centrarVistaSobreUnPunto.xml',
language='jython', title='Centrar a un punto', width=210, height=86)"
			      icon="images/default.png"
			      position="55"
			      />
		</extension>
	</extensions>
</plugin-config>
```

- Arrancar la aplicación y ver que al pulsar en la opción de menú que se ha añadido aparece la ventana.



- Añadimos una acción sobre el botón “Cerrar“ para ello en el tag correspondiente al botón incluimos la acción “**thinlet.closeWindow()"** en el fichero que define la ventana (en nuestro caso “centrarVistaSobreUnPunto.xml”).

  El objeto **thinlet** es la ventana de la extensión y permite el acceso a los controles.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<panel columns="3" gap="3">
    <label colspan="3" text="Coordenadas a centrar en la vista"/>
    <label colspan="2" halign="right" text="x:"/>
```

```
    <textfield name="txtX"/>
    <label colspan="2" halign="right" text="y:"/>
    <textfield name="txtY"/>
    <panel colspan="3" gap="2" halign="right">
        <button halign="right" name="botAplicar" text="Aplicar"/>
        <button halign="right" name="botCerrar" text="Cerrar"
action="thinlet.closeWindow()"/>
    </panel>
</panel>
```

Si se modifica algo en el fichero xml, no es preciso rearrancar la aplicación. Volvemos a ejecutar la opción del menú para ver que el botón “Cerrar” cierra la ventana.

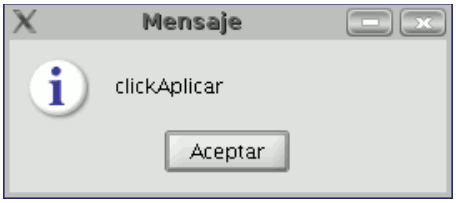
- Ahora vamos a añadir una acción sobre el botón “Aplicar”. Para esto crearemos un fichero “centrarVistaSobreUnPunto.py” junto al fichero “centrarVistaSobreUnPunto.xml” con el siguiente contenido:

```
from gvsiglib import *

def clickAplicar(thinlet):
    showMessageDialog("clickAplicar")
    return
```

Y el fichero centrarVistaSobreUnPunto.xml lo modificaríamos como sigue:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<panel columns="3" gap="3">
    <script language="jython" method="init" src="centrarVistaSobreUnPunto.py"/>
    <label colspan="3" text="Coordenadas a centrar en la vista"/>
    <label colspan="2" halign="right" text="x:"/>
    <textfield name="txtX"/>
    <label colspan="2" halign="right" text="y:"/>
    <textfield name="txtY"/>
    <panel colspan="3" gap="2" halign="right">
        <button halign="right" name="botAplicar" text="Aplicar"
action="clickAplicar(thinlet)"/>
        <button halign="right" name="botCerrar" text="Cerrar"
action="thinlet.closeWindow()"/>
    </panel>
</panel>
```

- Si volvemos a ejecutar la opción del menú y pulsamos en botón aplicar mostrará un mensaje.



- Modificamos el código de “centrarVistaSobreUnPunto.py” para recoger los valores de la ventana y centrar la vista. Debe quedar como se muestra a continuación:

```
import java.awt.geom.Point2D as Point2D
import java.awt.geom.Rectangle2D as Rectangle2D

from gvsiglib import *

mapContext = None

def getMapContext():
    """
    Devuelve el objeto mapContext asociado a la vista de gvSIG
    que tiene el foco en este momento.
    Si la ventana activa no es una vista retorna None
    """
    vista = gvSIG.getActiveDocument()
    if vista == None:
      print "No se puede acceder al documento activo."
      return None
    try:
      mapContext = vista.getModel().getMapContext()
    except Exception, e:
      print "El documento activo no parece ser una vista. Error %s %s" % (
str(e.__class__),
str(e)
)
      return None

    return mapContext

# Obtenemos el mapContext antes de que se muestre nuestra ventana
mapContext = getMapContext()

def clickAplicar(thinlet):
    global mapContext

    if mapContext == None:
      print "No se puede acceder al documento activo."
      return

    if mapContext.getLayers().getLayersCount() < 1:
      print "El documento activo no tiene capas disponibles."
```

```
      return

    # Accedemos a los controles de la ventana que
    # hemos definido para recoger las coordenadas x e y
    # a través del objeto thinlet.
    x = float(thinlet.getString(txtX, "text"))
    y = float(thinlet.getString(txtY, "text"))
    return centrarEnLasCoordenadas(mapContext, x,y)

def centrarEnLasCoordenadas(mapContext, x,y):
  try:
    oldExtent = mapContext.getViewPort().getAdjustedExtent()
    oldCenterX = oldExtent.getCenterX()
    oldCenterY = oldExtent.getCenterY()
    center=Point2D.Double(x,y)
    movX = x - oldCenterX
    movY = y - oldCenterY
    upperLeftCornerX = oldExtent.getMinX()+movX
    upperLeftCornerY = oldExtent.getMinY()+movY
    width = oldExtent.getWidth()
    height = oldExtent.getHeight()
    extent = Rectangle2D.Double(upperLeftCornerX, upperLeftCornerY, width, height)
    mapContext.getViewPort().setExtent(extent)
    return center
  except ValueError, e:
    print "Se ha producido un error realizando zoom a las coordenadas (%s,%s). Error  %s,
%s" % (
      repr(x),
      repr(y),
      str(e.__class__),
      str(e)
    )
    return None
```

- Para probarlo necesitaremos una vista con capas, por lo que podríamos añadir control para que si no está como documento activo una vista con capas no habilite el botón “Aplicar”. Para ello incluiremos al final del fichero el siguiente código:

```
def elDocumentoActivoEsUnaVistaValida():
    global mapContext

    if mapContext == None:
      print “El documento activo nop parece ser una vista”
      return False

    if mapContext.getLayers().getLayersCount() < 1:
      print "El documento activo no tiene capas disponibles."
      return False
    return True

if elDocumentoActivoEsUnaVistaValida():
```

```
  thinlet.setBoolean(botAplicar,"enabled",True)
else:
  thinlet.setBoolean(botAplicar,"enabled",False)
```

- Lo siguiente que se podría hacer es que pintase un punto en las coordenadas introducidas. Esto se consigue con la siguiente función:

```
def drawPoint(mapContext, center, color=None):
    """
    Esta función pintará un punto sobre la capa de gráficos
    asociada al mapContext.
    Todo mapContext además de las capas que tenga cargadas dispone
    una capa graphics sobre la que dibujar elementos gráficos.
    """

    if color == None:
        import java.awt.Color as Color
        color = Color.blue

    layer=mapContext.getGraphicsLayer()
    layer.clearAllGraphics()
    theSymbol = FSymbol(FConstant.SYMBOL_TYPE_POINT,color)
    idSymbol = layer.addSymbol(theSymbol)
    geom = ShapeFactory.createPoint2D(center.getX(),center.getY())
    theGraphic = FGraphic(geom, idSymbol)
    layer.addGraphic(theGraphic)
```

Y modificaríamos la función clicAplicar de la siguiente forma:

```
def clickAplicar(thinlet):
    vista = gvSIG.getActiveDocument()
    if vista == None:
      print "No se puede acceder al documento activo."
      return
    try:
      mapContext = vista.getModel().getMapContext()
    except:
      print "El documento activo no parece ser una vista."
      return

    if mapContext.getLayers().getLayersCount() < 1:
      print "El documento activo no tiene capas disponibles."
      return
    x = float(thinlet.getString(txtX, "text"))
    y = float(thinlet.getString(txtY, "text"))
    centro = centrarEnLasCoordenadas(mapContext, x,y)
    drawPoint(mapContext,centro)
```

- Una vez puesto el punto ... ¿cuando se borra el punto de la vista? Vamos a añadir

una opción de menú que limpie la capa de graphics, en esta ocasión vamos a utilizar el comando **run**.

- El comando **run** recibe como parámetros:

  fileName: Ruta relativa al directorio bin de gvSIG donde se encuentra el archivo con el código que debe ejecutarse.

  language: Lenguaje en el que está escrito el código que debe ejecutarse.

Dentro del fichero config.xml añadimos otra entrada de menú.

```
<menu text="Archivo/Scripting/Borrar puntos"
      tooltip="Borrar puntos"
      action-command=
"run(fileName='gvSIG/extensiones/centrarVistaSobreUnPunto/limpiarElGraphics.py',language='jy
thon')"
      icon="images/default.png"
      position="56"
        />
```

A continuación creamos un fichero llamado limpiarElGraphics.py con el siguiente contenido:

```
from gvsiglib import *

def main():
    vista = gvSIG.getActiveDocument()
    if vista == None:
      print "No se puede acceder al documento activo."
      return None
    try:
      mapContext = vista.getModel().getMapContext()
    except Exception, e:
      print "El documento activo no parece ser una vista."
      print "Error %s %s" % (str(e.__class__),str(e))
      return None
    if mapContext == None:
      return
    layer=mapContext.getGraphicsLayer()
    layer.clearAllGraphics()
    mapContext.invalidate()

main()
```

2.2 Pedir información de un punto.

En este ejemplo vamos a generar unas herramientas que nos permitan obtener la información asociada a un punto de una capa concreta cargada en una vista. Para esto crearemos una capa a partir de un fichero csv que será la que utilicemos en el ejemplo.

Crearemos dos herramientas nuevas, una que nos añada la capa con la que vamos a trabajar, y otra que implemente la operación de pedir información sobre la capa.

Para la primera herramienta partiremos de un fichero csv con el que crearemos una fuente de datos de gvSIG y a partir de esta fuente de datos generaremos una capa de eventos que cargaremos en la vista. A esta capa se le añadirá una propiedad para poder identificarla y poder presentar la ventana de información personalizada que crearemos para esa capa. La herramienta además debe controlar que la capa de trabajo que creamos no se añada a la vista más de una vez en caso de ejecutar la utilidad en repetidas ocasiones.

Para la herramienta de información personalizada tendremos que registrar un listener sobre el mapControl de la vista que reciba los eventos de clicado cuando se active la herramienta. La información que se recibe de un punto de una capa es una estructura xml por lo que se utiliza un parser de SAX para analizarla, esto implica que deberemos crear un SaxContentHandler para procesar el xml.

Además de realizar estas operaciones crearemos el interface de usuario adecuado a los datos de nuestra capa de trabajo.

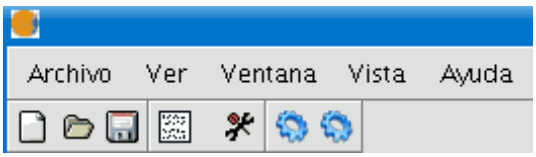
Para crear esta herramienta haremos:

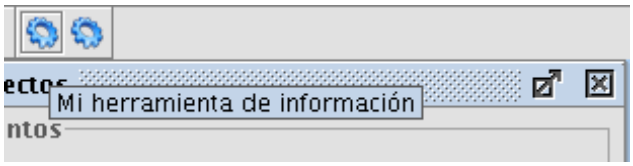
- Crearemos una carpeta “bin/gvSIG/extensiones/miHerramientaDeInformacion” en el directorio donde esté instalado gvSIG
- Una vez creada la carpeta “miHerramientaDeInformacion” crearemos dentro de ella otra carpeta que llamaremos “images”. En esta carpeta copiaremos el fichero “bin/gvSIG/extensiones/org.gvsig.scripting/images/default.png”.
- A continuación crearemos el fichero config.xml dentro de la carpeta “miHerramientaDeInformacion”. Este fichero es el encargado de definir los elementos que deben incluirse en la barra de menús y en la barra de herramientas. El contenido de este fichero es:

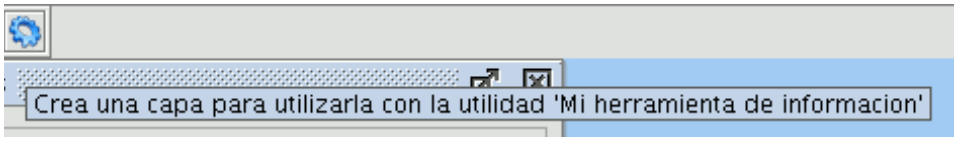
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<plugin-config>
        <libraries library-dir="../org.gvsig.scripting"/>
```

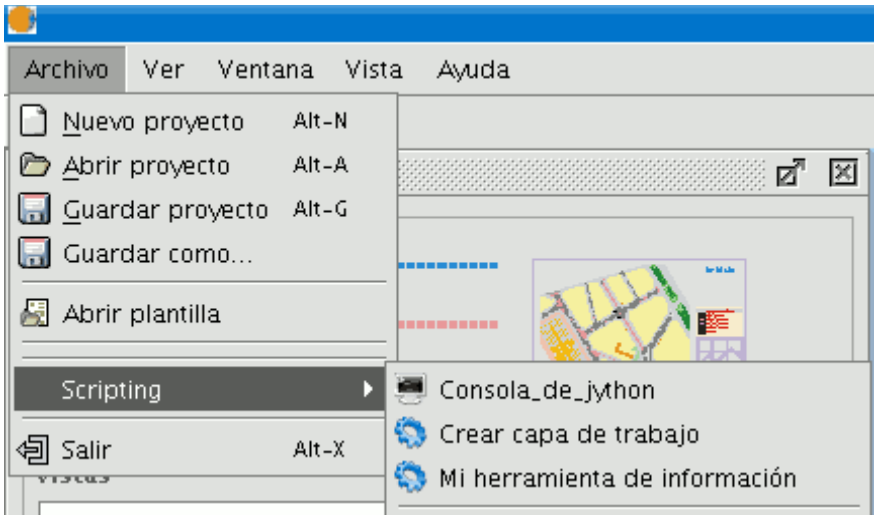
```
        <depends plugin-name="org.gvsig.scripting"/>
               <resourceBundle name="text"/>
        <extensions>
               <extension class-name="org.gvsig.scripting.ScriptingExtension"
                      description="Extension de soporte para Scripts de usuario."
                      active="true">
                      <menu text="Archivo/Scripting/Mi herramienta de información"
                            tooltip="Mi herramienta de información"
                            action-
command="run(fileName='gvSIG/extensiones/miHerramientaDeInformacion/miHerramientaDeInformaci
on.py',language='jython')"
            icon="images/default.png"
                             position="55"
                             />
      <menu text="Archivo/Scripting/Crear capa de trabajo"
            tooltip="Crea una capa para utilizarla con la utilidad 'Mi herramienta de
informacion'"
            action-
command="run(fileName='gvSIG/extensiones/miHerramientaDeInformacion/anyadirMiCapaDeTrabajo.p
y',language='jython')"
            icon="images/default.png"
            position="55"
        />
      <tool-bar name="Scripting">
        <action-tool icon="images/default.png"
                     action-
command="run(fileName='gvSIG/extensiones/miHerramientaDeInformacion/anyadirMiCapaDeTrabajo.p
y',language='jython')" tooltip="Crea una capa para utilizarla con la utilidad 'Mi
herramienta de informacion'"
                     />
        <action-tool icon="images/default.png"
                     action-
command="run(fileName='gvSIG/extensiones/miHerramientaDeInformacion/miHerramientaDeInformaci
on.py',language='jython')" tooltip="Mi herramienta de información"
                     />
      </tool-bar>
               </extension>
       </extensions>
</plugin-config>
```

- En este punto podemos arrancar la aplicación y ver que se han añadido los botones en la barra de herramientas y las entradas en la barra de menús.









- Dentro de la carpeta “miHerramientaDeInformacion” generaremos un fichero .”csv “ con los datos que se van a presentar con la herramienta de información. El fichero se llamará “municipiosAdemuz.csv” y tendrá el siguiente contenido:

```
Nombre;Codigo;Comarca;Provincia;X;Y
CASAS BAJAS;4609088;El Rincon de Ademuz;Valencia;648522.72;4431068.44
CASAS ALTAS;4609087;El Rincon de Ademuz;Valencia;649319.84;4433082.21
VALLANCA;4609252;El Rincon de Ademuz;Valencia;640425.70;4435263.80
PUEBLA DE SAN MIGUEL;4609201;El Rincon de Ademuz;Valencia;659430.60;4435809.18
CASTIELFABIB;4609092;El Rincon de Ademuz;Valencia;641977.97;4443528.63
ADEMUZ;4609001;El Rincon de Ademuz;Valencia;651081.88;4437193.65
TORREBAJA;4609242;El Rincon de Ademuz;Valencia;648648.53;4440549.03
```

- Podemos crear en este punto el script encargado de crear la capa de trabajo que

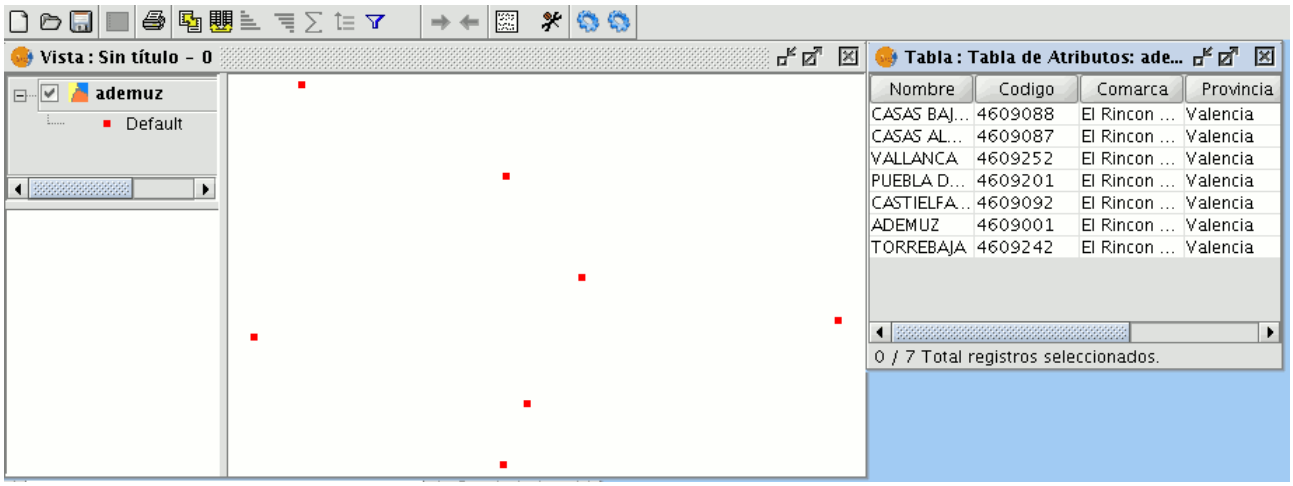
utilizaremos en nuestro ejemplo, el fichero que contendrá este script se llamará “anyadirMiCapaDeTrabajo.py”. El contenido de este script es el siguiente:

```
"""
Script que genera una capa de puntos a partir de un fichero csv
(municipiosAdemuz.csv) que se utiliza para trabajar con la herramienta
miHerramientaDeInformacion.
"""

import os.path

from gvsiglib import *


def getMapContext():
  """
  Comprueba que el documanto activo es una vista y devuelve
  el mapContext asociado a ella
  """
  vista = gvSIG.getActiveDocument()
  if vista == None:
    print "No se puede acceder al documento activo."
    return None
  try:
    mapContext = vista.getModel().getMapContext()

  except Exception, e:
    print "El documento activo no parece ser una vista."
    print "Error %s %s" % (str(e.__class__),str(e))
    return None

  return mapContext

def estaMiCapa(layers):
  """
  Funcion encargada de comprobar si la capa de trabajo
  se encuentra en la coloeccion de capas indicada.
  """
  for n in range(layers.getLayersCount()):
    layer = layers.getLayer(n)
    if isinstance(layer,LayerCollection):
      if estaMiCapa(layer):
        return True
    if layer.getProperty("capaConMiInformacionEspecialDeAdemuz")==1:
      return True
  return False

def crearMiCapaDeTrabajo():
  """
  Funcion encargada de crear y cargar en la vista la capa
  de trabajo a partir del fichero csv
  """
```

```
  mapContext = getMapContext()
  if mapContext==None:
    return

  #comprobamos si ya esta cargada la capa de trabajo en la vista
  layers=mapContext.getLayers()
  if estaMiCapa(layers):
    return

  # Lo primero a hacer sera crear un dataSource basado en el fichero
  # csv
  dataSourceFactory=LayerFactory.getDataSourceFactory()

  fileName = os.path.join(
      gvSIG.getScriptsDirectory(),
      "..",
      "..",
      "miHerramientaDeInformacion",
      "municipiosAdemuz.csv"
      )
  dataSourceFactory.addFileDataSource("csv string", "ademuz", fileName)
  ds = dataSourceFactory.createRandomDataSource("ademuz")
  ds.start()

  # Crearemos el driver que gestiona la capa de eventos y lo enlazaremos con
  # la fuente de datos que acabamos de crear indicandole que columnas de esta
  # representan los puntos de la geomretria
  xFieldIndex = ds.getFieldIndexByName("X")
  yFieldIndex = ds.getFieldIndexByName("Y")
  AddEventThemeDriver=gvSIG.classForName("com.iver.gvsig.addeventtheme.AddEventThemeDriver")
  addEventThemeDriver = AddEventThemeDriver()
  addEventThemeDriver.setData(ds, xFieldIndex, yFieldIndex)

  # Crearemos ahora la nueva capa basada en este driver
  capa = None
  try:
    capa=gvSIG.getExtensionPoints().get("Layers").create("com.iver.cit.gvsig.fmap.layers.FLa
yerGenericVectorial")
    capa.setName("ademuz")
    capa.setDriver(addEventThemeDriver)
    capa.setProjection(mapContext.getProjection())
  except Exception, e:
    print "Se ha producido un error creando la capa. Error %s %s" %
(str(e.__class__),str(e))
    return

  # Una vez creada la capa se le añade una propiedad para reconocerla
  # como nuestra capa de trabajo
  capa.setProperty("capaConMiInformacionEspecialDeAdemuz",1)

  # La añadiremos a la lista de capas del mapContext de la vista
  mapContext.getLayers().addLayer(capa)
```

```
  # Indicamos al mapCOntext que se debe repintar
  mapContext.invalidate()

crearMiCapaDeTrabajo()
```

- Una vez creado el script podemos comprobar que si se ejecuta la herramienta se carga una capa de puntos, y los datos de la tabla asociada a la capa coinciden con los datos del fichero csv que hemos creado anteriormente. **Comprobar que si se ejecuta la herramienta se añade la capa adecuada**



- Para continuar con nuestro ejemplo crearemos el script que gestionará la herramienta de información. Llamaremos a este script “miHerramientaDeInformacion.py” y tendrá el siguiente contenido:

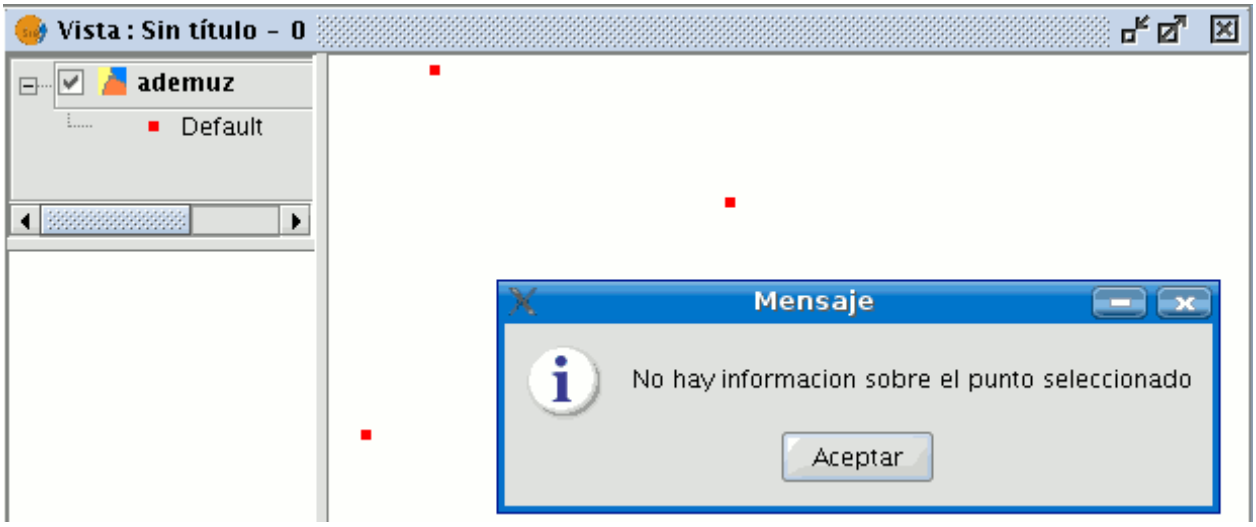
```
from java.util import HashMap
from java.awt import Cursor, Point
from java.awt.event import MouseEvent

from gvsiglib import *

panel = None # Almacena una instancia de la ventana de informacion mostrada

class MyContentHandler(SaxContentHandler):
  """
  Parsea el xml asociado a la informacion de un punto
  transformandolo en un diccionario clave-valor

  FIXME: Solo funciona con capas vectoriales. Habria que arreglarlo
```

```
  para otro tipo de capas
  """
  def __init__(self, valores):
    self.valores = valores

  def startElement(self, nameSpace, localName, qName, attrs):
    valor = {} # Crea un diccionario vacio
    for i in range(attrs.getLength()):
      name=attrs.getQName(i)
      if name in ("",None):
        name=attrs.getLocalName(i)
      valor[name] = attrs.getValue(i)
    if len(valor) >0:
      self.valores.append(valor)

  def endElement(self, nameSpace, localName, qName):
    pass

  def characters(self, value, start, length):
    pass

class MiHerramientaDeInformacionListener(PointListener):
    """
    Esta clase recibe los eventos de clicado sobre el mapControl
    """
    def __init__(self,vista,mapControl):
        self._cursor = Cursor.getPredefinedCursor(Cursor.HAND_CURSOR)
        self._mapControl=mapControl
        self._vista=vista

    def getCursor(self):
      "@sig public java.awt.Cursor getCursor()"
      return self._cursor

    def cancelDrawing(self):
      "@sig public boolean cancelDrawing()"
      return False;

    def pointDoubleClick(self, event):
      "@sig public void pointDoubleClick(PointEvent event) throws BehaviorException"
      pass

    def point(self,event):
      "@sig public void point(PointEvent event) throws BehaviorException"
      global panel
      # Este evento es invocado cada vez que se produce un clic sobre el mapControl
      # estando nuestra herramienta activa

      # Lo primero que haremos sera obtener la lista de capas activas en el TOC
      capasSeleccionadas = self._mapControl.getMapContext().getLayers().getActives()

      # Si no hay ninguna capa seleccionada en el TOC no hacemos nada
```

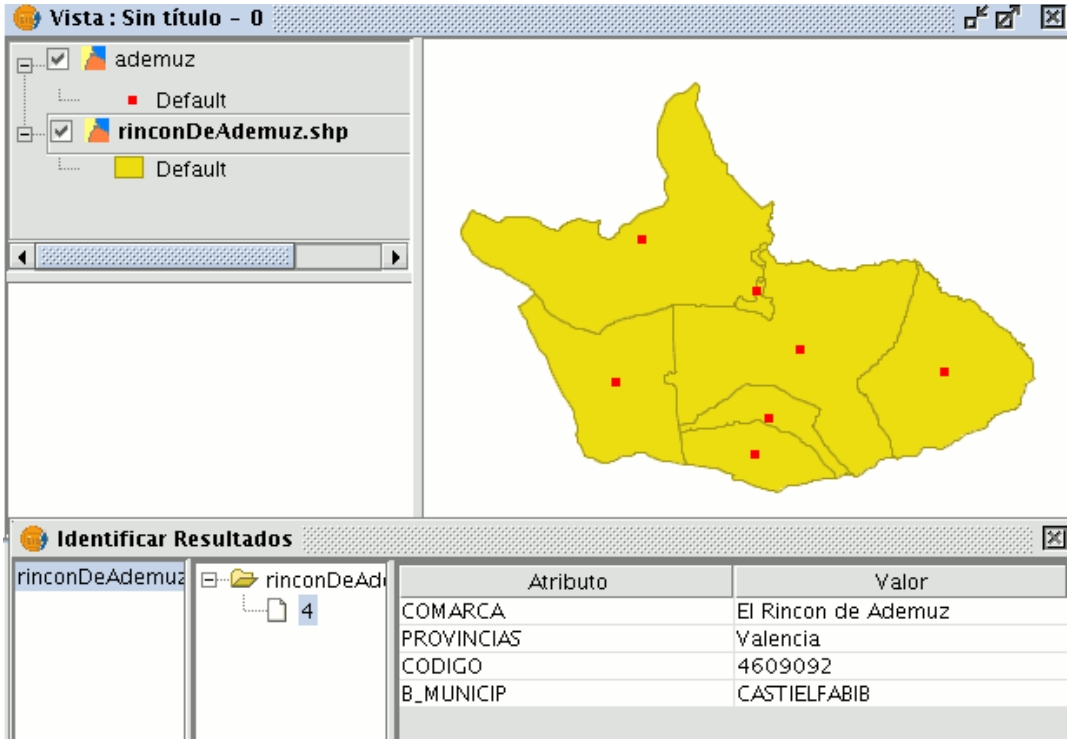
```
      if len (capasSeleccionadas)<1:
        return
      # Si hay mas de una capa activa presentaremos la ventana de informacion predeterminada
      if len(capasSeleccionadas) >1:
        showInfo(self._vista, event.getPoint())
        return
      # Si no esta activa la capa de trabajo presentaremos la ventana de informacion
predeterminada
      if capasSeleccionadas[0].getProperty("capaConMiInformacionEspecialDeAdemuz")!=1:
        showInfo(self._vista, event.getPoint())
        return

      # Si hemos llegado hasta aqui es que solo estaba activa en el TOC nuestra capa de
trabajo
      # procederemos a recuperar la informacion asociada al pùnto que se ha clicado
      tolerancia = self._mapControl.getViewPort().toMapDistance(10) # Transforma pixels a
unidades de mapa
      punto = Point(int(event.getPoint().getX()),int(event.getPoint().getY()))
      valores = [] # Crea una lista vacia

      capa = capasSeleccionadas[0]
      info = capa.getInfo(punto, tolerancia, None)
      for atributo in info:
        atributo.parse(MyContentHandler(valores))

      if len(valores)<1:
        showMessageDialog("No hay informacion sobre el punto seleccionado")
        return

      # Una vez hemos recogido los atributos los presentamos usando el panel de informacion
definido para ello
      parametros= HashMap()
      parametros.put("valores", valores)

      if panel != None:
        panel.close()

      panel=gvSIG.show("gvSIG/extensiones/miHerramientaDeInformacion/miPanelDeInformacion.xm
l","jython",325,150,parametros)

def showInfo(vista,point):
  """
  Muestra la ventana de informacion predeterminada para el punto indicado de la vista
  """
  mapControl = vista.getMapControl()
  infoListener = InfoListener(mapControl)
  event =
MouseEvent(vista,MouseEvent.BUTTON1,MouseEvent.ACTION_EVENT_MASK,MouseEvent.MOUSE_CLICKED,50
0,400,1,True)
  pointEvent = PointEvent(point,event)
  infoListener.point(pointEvent)
```

```
def main():
    vista = gvSIG.getActiveDocument()
    if vista == None:
      print "No se puede acceder al documento activo."
      return None
    try:
      mapContext = vista.getModel().getMapContext()
      mapControl = vista.getMapControl()

    except Exception, e:
      print "El documento activo no parece ser una vista."
      print "Error %s %s" % (str(e.__class__),str(e))
      return None
    if mapContext == None:
      return

    # Si no hemos registrado en el mapControl nuestra herramienta de informacion
    # Creamos nuestro Listener y lo registramos en el mapControl
    if not mapControl.hasTool("MiHerramientaDeInformacion"):
      il=MiHerramientaDeInformacionListener(vista,mapControl)
      mapControl.addMapTool("MiHerramientaDeInformacion", PointBehavior(il))

    # Indicamos al mapControl que esta activa nuestra herramienta de informacion
    mapControl.setTool("MiHerramientaDeInformacion")

main()
```

- Si arrancamos la aplicación y ejecutamos la herramienta para crear la capa de trabajo y a continuación, una vez que se haya cargado la capa, ejecutamos la herramienta de pedir información de un punto podremos comprobar que si no se pulsa en los puntos o en el área de tolerancia de ese punto, se muestra una ventana indicando que no hay información asociada al punto seleccionado.



- Para realizar este ejemplo hemos utilizado una capa de polígonos con las localidades que integran el termino municipal del Rincón de Ademuz. Si se activa la utilidad “miHerramientaDeInformacion” con otra capa activa que no sea la capa que hemos creado nosotros anteriormente, y se solicita información sobre ésta aparecerá la ventana estandard de información de gvSIG.



- A continuación, en la misma carpeta donde hemos creado los archivos anteriores creamos otro archivo nuevo que llamaremos “miPanelDeInformacion.xml”. Este archivo es el que definirá la ventana que utilizaremos para mostrar la información de nuestra capa. El contenido de este archivo es:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- generated by ThinG, the Thinlet GUI editor -->
<panel columns="1" gap="3" >
  <script language="jython" method="init" src="miPanelDeInformacion.py"/>
  <panel halign="right" >
        <label name="lblContadorDeFichas" text="Ficha 1 de X"/>
    </panel>
    <panel columns="2" gap="3" height="100" valign="top" width="300">
        <label name="lblCodigo" text="Codigo"/>
        <textfield halign="left" height="20" name="txtCodigo" width="100"/>
        <label name="lblNombre" text="Nombre"/>
        <textfield height="20" name="txtNombre" width="200"/>
        <label name="lblProvincia" text="Provincia"/>
        <textfield name="txtProvincia"/>
        <label name="lblComarca" text="Comarca"/>
        <textfield name="txtComarca"/>
    </panel>
    <panel gap="3" halign="right" valign="center">
        <button action="clickAnterior(thinlet)" name="btnAnterior" text="Anterior"/>
```

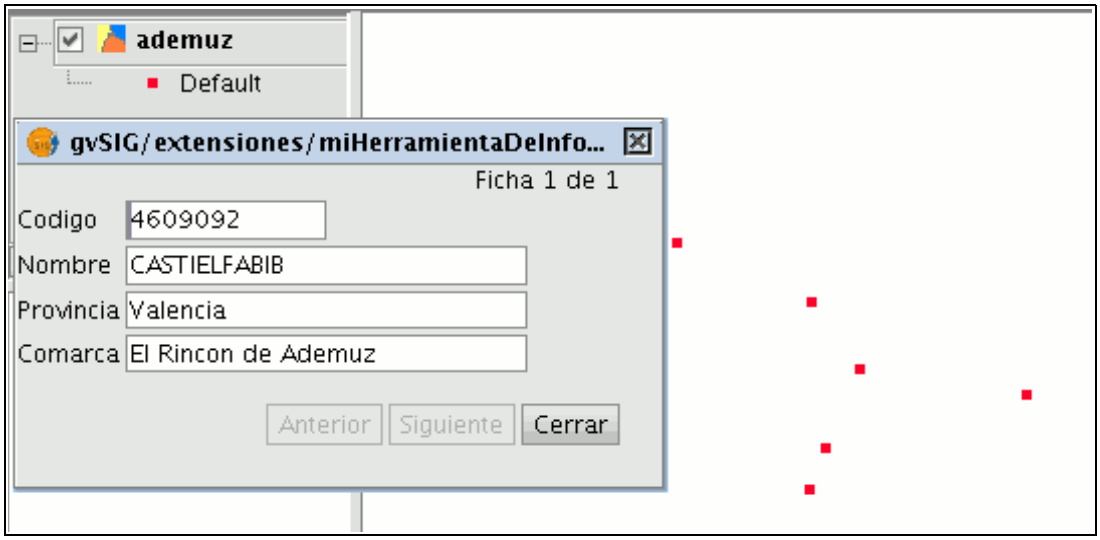
```
        <button action="clickSiguiente(thinlet)" name="btnSiguiente" text="Siguiente"/>
        <button action="thinlet.closeWindow()" name="btnCerrar" text="Cerrar"/>
    </panel>
</panel>
```

- Creamos a continuación un fichero llamado “miPanelDeInformacion.py”, que será el encargado de gestionar como se visualizan los datos en la ventana que hemos creado anteriormente. El contenido de este fichero será:

```
"""
Modulo que se encarga de gestionar la visualizacion de los datos en la
ventana miPaneldeInformacion.xml
"""
from gvsiglib import *

current=0
valores=None

def clickSiguiente(thinlet):
  rellenarFicha(current+1)

def clickAnterior(thinlet):
  rellenarFicha(current-1)

def setValores(thinlet, misValores):
  global valores

  valores = misValores
  rellenarFicha(0)

def rellenarFicha(indice):
  """
  Carga los datos en los controles asignados y habilita los botones de
  siguiente y anterior segun proceda
  """
  global current

  if indice <0:
    return
  if indice >= len(valores):
    return

  current = indice

  valor=valores[current]

  thinlet.setString(lblContadorDeFichas,"text","Ficha %s de %s" %(current +1 ,len(valores)))
  thinlet.setString(txtCodigo,"text",valor.get("Codigo",""))
  thinlet.setString(txtNombre,"text",valor.get("Nombre",""))
  thinlet.setString(txtProvincia,"text",valor.get("Provincia",""))
```

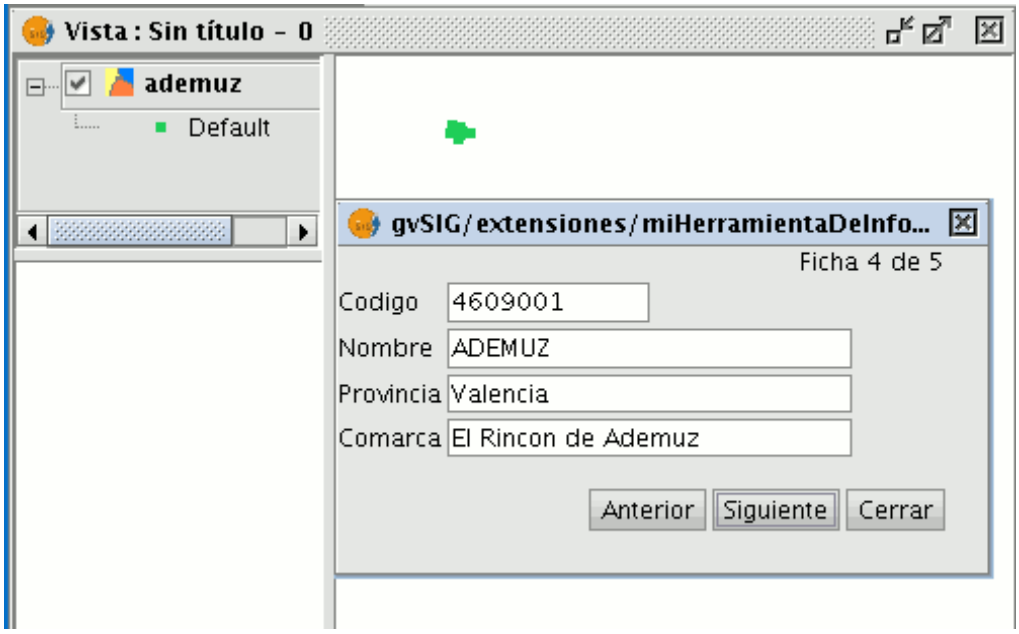
```
    thinlet.setString(txtComarca,"text",valor.get("Comarca",""))
    if current <1:
      thinlet.setBoolean(btnAnterior,"enabled",False)
    else:
      thinlet.setBoolean(btnAnterior,"enabled",True)

    if current >=len(valores)-1:
      thinlet.setBoolean(btnSiguiente,"enabled",False)
    else:
      thinlet.setBoolean(btnSiguiente,"enabled",True)

# Recoge los parametros pasados al thinlet en la llamada a la funcion
# gvSIG.show que recibimos en la variable global params para inicializar
# los datos del gui
setValores(thinlet,params.get("valores"))
```

- En este momento podemos arrancar gvSIG y utilizar nuestras herramientas. Para comprobar el funcionamiento, active miHerramientaDeInformacion con la capa de puntos que hemos creado seleccionada, a continuación pulse sobre uno de los puntos y se abrirá la ventana de información que hemos creado.



- Si reducimos el zoom con el que visualizamos la capa, de manera que los puntos queden superpuestos unos con otros y solicitamos información con nuestra herramienta se cargará en la ventana la información de los puntos que estén situados dentro del área de tolerancia que hayamos definido. Para poder visualizar la información cuando existen múltiples entidades se habilitan los botones anterior

y siguiente.


Vista : Sin título - 0
ademuz
Default
gvSIG/extensiones/miHerramientaDeInfo...
Ficha 4 de 5
Codigo
4609001
Nombre
ADEMUZ
Provincia
Valencia
Comarca
El Rincon de Ademuz
Anterior
Siguiente
Cerrar

3 Anexos

3.1 Centrar vista sobre un punto

3.1.1 config.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<plugin-config>
        <libraries library-dir="../org.gvsig.scripting"/>
        <depends plugin-name="org.gvsig.scripting"/>
                <resourceBundle name="text"/>
        <extensions>
                <extension class-name="org.gvsig.scripting.ScriptingExtension"
                        description="Extension de soporte para Scripts de usuario."
                        active="true">
                        <menu text="Archivo/Scripting/Centrar vista en un punto"
                          tooltip="Centrar la vista en un punto"
                                action-command =
"show(fileName='gvSIG/extensiones/centrarVistaSobreUnPunto/centrarVistaSobreUnPunto.xml',language='j
ython',title='Centrar la vista a un punto',width=210,height=86)"
                                 icon="images/default.png"
                                 position="55"
                                 />
                        <menu text="Archivo/Scripting/Borrar puntos"
                                tooltip="Borrar puntos"
                                action-command =
"run(fileName='gvSIG/extensiones/centrarVistaSobreUnPunto/limpiarElGraphics.py',language='jython')"
                                icon="images/default.png"
                                position="56"
                                 />
                </extension>
        </extensions>
</plugin-config>
```

3.1.2 centrarVistaSobreUnPunto.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- generated by ThinG, the Thinlet GUI editor -->
<panel columns="3" gap="3">
    <script language="jython" method="init" src="centrarVistaSobreUnPunto.py"/>
    <label colspan="3" text="Coordenadas para centrar la vista"/>
    <label colspan="2" halign="right" text="Coordenada x:"/>
    <textfield name="txtX"/>
    <label colspan="2" halign="right" text="Coordenada y:"/>
    <textfield name="txtY"/>
    <panel colspan="3" gap="2" halign="right">
        <button halign="right" name="botAplicar" text="Aplicar" action="clickAplicar(thinlet)"/>
        <button halign="right" name="botCerrar" text="Cerrar"     action="thinlet.closeWindow()"/>

    </panel>
</panel>
```

3.1.3 centrarVistaSobreUnPunto.py

```
import java.awt.geom.Point2D as Point2D
import java.awt.geom.Rectangle2D as Rectangle2D

import sys

from gvsiglib import *

mapContext = None

def getMapContext():
    view = gvSIG.getActiveDocument()
    if view == None:
      print "No se puede acceder al documento activo."
      return None
    try:
      mapContext = view.getModel().getMapContext()

    except Exception, e:
      print "El documento activo no parece ser una vista."
      print "Error %s %s" % (str(e.__class__),str(e))
      return None

    return mapContext

mapContext = getMapContext()

def clickAplicar(thinlet):

    global mapContext

    if mapContext == None:
      print "No se puede acceder al documento activo."
      return

    if mapContext.getLayers().getLayersCount() < 1:
      print "El documento activo no tiene capas disponibles."
      return
    x = float(thinlet.getString(txtX, "text"))
    y = float(thinlet.getString(txtY, "text"))
    center = zoomToCoordinates(mapContext, x,y)
    drawPoint(mapContext,center)

def zoomToCoordinates(mapContext, x,y):
  try:
    oldExtent = mapContext.getViewPort().getAdjustedExtent()
    oldCenterX = oldExtent.getCenterX()
    oldCenterY = oldExtent.getCenterY()
    center=Point2D.Double(x,y)
```

```
    movX = x-oldCenterX
    movY = y-oldCenterY
    upperLeftCornerX = oldExtent.getMinX()+movX
    upperLeftCornerY = oldExtent.getMinY()+movY
    width = oldExtent.getWidth()
    height = oldExtent.getHeight()
    extent = Rectangle2D.Double(upperLeftCornerX, upperLeftCornerY, width, height)
    mapContext.getViewPort().setExtent(extent)
    return center
  except ValueError, e:
    print "Se ha producido un error realizando zoom a las coordenadas (%s,%s). Error  %s, %s" % (
      repr(x),
      repr(y),
      str(e.__class__),
      str(e)
    )
    return None

def drawPoint(mapContext, center, color=None):
    """
    Esta función pintará un punto sobre la capa de gráficos
    asociada al mapContext.
    Todo mapContext además de las capas que tenga cargadas dispone
    una capa graphics sobre la que dibujar elementos gráficos.
    """

    if color == None:
        import java.awt.Color as Color
        color = Color.blue

    layer=mapContext.getGraphicsLayer()
    layer.clearAllGraphics()
    theSymbol = FSymbol(FConstant.SYMBOL_TYPE_POINT,color)
    idSymbol = layer.addSymbol(theSymbol)
    geom = ShapeFactory.createPoint2D(center.getX(),center.getY())
    theGraphic = FGraphic(geom, idSymbol)
    layer.addGraphic(theGraphic)

def elDocumentoActivoEsUnaVistaValida():
    global mapContext

    if mapContext == None:
      print "El documento activo nop parece ser una vista"
      return False

    if mapContext.getLayers().getLayersCount() < 1:
      print "El documento activo no tiene capas disponibles."
      return False
    return True

if activeDocumentIsAValidView():
  thinlet.setBoolean(botAplicar,"enabled",True)
```

```
else:
  thinlet.setBoolean(botAplicar,"enabled",False)
```

3.1.4 limpiarElGraphics.py

```
from gvsiglib import *

def main():
    view = gvSIG.getActiveDocument()
    if view == None:
      print "No se puede acceder al documento activo."
      return None
    try:
      mapContext = view.getModel().getMapContext()
      mapControl = view.getMapControl()

    except Exception, e:
      print "El documento activo no parece ser una vista."
      print "Error %s %s" % (str(e.__class__),str(e))
      return None
    if mapContext == None:
      return
    layer=mapContext.getGraphicsLayer()
    layer.clearAllGraphics()
    mapContext.invalidate()

main()
```

3.2 Mi herramienta de información

3.2.1 config.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<plugin-config>
        <libraries library-dir="../org.gvsig.scripting"/>
        <depends plugin-name="org.gvsig.scripting"/>
                <resourceBundle name="text"/>
        <extensions>
                <extension class-name="org.gvsig.scripting.ScriptingExtension"
                        description="Extension de soporte para Scripts de usuario."
                        active="true">
                        <menu text="Archivo/Scripting/Mi herramienta de información"
                              tooltip="Mi herramienta de información"
                              action-
command="run(fileName='gvSIG/extensiones/miHerramientaDeInformacion/miHerramientaDeInformacion.py',l
anguage='jython')"
             icon="images/default.png"
                               position="55"
                               />
      <menu text="Archivo/Scripting/Crear capa de trabajo"
             tooltip="Crea una capa para utilizarla con la utilidad 'Mi herramienta de informacion'"
             action-
command="run(fileName='gvSIG/extensiones/miHerramientaDeInformacion/anyadirMiCapaDeTrabajo.py',langu
age='jython')"
             icon="images/default.png"
             position="55"
        />
      <tool-bar name="Scripting">
        <action-tool icon="images/default.png"
                     action-
command="run(fileName='gvSIG/extensiones/miHerramientaDeInformacion/anyadirMiCapaDeTrabajo.py',langu
age='jython')" tooltip="Crea una capa para utilizarla con la utilidad 'Mi herramienta de
informacion'"
                     />
        <action-tool icon="images/default.png"
                     action-
command="run(fileName='gvSIG/extensiones/miHerramientaDeInformacion/miHerramientaDeInformacion.py',l
anguage='jython')" tooltip="Mi herramienta de información"
                     />
      </tool-bar>
                </extension>
        </extensions>
</plugin-config>
```

3.2.2 municipiosAdemuz.csv

```
Nombre;Codigo;Comarca;Provincia;X;Y
CASAS BAJAS;4609088;El Rincon de Ademuz;Valencia;648522.72;4431068.44
CASAS ALTAS;4609087;El Rincon de Ademuz;Valencia;649319.84;4433082.21
VALLANCA;4609252;El Rincon de Ademuz;Valencia;640425.70;4435263.80
PUEBLA DE SAN MIGUEL;4609201;El Rincon de Ademuz;Valencia;659430.60;4435809.18
CASTIELFABIB;4609092;El Rincon de Ademuz;Valencia;641977.97;4443528.63
ADEMUZ;4609001;El Rincon de Ademuz;Valencia;651081.88;4437193.65
TORREBAJA;4609242;El Rincon de Ademuz;Valencia;648648.53;4440549.03
```

3.2.3 miHerramientaDeInformacion.py

```
from java.util import HashMap
from java.awt import Cursor, Point
from java.awt.event import MouseEvent

from gvsiglib import *

panel = None # Almacena una instancia de la ventana de informacion mostrada

class MyContentHandler(SaxContentHandler):
  """
  Parsea el xml asociado a la informacion de un punto
  transformandolo en un diccionario clave-valor

  FIXME: Solo funciona con capas vectoriales. Habria que arreglarlo
  para otro tipo de capas
  """
  def __init__(self, values):
    self.values = values

  def startElement(self, nameSpace, localName, qName, attrs):
    value = {} # Crea un diccionario vacio
    for i in range(attrs.getLength()):
      name=attrs.getQName(i)
      if name in ("",None):
        name=attrs.getLocalName(i)
      value[name] = attrs.getValue(i)
    if len(value) >0:
      self.values.append(value)

  def endElement(self, nameSpace, localName, qName):
    pass

  def characters(self, value, start, length):
    pass

class MyInformationToolListener(PointListener):
    """
    Esta clase recibe los eventos de click sobre el mapControl
    """
    def __init__(self,view,mapControl):
        self._cursor = Cursor.getPredefinedCursor(Cursor.HAND_CURSOR)
        self._mapControl=mapControl
        self._view=view

    def getCursor(self):
      "@sig public java.awt.Cursor getCursor()"
      return self._cursor

    def cancelDrawing(self):
```

```
      "@sig public boolean cancelDrawing()"
      return False;

    def pointDoubleClick(self, event):
      "@sig public void pointDoubleClick(PointEvent event) throws BehaviorException"
      pass

    def point(self,event):
      "@sig public void point(PointEvent event) throws BehaviorException"
      global panel
      # Este evento es invocado cada vez que se produce un clic sobre el mapControl
      # estando nuestra herramienta activa

      # Lo primero que haremos sera obtener la lista de capas activas en el TOC
      selectedLayers = self._mapControl.getMapContext().getLayers().getActives()

      # Si no hay ninguna capa seleccionada en el TOC no hacemos nada
      if len (selectedLayers)<1:
        return
      # Si hay mas de una capa activa presentaremos la ventana de informacion predeterminada
      if len(selectedLayers) >1:
        showInfo(self._view, event.getPoint())
        return
      # Si no esta activa la capa de trabajo presentaremos la ventana de informacion predeterminada
      if selectedLayers[0].getProperty("capaConMiInformacionEspecialDeAdemuz")!=1:
        showInfo(self._view, event.getPoint())
        return

      # Si hemos llegado hasta aqui es que solo estaba activa en el TOC nuestra capa de trabajo
      # procederemos a recuperar la informacion asociada al pùnto que se ha clicado
      tolerance = self._mapControl.getViewPort().toMapDistance(10) # Transforma pixels a unidades de
mapa
      thePoint = Point(int(event.getPoint().getX()),int(event.getPoint().getY()))
      values = [] # Crea una lista vacia

      layer = selectedLayers[0]
      info = layer.getInfo(thePoint, tolerance, None)
      for atribute in info:
        atribute.parse(MyContentHandler(values))

      if len(values)<1:
        showMessageDialog("No hay informacion sobre el punto seleccionado")
        return

      # Una vez hemos recogido los atributos los presentamos usando el panel de informacion definido
para ello
      params= HashMap()
      params.put("values", values)

      if panel != None:
        panel.close()
```

```
      panel=gvSIG.show("gvSIG/extensiones/miHerramientaDeInformacion/miPanelDeInformacion.xml","jyth
on",325,150,params)

def showInfo(view,point):
  """
  Muestra la ventana de informacion predeterminada para el punto indicado de la vista
  """
  mapControl = view.getMapControl()
  infoListener = InfoListener(mapControl)
  event =
MouseEvent(view,MouseEvent.BUTTON1,MouseEvent.ACTION_EVENT_MASK,MouseEvent.MOUSE_CLICKED,500,400,1,T
rue)
  pointEvent = PointEvent(point,event)
  infoListener.point(pointEvent)


def main():
    view = gvSIG.getActiveDocument()
    if view == None:
      print "No se puede acceder al documento activo."
      return None
    try:
      mapContext = view.getModel().getMapContext()
      mapControl = view.getMapControl()

    except Exception, e:
      print "El documento activo no parece ser una vista."
      print "Error %s %s" % (str(e.__class__),str(e))
      return None
    if mapContext == None:
      return

    # Si no hemos registrado en el mapControl nuestra herramienta de informacion
    # Creamos nuestro Listener y lo registramos en el mapControl
    if not mapControl.hasTool("MyInformationToolListener"):
      il=MyInformationToolListener(view,mapControl)
      mapControl.addMapTool("MyInformationToolListener", PointBehavior(il))

    # Indicamos al mapControl que esta activa nuestra herramienta de informacion
    mapControl.setTool("MyInformationToolListener")

main()
```

3.2.4 miPanelDeInformacion.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- generated by ThinG, the Thinlet GUI editor -->
<panel columns="1" gap="3" >
  <script language="jython" method="init" src="miPanelDeInformacion.py"/>
  <panel halign="right" >
        <label name="lblContadorDeFichas" text="Ficha 1 de X"/>
    </panel>
    <panel columns="2" gap="3" height="100" valign="top" width="300">
        <label name="lblCodigo" text="Codigo"/>
        <textfield halign="left" height="20" name="txtCodigo" width="100"/>
        <label name="lblNombre" text="Nombre"/>
        <textfield height="20" name="txtNombre" width="200"/>
        <label name="lblProvincia" text="Provincia"/>
        <textfield name="txtProvincia"/>
        <label name="lblComarca" text="Comarca"/>
        <textfield name="txtComarca"/>
    </panel>
    <panel gap="3" halign="right" valign="center">
        <button action="clickAnterior(thinlet)" name="btnAnterior" text="Anterior"/>
        <button action="clickSiguiente(thinlet)" name="btnSiguiente" text="Siguiente"/>
        <button action="thinlet.closeWindow()" name="btnCerrar" text="Cerrar"/>
    </panel>
</panel>
```

3.2.5 miPanelDeInformacion.py

```
"""
Modulo que se encarga de gestionar la visualizacion de los datos en la
ventana miPaneldeInformacion.xml
"""
from gvsiglib import *

current=0
values=None


def clickSiguiente(thinlet):
  fillCard(current+1)

def clickAnterior(thinlet):
  fillCard(current-1)

def setValues(thinlet, myValues):
  global values

  values = myValues
  fillCard(0)

def fillCard(index):
  """
  Carga los datos en los controles asignados y habilita los botones de
  siguiente y anterior segun proceda
  """
  global current

  if index <0:
    return
  if index >= len(values):
    return

  current = index

  value=values[current]

  thinlet.setString(lblContadorDeFichas,"text","Ficha %s de %s" %(current +1 ,len(values)))
  thinlet.setString(txtCodigo,"text",value.get("Codigo",""))
  thinlet.setString(txtNombre,"text",value .get("Nombre",""))
  thinlet.setString(txtProvincia,"text",value .get("Provincia",""))
  thinlet.setString(txtComarca,"text",value .get("Comarca",""))
  if current <1:
    thinlet.setBoolean(btnAnterior,"enabled",False)
  else:
    thinlet.setBoolean(btnAnterior,"enabled",True)

  if current >=len(values)-1:
```

```
    thinlet.setBoolean(btnSiguiente,"enabled",False)
  else:
    thinlet.setBoolean(btnSiguiente,"enabled",True)

# Recoge los parametros pasados al thinlet en la llamada a la funcion
# gvSIG.show que recibimos en la variable global params para inicializar
# los datos del gui
setValues(thinlet,params.get("values"))
```

3.2.6 anyadirMiCapaDeTrabajo.py

```
"""
Script que genera una capa de puntos a partir de un fichero csv
(municipiosAdemuz.csv) que se utiliza para trabajar con la herramienta
miHerramientaDeInformacion.
"""

import os.path

from gvsiglib import *

def getMapContext():
  """
  Comprueba que el documanto activo es una vista y devuelve
  el mapContext asociado a ella
  """
  view = gvSIG.getActiveDocument()
  if view == None:
    print "No se puede acceder al documento activo."
    return None
  try:
    mapContext = view.getModel().getMapContext()
  except Exception, e:
    print "El documento activo no parece ser una vista."
    print "Error %s %s" % (str(e.__class__),str(e))
    return None

  return mapContext

def isThereMyLayer(layers):
  """
  Funcion encargada de comprobar si la capa de trabajo
  se encuentra en la coloeccion de capas indicada.
  """
  for n in range(layers.getLayersCount()):
    layer = layers.getLayer(n)
    if isinstance(layer,LayerCollection):
      if isThereMyLayer(layer):
        return True
    if layer.getProperty("capaConMiInformacionEspecialDeAdemuz")==1:
      return True
  return False

def createMyWorkLayer():
  """
  Funcion encargada de crear y cargar en la vista la capa
  de trabajo a partir del fichero csv
  """
  mapContext = getMapContext()
  if mapContext==None:
```

```
  return

#comprobamos si ya esta cargada la capa de trabajo en la vista
layers=mapContext.getLayers()
if isThereMyLayer(layers):
  return

# Lo primero a hacer sera crear un dataSource basado en el fichero
# csv
dataSourceFactory=LayerFactory.getDataSourceFactory()

fileName = os.path.join(
    gvSIG.getScriptsDirectory(),
    "..",
    "..",
    "miHerramientaDeInformacion",
    "municipiosAdemuz.csv"
    )
dataSourceFactory.addFileDataSource("csv string", "ademuz",fileName)
ds = dataSourceFactory.createRandomDataSource("ademuz")
ds.start()

# Crearemos el driver que gestiona la capa de eventos y lo enlazaremos con
# la fuente de datos que acabamos de crear indicandole que columnas de esta
# representan los puntos de la geomretria
xFieldIndex = ds.getFieldIndexByName("X")
yFieldIndex = ds.getFieldIndexByName("Y")
AddEventThemeDriver=gvSIG.classForName("com.iver.gvsig.addeventtheme.AddEventThemeDriver")
addEventThemeDriver = AddEventThemeDriver()
addEventThemeDriver.setData(ds, xFieldIndex, yFieldIndex)

# Crearemos ahora la nueva capa basada en este driver
capa = None
try:
  myLayer=gvSIG.getExtensionPoints().get("Layers").create("GenericVectorial")
  myLayer.setName("ademuz")
  myLayer.setDriver(addEventThemeDriver)
  myLayer.setProjection(mapContext.getProjection())
except Exception, e:
  print "Se ha producido un error creando la capa. Error %s %s" % (str(e.__class__),str(e))
  return

# Una vez creada la capa se le añade una propiedad para reconocerla
# como nuestra capa de trabajo
myLayer.setProperty("capaConMiInformacionEspecialDeAdemuz",1)

# La añadiremos a la lista de capas del mapContext de la vista
mapContext.getLayers().addLayer(myLayer)

# Indicamos al mapContext que se debe repintar
mapContext.invalidate()
```

```
createMyWorkLayer()
```