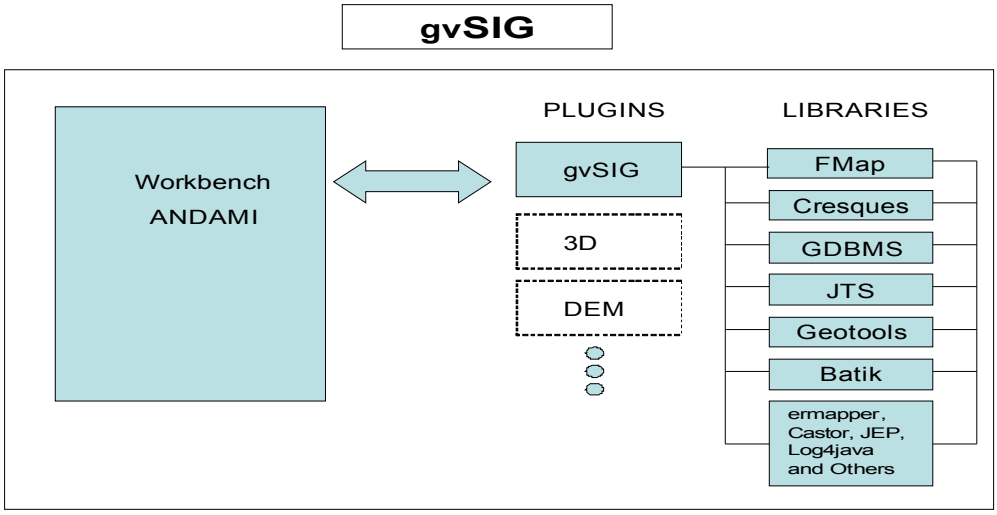
## Scripting on gvSIG

gvSIG is based on a framework that together with the scripting engine lets you add new functionality without extensive knowledge of the application core.

gvSIG is a multi tier application in which different layers of functionality are integrated using **extensions**, in the same way each extension can define its own **extension points**. This model allows developers to add new modules to gvSIG in a very easy way using all the functionality and tools offered by the application core.




- **FrameWork: Andami**. Supports the construction of MDI swing applications, extensible by plugins.
- **Application: gvSIG**. Extension that transforms Andami into a GIS client. It uses the FMap library to do most of the job. The majority of classes here define the User Interface.
- **Libraries:** libFMap, Lib CQ CMS Cresques, libGDBMS, libIverUtiles, libRemoteServices, LibUI,Kxml
- **Andami Extensions (plugins)**: ExtCAD, extWMS, extWFS2, extJDBC, catalogue, gazetteer.

### *Andami extensions.*

Andami extensions are defined by a *.xml* file located at *bin/gvSIG/extensiones*. This configuration file defines:

- The classes that Andami will load when starting up the application
- The menu options and the tools to appear in the toolbar.

### *GUI library for scripting*

**Thinlet** is a library for GUI generation that separates the graphical presentation and the application methods. For the definition of the graphic interface an XML file in XUL format is used.

An additional tool called **ThinG** can be used to design the graphic interface. For more information.

http://thinlet.sourceforge.net/
http://sjobic.club.fr/thinlet/scriptablethinlet/index.html
http://thing.sourceforge.net/

## Examples

gvSIG supports several scripting languages: Jython, Javascript, Beanshell, etc. To practise scripting with gvSIG we will see now some examples using Python 2.1 in its implementation for java, Jython.

### *Exercise 1: Center view on point.*

We are going to create a gvSIG extension which, given a geographical location, centers the view on it and draws a point to mark the location.

1. Create a *bin/gvSIG/extensiones/centerViewOnPoint* directory in the folder where gvSIG is installed.

2. Inside this directory, create a file called *config.xml* with this content.

```
<?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="support extension for user scripts"
active="true">
<menu text="Archivo/Scripting/Center view on point"
tooltip="Center view on point" action-command=""
position="55"/>
</extension>
</extensions>
</plugin-config>
```

Once the file is created, run gvSIG to see that the menu option has been added.

3. In the directory we created before, create a new one called **images** and copy on it a file that we use as icon for the new tool. You can find the icon file at bin/gvSIG/extensiones/org.gvsig.scripting/images/default.png".

4.Add in the config.xml a new tag in the menu section.

```
<menu text="Archivo/Scripting/Center view on point"
tooltip="Center view on point"
action-command=""
icon="images/default.png"
position="55"/>
```

5. Run gvSIG again to make sure the icon has been added to the menu.

6. Create file *centerViewOnPoint.xml* in the same location as *config.xml*, with this content.

```
<?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="centerViewOnPoint.py"/>
<label colspan="3" text="coordinates"/>
<label colspan="2" halign="right" text="coord x:"/>
<textfield name="txtX"/>
<label colspan="2" halign="right" text="coord y:"/>
<textfield name="txtY"/>
<panel colspan="3" gap="2" halign="right">
<button halign="right" name="botAply" text="Apply" />
<button halign="right" name="botClose" text="Close" />
</panel>
</panel>
```

This file defines a simple GUI that will show up when our tool is selected.

7. Now we have to add the instruction "**show"** to the *actioncommand* tag in the config.xml.

**show** parameters:
*filename*: window definition file name.
*language*: scripting language
*title:* window title.
*width*: window width.
*height*: window height.

```
<?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="support extension for user scripts"
active="true">
<menu text="Archivo/Scripting/Center view on point"
tooltip="Center view on point"
action-command =
"show(fileName='gvSIG/extensiones/centerViewOnPoint/centerViewOnPoint.xml',language='jyt
hon',title='Center view on point',width=210,height=86)"
icon="images/default.png"
position="55"/>
</extension>
</extensions>
</plugin-config>
```

8. Run gvSIG to see how the window shows up when selecting the tool.
9. Add an action on the "Close" button. To do this we have to add the action "**thinlet.closeWindow()"** in the file that defines the window (in our case "centerViewOnPoint.xml").

The thinlet object is the extension window and it gives access to the controls.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- generated by ThinG, the Thinlet GUI editor -->
<panel columns="3" gap="3">
<label colspan="3" text="coordinates"/>
<label colspan="2" halign="right" text="coord x:"/>
<textfield name="txtX"/>
<label colspan="2" halign="right" text="coord y:"/>
<textfield name="txtY"/>
<panel colspan="3" gap="2" halign="right">
<button halign="right" name="botAply" text="Apply" action="clickApply(thinlet)"/>
<button halign="right" name="botClose" text="Close" action="thinlet.closeWindow()"/>
</panel>
</panel>
```

After the xml file is modified is not necessary to restart the application, we can just open the tool again to see that the "Close" button closes the window.

10. Now we will add some action on the "Apply" button. To do this, we should create a file "*centerViewOnPoint.py*" in the same location as the centerViewOnPoint.xml, the file should contain the following text:

```
from gvsiglib import *
def clickApply(thinlet):
showMessageDialog("clickApply")
return
```

And the *centerViewOnPoint.xml* file will be modified as follows:

```
<?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="centerViewOnPoint.py"/>
<label colspan="3" text="coordinates"/>
<label colspan="2" halign="right" text="coord x:"/>
<textfield name="txtX"/>
<label colspan="2" halign="right" text="coord y:"/>
<textfield name="txtY"/>
<panel colspan="3" gap="2" halign="right">
<button halign="right" name="botAply" text="Apply" action="clickApply(thinlet)"/>
<button halign="right" name="botClose" text="Close" action="thinlet.closeWindow()"/>
</panel>
</panel>
```

11. If we run the tool again, we will see the new message by clicking "apply".

12. Add code to pick up the coordinates from the window and center the view (see Annex 1):

13. We can also add a condition for the "apply" button: it will be enabled if there is an active view with some layers. To do this, we have to add the following code:

```
def isValidView():

	global mapContext

	if mapContext.getLayers().getLayersCount() < 1:
		print "No layers in active document."
		return False
	return True

if isValidView():
	thinlet.setBoolean(botApply,"enabled",True)
else:
	thinlet.setBoolean(botApply,"enabled",False)
```

14. The next thing we can do is to draw a point at the centered location.

```
def drawPoint(mapContext, center, color=None):
	"""
	This function draws a point on the mapcontext graphic layer.
	"""
	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)
```

15. Once we draw the point, what if we want to delete it? Lets now add an entry in the menu that can clear the graphic layer.

This time we will use the command *run* instead of *show. It uses the following parameters:*

- *fileName*: path relative to gvSIG bin directory, where you can find the executable file
- *language*: scripting language.

In the config.xml file we will add the new menu entry.

```
<menu text="Archivo/Scripting/Clear points"
tooltip="Clear points"
action-command =
"run(fileName='gvSIG/extensiones/centerViewOnPoint/clearGraphics.py',language='jython')"
icon="images/default.png"
position="56"/>
```

Finally we will create a new script called clearGraphics.py located in the same directory as the centerViewOnPoint.py with this code.

from gvsiglib import *

```
def main():
	view = gvSIG.getActiveDocument()
	if view == None:
		print "Active document can not be accessed."
		return None
	try:
		mapContext = view.getModel().getMapContext()
		mapControl = view.getMapControl()

	except Exception, e:
		print "Active document is not a view."
		print "Error %s %s" % (str(e.__class__),str(e))
		return None
	if mapContext == None:
		return
	layer=mapContext.getGraphicsLayer()
	layer.clearAllGraphics()
	mapContext.invalidate()

main()
```

# **ANNEX**:

Code to pick up the coordinates from the window and center the view.

```
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 "Active document can not be accessed."
		return None
	try:
		print "view= %s dir=%s"%(repr(view), dir(view))
		mapContext = view.getModel().getMapContext()

	except Exception, e:
		print "Active document is not a view."
		print "Error %s %s" % (str(e.__class__),str(e))
		return None

	return mapContext

mapContext = getMapContext()

def clickApply(thinlet):

	global mapContext

	if mapContext == None:
		print "Active document can not be accessed."
		return

	if mapContext.getLayers().getLayersCount() < 1:
		print "No layers in active document."
		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 "Error when zooming to coordinates (%s,%s). Error %s, %s" % (
		repr(x),
		repr(y),
		str(e.__class__),
		str(e))
		return None
```