Taller de desarrollo gvSIG 2.0

Documentación complementaria

Índice de contenido

Crear proyecto

Instalación de gvSIG

1. Instalar gvSIG
2. Instalar el plugin org.gvsig.mkmvnproject. Para ello:
    1. Acceder a *Herramientas* > *Instalar un nuevo paquete*
    2. Marcar *Instalación desde archivo*, seleccionando el archivo .gvspks
    3. Marcamos el plugin con ID *org.gvsig.mkmvnproject* y pulsamos *Siguiente*.

Creación del proyecto desde la aplicación gvSIG

1. Opción *Herramientas* > *Development* > *Create plugin*
2. Datos a incluir:
    1. Nombre: LandRegistryViewer (ojo con las mayúsculas)
    2. Group Id: org.gvsig
    3. Create project in: el directorio del nuevo workspace
    4. Choose project type: 1, Basic
    5. Create test application: marcado
    6. Create swing library projects: marcado
    7. Create gvSIG plugin: marcado
3. Eclipse workspace folder to configure? (Aceptar)

Crear espacio de trabajo

1. Abrimos eclipse sobre el nuevo espacio de trabajo creado.
2. Establecemos el encoding a ISO-8859-1
3. En *Java* > *Code Style*, importamos todas las configuraciones definidas por las normas de codificación de gvSIG. Los archivos correspondientes están disponibles en el directorio:

```
$WORKSPACE/org.gvsig.maven.base.build/eclipse-configs
```

4. En *Java* > *Code Style* > *Code Templates* marcar *Automatically add comments for new methods and types*
5. En *Java* > *Editor* > *Save Actions* activar *Format edited lines* y *Organize imports* al guardar.

Cargar e importar los proyectos.

1. Importamos los proyectos de *org.gvsig.landregistryviewer*
2. Importamos los proyectos de *org.gvsig.landregistryviewer.app.extension*
3. Creamos nuevo proyecto *org.gvsig.landregistryviewer*
4. Creamos nuevo proyecto *org.gvsig.landregistryviewer.app*
5. Añadimos las external-tools que necesitamos desde *external tools/Organize favorites/Add* (mvn install, clean, eclipse-eclipse, eclipse-clean).
6. Hacemos un *mvn install*
7. Añadimos desde favoritos el launcher de *gvSIG*.
8. Ejecutamos el launcher de gvSIG.
9. gvSIG arranca y observamos que tenemos la opción *XXXX*.
10. Cerramos gvSIG.

Personalizar gvSIG

Añadimos el tema de andami

Añadiremos:

- Carpeta theme
- splash-taller.png
- andami-theme.xml

1. Crearemos la carpeta *theme* en *src/main/resources* del proyecto *org.gvsig.landregistryviewer.app.extension*.
2. Copiaremos allí el fichero */home/data/theme/splash-taller.png*
3. Crearemos un fichero *andami-theme.xml* y le dejaremos el contenido:

```
<AndamiProperties>
  <ApplicationImages>
    <SplashImages>
      <Splash
        path="splash-taller.png"
        timer="10000"
        x="270" y="240"
        fontsize="18"
        color="80,170,240"
        version="2.0"/>
    </SplashImages>
    <!--BackgroundImage path="theme/logo_es.png"/-->
    <!--WallpaperType value="CENTERED"/-->
    <Icon path="$GVSIG_INSTALL/theme/icon.png"/>
  </ApplicationImages>
  <ApplicationName value="gvSIG 2.0.0"/>
</AndamiProperties>
```

4. Ejecutaremos un *mvn install* para desplegar la extensión sobre gvSIG.
5. Probaremos a arrancar de nuevo la aplicación desde el launcher de gvSIG.

Cambio en la visibilidad de herramientas

Modificaremos :

- LandRegistryViewerExtension

1. Sobre la aplicación arrancada creamos una vista, cargamos la capa, la seleccionamos y comprobamos que están las herramientas de edición en el menú capa.
2. Editamos *LandRegistryViewerExtension.java* y sustituimos el método *initialize* por:

```
private Set<String> extensionsToHidde = new HashSet<String>();

  public void initialize() {
    this.initializeVisibilityControl();
  }

  /**
    * Initialize the set of extension's names and set this
    * class to control the visibility of all extensions.
    */
  private void initializeVisibilityControl() {

    // Prepare a set with the names of the extensions to hide.
    String[] extensions = new String[] {
        "org.gvsig.editing.CreateNewLayer",
        "org.gvsig.editing.StartEditing",
```

```
            "org.gvsig.editing.StopEditing",
            "org.gvsig.editing.ExportTo",
            "org.gvsig.editing.UndoViewExtension",
            "org.gvsig.editing.RedoViewExtension",
            "org.gvsig.editing.ViewCommandStackExtension",
            "org.gvsig.editing.TableCommandStackExtension",
            "org.gvsig.editing.CADExtension",
            "org.gvsig.editing.ExploitExtension",
            "org.gvsig.editing.MoveGeometryExtension",
            "org.gvsig.editing.InsertPointExtension",
            "org.gvsig.editing.InsertMultiPointExtension",
            "org.gvsig.editing.InsertLineExtension",
            "org.gvsig.editing.InsertPolyLineExtension",
            "org.gvsig.editing.InsertPolygonExtension",
            "org.gvsig.editing.JoinExtension",
            "org.gvsig.editing.InternalPolygonExtension",
            "org.gvsig.editing.StretchExtension",
            "org.gvsig.editing.ComplexSelectionGeometryExtension",
            "org.gvsig.editing.SelectionGeometryExtension",
            "org.gvsig.editing.MatrixExtension",
            "org.gvsig.editing.AutoCompletePolygonExtension",
            "org.gvsig.editing.SplitGeometryCADToolExtension"
        };
        for( int i=0; i<extensions.length ; i++ ) {
          this.extensionsToHidde.add(extensions[i]);
        }

        // Set me to manage the visibility of all extensions
        PluginServices.setExclusiveUIExtension(this);
    }

    /**
     * Check if an extension is enabled.
     */
    public boolean isEnabled(IExtension extension) {
      // Is the extension's name is in the set of extension to hide
      // always return false to disable the extension.
      if( this.extensionsToHidde.contains(extension.getClass().getName()) ) {
        return false;
      }
      // Relies in the extension to verify if is enabled
      return extension.isEnabled();
    }

    /**
     * Check if an extension is visible.
     */
    public boolean isVisible(IExtension extension) {
      // Is the extension's name is in the set of extension to hide
      // always return false to hide the extension.
      if( this.extensionsToHidde.contains(extension.getClass().getName()) ) {
        return false;
      }
      // Relies in the extension to verify if is visible
      return extension.isVisible();
    }
```

3. Y hacemos que la clase implemente el interface *ExclusiveUIExtension*.
4. Ejecutaremos un *mvn install* para desplegar los cambios sobre gvSIG.
5. Probaremos a arrancar de nuevo la aplicación desde el launcher de gvSIG.
6. Creamos una vista, cargamos la capa, la seleccionamos y comprobamos que ya no están las herramientas de edición en el menú capa.

Creación de una vista y carga de una capa

Añadiremos:

- LandRegistryViewerExtension

Modificaremos :

- pom.xml

1. Vamos a trabajar con el proyecto *org.gvsig.landregistryviewer.app.extension*.
2. Para poder acceder a las clases de gvSIG deberemos añadir sus dependencias. Para ello, primero las añadiremos al apartado *dependencyManagement* del pom.xml del proyecto padre (*org.gvsig.landregistryviewer.app*):

```
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.app</artifactId>
  <version>2.0-SNAPSHOT</version>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.fmap.control</artifactId>
  <version>2.0-SNAPSHOT</version>
</dependency>
```

   A continuación las añadiremos al pom.xml del proyecto *org.gvsig.landregistryviewer.app.extension* dentro del apartado *dependencies*, aunque esta vez sin la versión:

```
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.app</artifactId>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.fmap.control</artifactId>
</dependency>
```

3. Regeneraremos los proyecto de eclipse con *mvn eclise-eclipse*
4. Añadiremos el siguiente código a *LandRegistryViewerExtension.java* :

```
private static final String VIEW_NAME = "Land registry viewer";
private static final Logger LOG =
LoggerFactory.getLogger(LandRegistryViewerExtension.class);

/**
  * Create the view in the project. Add the blocks layer
  * to the view, and register the tool for get info of the
  * blocks.
  */
  private void initializeView() throws LoadLayerException {
    ApplicationManager application = ApplicationLocator.getManager();

    ProjectManager projectManager = application.getProjectManager();

    // 1. Create a new view and set the name.
    ViewManager viewManager = (ViewManager)
projectManager.getDocumentManagers(ViewManager.TYPENAME);
    ViewDocument view = (ViewDocument) viewManager.createDocument();
    view.setName(VIEW_NAME);

    // 2. Create a new layer with the blocks
    FLyrVect layer = (FLyrVect)
application.getMapContextManager().createLayer("Blocks", getBlocks());
```

```
      // 3. Add this layer to the mapcontext of the new view.
      view.getMapContext().getLayers().addLayer(layer);

      // 4. Add the view to the current project.
      projectManager.getCurrentProject().add(view);

      // 5. Get the panel/iwindow of the view
      IView viewPanel = (IView) viewManager.getMainWindow(view);

      // 6. Show the view
      application.getUIManager().addCentredWindow((IWindow) viewPanel);
      try {
        application.getUIManager().setMaximum((IWindow) viewPanel, true);
      } catch (PropertyVetoException e) {
        LOG.error("Can't maximize the view",e);
      }
    }

    /**
      * Get a resource as a File from a path name in the
      * class path.
      *
      * @param pathname
      *
      * @return resource as a File
      */
    private File getResource(String pathname) {
      URL res = this.getClass().getClassLoader().getResource(pathname);
      return new File(res.getPath());
    }

    /**
      * Open the file as a feature store of type shape.
      *
      * @param shape file to be opened
      *
      * @return the feature store
      */
    private FeatureStore openShape(File shape) {
      try {

        DataStoreParameters parameters;
        DataManager manager = DALLocator.getDataManager();

        parameters = manager.createStoreParameters("Shape");
          parameters.setDynValue("shpfile", shape);
          parameters.setDynValue("crs", "EPSG:23030");
          return (FeatureStore) manager.openStore("Shape", parameters);

      } catch (InitializeException e) {
        LOG.error(e.getMessageStack());
        throw new RuntimeException(e);
      } catch (ProviderNotRegisteredException e) {
        LOG.error(e.getMessageStack());
        throw new RuntimeException(e);
      } catch (ValidateDataParametersException e) {
        LOG.error(e.getMessageStack());
        throw new RuntimeException(e);
      }
    }

    private FeatureStore getBlocks() {
      return openShape( getResource("data/blocks.shp") );
    }
```

5. Modificaremos el metodo *postInitialize* para añadir la llamada a *initializeView* :

```
  try {
    this.initializeView();
```

```
  } catch (LoadLayerException e) {
    LOG.error("Error inicializando la vista",e);
  }
```

6. Copiaremos los ficheros de datos con la cartografía a la carpeta *resources* del proyecto, de forma que en la carpeta *resources* nos quede una carpeta *data* con los *shapes*.
7. Ejecutaremos un *mvn install* para desplegar los cambios sobre gvSIG.
8. Probaremos a arrancar de nuevo la aplicación desde el launcher de gvSIG.

Desarrollo de la librería

Creación del API de Land registry viewer

Añadiremos:

- LandRegistryViewerBlock
- LandRegistryViewerProperty

Modificaremos :

- pom.xml
- LandRegistryViewerManager
- LandRegistryViewerManagerTest

Y borraremos:

- LandregistryViewerService
- LandRegistryViewerMessageException
- LandRegistryViewerServiceTest

1. Vamos a trabajar con el proyecto de eclipse *org.gvsig.landregistryviewer.lib.api*.
2. Añadiremos como dependencias de compilación la *librería de acceso a datos* y la de *geometrías* en el apartado *dependencyManagement* del proyecto raíz *org.gvsig.landregistryviewer*:

```
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.fmap.geometry</artifactId>
  <version>2.0-SNAPSHOT</version>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.fmap.dal</artifactId>
  <version>2.0-SNAPSHOT</version>
</dependency>
```

3. Editaremos el pom.xml de *org.gvsig.landregistryviewer.lib.api* y añadiremos las dependencias anteriores en el apartado *dependencies*, sin la versión:

```
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.fmap.geometry</artifactId>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.fmap.dal</artifactId>
</dependency>
```

4. Ejecutaremos un *mvn eclipse-eclipse*.
5. Modificaremos *LandRegistryViewerManager* sustituyendolo por:

```
public interface LandRegistryViewerManager {

  public void initialize(FeatureStore properties, FeatureStore blocks);

  public void initialize(File properties, File blocks);

  public FeatureStore getProperties();

  public FeatureStore getBlocks();
```

```
    public LandRegistryViewerBlock getBlock(Geometry point)
        throws LandRegistryViewerException;

}
```

6. Borraremos el interface *LandregistryViewerService*.
7. Añadiremos el interface:

```
public interface LandRegistryViewerBlock {

    /**
     * Returns the {@link LandRegistryViewerManager}
     *
     * @return {@link LandRegistryViewerManager}
     * @see {@link LandRegistryViewerManager}
     */
    public LandRegistryViewerManager getManager();

    /**
     * Returns the LandRegistryViewer's shape of the block.
     *
     * @return the shape associated to a LandRegistryViewerBlock as a Geometry
     * @throws LandRegistryViewerMessageException
     *             if there is an error getting the LandRegistryViewer's message
     */
    public Geometry getShape();

    public List<LandRegistryViewerProperty> getProperties() throws
LandRegistryViewerException;

}
```

8. Crearemos el interface:

```
public interface LandRegistryViewerProperty {

    /**
     * Returns the {@link LandRegistryViewerManager}
     *
     * @return {@link LandRegistryViewerManager}
     * @see {@link LandRegistryViewerManager}
     */
    public LandRegistryViewerManager getManager();

    public String getCode();

    public int getCreationDate();

    public Geometry getShape();

    public int getMunicipalityCode();

}
```

9. Borraremos la clase *LandRegistryViewerMessageException*.
10. Ahora tendremos que actualizar los test.
11. Borraremos la clase *LandRegistryViewerServiceTest*
12. Modificaremos la clase:

```
public abstract class LandRegistryViewerManagerTest extends
  AbstractLibraryAutoInitTestCase {

  private static final Logger LOG =
LoggerFactory.getLogger(LandRegistryViewerManagerTest.class);
```

```
  protected LandRegistryViewerManager manager;

  private File getResource(String pathname) {
    URL res = this.getClass().getClassLoader().getResource(pathname);
    return new File(res.getPath());
  }

  @Override
  protected void doSetUp() throws Exception {
    manager = LandRegistryViewerLocator.getManager();
  }

  public void testManagerInitialize() throws Exception {
    try {
      manager.initialize(
          getResource("data/properties.shp"),
          getResource("data/blocks.shp")
      );
    } catch (Exception e) {
      LOG.error("Can't initialize manager", e);
      throw e;
    }
  }

  public void testGetBlock() throws Exception {
    // TODO
  }

  public void testGetProperties() throws Exception {
    // TODO
  }

}
```

13. Haremos un *mvn install* para compilar y desplegar el API.
14. Al haber cambiado el API el resto de proyectos deberán tener errores, pero este no.

Modificar la implementación de lib

Añadiremos :

- DefaultLandRegistryViewerBlock
- DefaultLandRegistryViewerProperty
- IntersectsEvaluator
- pom.xml

Modificaremos :

- DefaultLandRegistryViewerManager

Y borraremos:

- DefaultLandRegistryViewerService
- DefaultLandRegistryViewerServiceTest

1. Vamos a trabajar con el proyecto *org.gvsig.landregistryviewer.lib.impl*
2. Actualizaremos la configuración de proyecto con un *mvn eclipse-eclipse*.
3. Borraremos la clase *DefaultLandRegistryViewerService*.
4. Modificaremos la clase:

```
public class DefaultLandRegistryViewerManager implements LandRegistryViewerManager
{
```

```
  private static final Logger LOG =
LoggerFactory.getLogger(DefaultLandRegistryViewerManager.class);

  private FeatureStore properties;
  private FeatureStore blocks;

  public void initialize(FeatureStore properties, FeatureStore blocks) {
    this.properties = properties;
    this.blocks = blocks;
  }

  public void initialize(File properties, File blocks) {
    this.initialize(openShape(properties), openShape(blocks));
  }

  public LandRegistryViewerBlock getBlock(Geometry point)
      throws LandRegistryViewerException {

    FeatureSet set = null;
    DisposableIterator it = null;

    try {
      String attrGeomName =
blocks.getDefaultFeatureType().getDefaultGeometryAttributeName();
      FeatureQuery query = blocks.createFeatureQuery();
      query.setFilter( new IntersectsEvaluator(attrGeomName,point) );
      set = blocks.getFeatureSet(query);
      if( set.isEmpty() ) {
        return null;
      }
      it = set.iterator();
      Feature f = (Feature) it.next();
      LandRegistryViewerBlock block = new DefaultLandRegistryViewerBlock(
          this,
          f.getGeometry(attrGeomName)
        );
      return block;

    } catch (DataException e) {
      throw new LandRegistryViewerException(e);
    } catch (GeometryOperationNotSupportedException e) {
      throw new LandRegistryViewerException(e);
    } catch (GeometryOperationException e) {
      throw new LandRegistryViewerException(e);
    } finally {
      if( it != null ) {
        it.dispose();
      }
      if( set != null ) {
        set.dispose();
      }
    }
  }

  public FeatureStore getProperties() {
    return this.properties;
  }

  public FeatureStore getBlocks() {
    return this.properties;
  }

  /**
   * Open the file as a feature store of type shape.
   *
   * @param shape file to be opened
   *
   * @return the feature store
   */
```

```
  private FeatureStore openShape(File shape) {
    try {

      DataStoreParameters parameters;
      DataManager manager = DALLocator.getDataManager();

      parameters = manager.createStoreParameters("Shape");
      parameters.setDynValue("shpfile", shape);
      parameters.setDynValue("crs", "EPSG:23030");
      return (FeatureStore) manager.openStore("Shape", parameters);

    } catch (InitializeException e) {
      LOG.error(e.getMessageStack());
      throw new RuntimeException(e);
    } catch (ProviderNotRegisteredException e) {
      LOG.error(e.getMessageStack());
      throw new RuntimeException(e);
    } catch (ValidateDataParametersException e) {
      LOG.error(e.getMessageStack());
      throw new RuntimeException(e);
    }
  }

}
```

5. Crearemos la clase:

```
public class DefaultLandRegistryViewerBlock implements LandRegistryViewerBlock {

  private static final Logger LOG =
LoggerFactory.getLogger(DefaultLandRegistryViewerBlock.class);

  private static final String PROPERTIES_CODE = "PARCELA";
  private static final String PROPERTIES_CREATIONDATE = "FECHAALTA";
  private static final String PROPERTIES_MUNICODE = "MUNICIPIO";

  private DefaultLandRegistryViewerManager manager;
  private Geometry shape;

  /**
   * {@link DefaultLandRegistryViewerBlock} constructor with a
   * {@link LandRegistryViewerManager}.
   *
   * @param manager
   *            to use in the service
   */
  public DefaultLandRegistryViewerBlock(DefaultLandRegistryViewerManager manager,
Geometry shape) {
      this.manager = manager;
      this.shape = shape;
  }

  public LandRegistryViewerManager getManager() {
      return this.manager;
  }

  public Geometry getShape() {
    return this.shape;
  }

  public List<LandRegistryViewerProperty> getProperties()
      throws LandRegistryViewerException {

    FeatureSet set = null;
    DisposableIterator it = null;
    List<LandRegistryViewerProperty> properties = new
ArrayList<LandRegistryViewerProperty>();

    try {
```

```
      FeatureStore store = this.manager.getProperties();
      String attrGeomName =
store.getDefaultFeatureType().getDefaultGeometryAttributeName();
      FeatureQuery query = store.createFeatureQuery();
      query.setFilter( new IntersectsEvaluator(attrGeomName, this.shape) );
      set = this.manager.getProperties().getFeatureSet(query);
      if( set.isEmpty() ) {
        return null;
      }
      it = set.iterator();
      while( it.hasNext() ) {
        Feature f = (Feature) it.next();
        LandRegistryViewerProperty property = new
DefaultLandRegistryViewerProperty(
          this.manager,
          f.getString(PROPERTIES_CODE),
          f.getGeometry(attrGeomName),
          f.getInt(PROPERTIES_CREATIONDATE),
          f.getInt(PROPERTIES_MUNICODE)
        );
        properties.add(property);
      }
      return properties;

    } catch (DataException e) {
      throw new LandRegistryViewerException(e);
    } catch (GeometryOperationNotSupportedException e) {
      throw new LandRegistryViewerException(e);
    } catch (GeometryOperationException e) {
      throw new LandRegistryViewerException(e);
    } finally {
      if( it != null ) {
        it.dispose();
      }
      if( set != null ) {
        set.dispose();
      }
    }
  }

}
```

6. Crearemos la clase:

```
public class DefaultLandRegistryViewerProperty implements
LandRegistryViewerProperty {

  private static final Logger LOG =
LoggerFactory.getLogger(DefaultLandRegistryViewerProperty.class);

  private String code;
  private int municipioCode;
  private int creationDate;
  private Geometry shape;
  private DefaultLandRegistryViewerManager manager;

  public DefaultLandRegistryViewerProperty(DefaultLandRegistryViewerManager
manager,
    String code, Geometry shape, int creationDate, int municipioCode) {

    this.manager = manager;
    this.code = code;
    this.municipioCode= municipioCode;
    this.creationDate = creationDate;
    this.shape = shape;
  }

  public String getCode() {
    return this.code;
```

```
    }

    public int getCreationDate() {
      return this.creationDate;
    }

    public LandRegistryViewerManager getManager() {
      return this.manager;
    }

    public int getMunicipalityCode() {
      return this.municipioCode;
    }

    public Geometry getShape() {
      return this.shape;
    }

  }
```

7. Crearemos la clase:

```
  public class IntersectsEvaluator extends AbstractEvaluator {

    private Geometry op1geom;
    private String op2attrname;
    private String where;

    public IntersectsEvaluator(String op1attrname, Geometry op2geom)
      throws GeometryOperationNotSupportedException, GeometryOperationException {

      this.op1geom = op2geom;
      this.op2attrname = op1attrname;
      this.where = MessageFormat.format(
        " intersects(GeomFromText('{1}','{2}'),{0}) ",
        new Object[] {
          this.op1geom.convertToWKT(),
          "", // this.op2geom.getCRS()
          this.op2attrname
        }
      );
    }

    public String getName() {
      return "intersets";
    }

    public Object evaluate(EvaluatorData data) throws EvaluatorException {
      Geometry op1geom = (Geometry) data.getDataValue(this.op2attrname);
      try {

        return new Boolean(this.op1geom.intersects(op1geom));

      } catch (GeometryOperationNotSupportedException e) {
        throw new EvaluatorException(e);
      } catch (GeometryOperationException e) {
        throw new EvaluatorException(e);
      }
    }

    public String getCQL() {
      return this.where;
    }

  }
```

8. Eliminar *DefaultLandRegistryViewerServiceTest*.

9. Copiaremos la carpeta *data* de los recursos de *org.gvsig.landregistryviewer.app.extension* a la carpeta de recursos de *org.gvsig.landregistryviewer.lib.impl*.

   Seran necesarios para que pasen los test.

10. Añadiremos una serie de dependencias de ejecución al *dependencyManagement* para que funcionen los tests en el pom del proyecto *org.gvsig.landregistryviewer*:

```
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.fmap.dal</artifactId>
  <version>2.0-SNAPSHOT</version>
  <classifier>impl</classifier>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.fmap.dal.file</artifactId>
  <version>2.0-SNAPSHOT</version>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.fmap.dal.file</artifactId>
  <version>2.0-SNAPSHOT</version>
  <classifier>store.dbf</classifier>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.fmap.dal.file</artifactId>
  <version>2.0-SNAPSHOT</version>
  <classifier>store.shp</classifier>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.fmap.geometry</artifactId>
  <version>2.0-SNAPSHOT</version>
  <classifier>impl</classifier>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.fmap.geometry</artifactId>
  <version>2.0-SNAPSHOT</version>
  <classifier>operation</classifier>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.metadata</artifactId>
  <version>2.0-SNAPSHOT</version>
  <classifier>simple</classifier>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.projection</artifactId>
  <version>2.0-SNAPSHOT</version>
  <classifier>cresques-impl</classifier>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.fmap.dal</artifactId>
  <version>2.0-SNAPSHOT</version>
  <classifier>spi</classifier>
  <scope>runtime</scope>
```

```
  </dependency>
  <dependency>
    <groupId>org.gvsig</groupId>
    <artifactId>org.gvsig.compat</artifactId>
    <version>2.0-SNAPSHOT</version>
  </dependency>
  <dependency>
    <groupId>org.gvsig</groupId>
    <artifactId>org.gvsig.compat</artifactId>
    <version>2.0-SNAPSHOT</version>
    <classifier>se</classifier>
    <scope>runtime</scope>
  </dependency>
```

11. A continuación las añadiremos al apartado *dependencies* del pom.xml del proyecto *org.gvsig.landregistryviewer.lib.impl*, ya sin versión:

```
  <dependency>
    <groupId>org.gvsig</groupId>
    <artifactId>org.gvsig.fmap.dal</artifactId>
    <classifier>impl</classifier>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.gvsig</groupId>
    <artifactId>org.gvsig.fmap.dal.file</artifactId>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.gvsig</groupId>
    <artifactId>org.gvsig.fmap.dal.file</artifactId>
    <classifier>store.dbf</classifier>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.gvsig</groupId>
    <artifactId>org.gvsig.fmap.dal.file</artifactId>
    <classifier>store.shp</classifier>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.gvsig</groupId>
    <artifactId>org.gvsig.fmap.geometry</artifactId>
    <classifier>impl</classifier>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.gvsig</groupId>
    <artifactId>org.gvsig.fmap.geometry</artifactId>
    <classifier>operation</classifier>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.gvsig</groupId>
    <artifactId>org.gvsig.metadata</artifactId>
    <classifier>simple</classifier>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.gvsig</groupId>
    <artifactId>org.gvsig.projection</artifactId>
    <classifier>cresques-impl</classifier>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.gvsig</groupId>
    <artifactId>org.gvsig.fmap.dal</artifactId>
    <classifier>spi</classifier>
```

```
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.gvsig</groupId>
    <artifactId>org.gvsig.compat</artifactId>
  </dependency>
  <dependency>
    <groupId>org.gvsig</groupId>
    <artifactId>org.gvsig.compat</artifactId>
    <classifier>se</classifier>
    <scope>runtime</scope>
  </dependency>
```

12.Ejecutamos un *mvn eclipse-eclipse* de *org.gvsig.landregistryviewer.lib.impl*.

13.Ejecutamos un *mvn install* de *org.gvsig.landregistryviewer.lib.impl*.

Desarrollo de la librería de componentes swing

Creación del api de swing

Añadiremos :

- JLandRegistryViewerBlockPanel

Modificaremos :

- LandRegistryViewerSwingManager

Y borraremos:

- JLandRegistryViewerServicePanel

1. Abriremos en el arbol de proyectos *org.gvsig.landregistryviewer.swing.api* para trabajar con él.
2. Eliminaremos la clase *JLandRegistryViewerServicePanel*.
3. Modificaremos la clase :

```
public interface LandRegistryViewerSwingManager {

    /**
     * Returns the panel associated to a {@link LandRegistryViewerBlock}.
     *
     * @param cookie
     *            {@link LandRegistryViewerBlock} contained on the panel
     * @return a {@link JLandRegistryViewerBlockPanel} with the panel of the
     *         {@link LandRegistryViewerBlock}
     * @see JLandRegistryViewerBlockPanel
     * @see LandRegistryViewerBlock
     */
    public JLandRegistryViewerBlockPanel createJLandRegistryViewerBlockPanel(
        LandRegistryViewerBlock block);

    /**
     * Returns the {@link LandRegistryViewerManager}.
     *
     * @return {@link LandRegistryViewerManager}
     * @see {@link LandRegistryViewerManager}
     */
    public LandRegistryViewerManager getManager();

    /**
     * Returns the translation of a string.
     *
     * @param key
     *            String to translate
     * @return a String with the translation of the string passed by parameter
     */
    public String getTranslation(String key);

    /**
     * Registers a new instance of a WindowManager which provides services to
     * the management of the application windows.
     *
     * @param manager
     *            {@link LandRegistryViewerWindowManager} to register in the
     *            ScriptingUIManager.
     * @see LandRegistryViewerWindowManager
     */
    public void registerWindowManager(LandRegistryViewerWindowManager manager);

    /**
```

```
     * Returns the {@link LandRegistryViewerWindowManager}.
     *
     * @return {@link LandRegistryViewerWindowManager}
     * @see {@link LandRegistryViewerWindowManager}
     */
    public LandRegistryViewerWindowManager getWindowManager();
}
```

4. Añadiremos la clase :

```
public abstract class JLandRegistryViewerBlockPanel extends JPanel {

 /**
   *
   */
 private static final long serialVersionUID = 8697975258030694200L;

 /**
   * Returns the {@link LandRegistryViewerBlock} associated with the panel.
   *
   * @return the {@link LandRegistryViewerBlock}
   */
 public abstract LandRegistryViewerBlock getLandRegistryViewerBlock();

}
```

5. Borraremos todas las clases de dentro de los test de *swing.api* (3 clases).
6. Ejecutamos un *mvn install* de *org.gvsig.landregistryviewer.swing.api*.

Modificación de la implementación de swing

Añadiremos :

- DefaultJLandRegistryViewerBlockPanel

Modificaremos :

- DefaultLandRegistryViewerSwingManager

Y borraremos:

- DefaultJLandRegistryViewerServiceInfoPanel
- DefaultJLandRegistryViewerServicePanel

1. Abriremos en el arbol de proyectos *org.gvsig.landregistryviewer.swing.impl* para trabajar con el.
2. Borraremos las clases *DefaultJLandRegistryViewerServiceInfoPanel* y *DefaultJLandRegistryViewerServicePanel*.
3. Modificaremos la clase :

```
public class DefaultLandRegistryViewerSwingManager implements
    LandRegistryViewerSwingManager {

  private LandRegistryViewerManager manager;
  private I18nManager i18nmanager = null;
  private LandRegistryViewerWindowManager windowManager;

  public DefaultLandRegistryViewerSwingManager() {
      this.i18nmanager = ToolsLocator.getI18nManager();
      this.manager = LandRegistryViewerLocator.getManager();
      this.windowManager = new DefaultLandRegistryViewerWindowManager();
  }

  public JLandRegistryViewerBlockPanel createJLandRegistryViewerBlockPanel(
```

```
      LandRegistryViewerBlock block) {
        JLandRegistryViewerBlockPanel panel =
            new DefaultJLandRegistryViewerBlockPanel(this, block);
        return panel;
    }

    public LandRegistryViewerManager getManager() {
        return this.manager;
    }

    public String getTranslation(String key) {
        return this.i18nmanager.getTranslation(key);
    }

    public void registerWindowManager(LandRegistryViewerWindowManager manager) {
        this.windowManager = manager;
    }

    public LandRegistryViewerWindowManager getWindowManager() {
        return this.windowManager;
    }

}
```

4. Crearemos la clase :

```
public class DefaultJLandRegistryViewerBlockPanel extends
    JLandRegistryViewerBlockPanel {

  /**
    *
    */
  private static final long serialVersionUID = 3638250585757641443L;


  private static final Logger LOG =
        LoggerFactory.getLogger(DefaultJLandRegistryViewerBlockPanel.class);


  private LandRegistryViewerBlock block;
  private LandRegistryViewerSwingManager uimanager;

  protected JButton accept = null;

  public DefaultJLandRegistryViewerBlockPanel(
      DefaultLandRegistryViewerSwingManager uimanager, LandRegistryViewerBlock
block) {

      this.block = block;
      this.uimanager = uimanager;

      this.setLayout(new BorderLayout());
      this.setPreferredSize(new Dimension(550, 150));

      JLabel text = new JLabel(getText());

      JScrollPane scrollPane = new JScrollPane(text);
      scrollPane.setPreferredSize(new Dimension(550, 150));

      // TODO: replace with the UsabilitySwingManager.createJButton()
      accept = new JButton(this.uimanager.getTranslation("Accept"));

      JPanel optionsPane = new JPanel();
      optionsPane.setLayout(new BoxLayout(optionsPane, BoxLayout.LINE_AXIS));
      optionsPane.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
      optionsPane.add(Box.createHorizontalGlue());

      accept.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
```

```
            setVisible(false);
          }
        });
        optionsPane.add(accept);
        optionsPane.add(Box.createRigidArea(new Dimension(10, 0)));

        this.add(scrollPane, BorderLayout.CENTER);
        this.add(optionsPane, BorderLayout.SOUTH);
    }

    public LandRegistryViewerBlock getLandRegistryViewerBlock() {
      return this.block;
    }

    private String getText() {
      try {
        List<LandRegistryViewerProperty> properties = this.block.getProperties();

        StringBuffer buffer = new StringBuffer();
        buffer.append("<html>\n<body>\n");
        buffer.append("<p><b>Block</b></p>\n<br>\n");
        buffer.append("<p>has ");
        buffer.append(properties.size());
        buffer.append("properties.</p>\n<br>\n");
        buffer.append("<ul>\n");

        Iterator<LandRegistryViewerProperty> it = properties.iterator();
        while( it.hasNext() ) {
          LandRegistryViewerProperty property = it.next();
          buffer.append("<li>Property code ");
          buffer.append(property.getCode());
          buffer.append(" of municipality  ");
          buffer.append(property.getMunicipalityCode());
          buffer.append(" (");
          buffer.append(property.getCreationDate());
          buffer.append(")</li>\n");
        }
        buffer.append("</ul>\n");
        buffer.append("</body>\n</html>\n");
        return buffer.toString();

      } catch (LandRegistryViewerException e) {
        LOG.error("Can't create description of Block",e);

        StringBuffer buffer = new StringBuffer();
        buffer.append("<html>\n<body>\n");
        buffer.append("<p>Se ha producido un error componiendo la" +
            "descripcion de la manzana.</p>\n");
        buffer.append("<pre>\n");
        buffer.append(e.getMessageStack());
        buffer.append("\n</pre>\n");
        buffer.append("</body>\n</html>\n");
        return buffer.toString();
      }
    }
  }
```

5. Borraremos todas las clases de dentro de los test de *swing.impl* (3 clases).
6. Ejecutamos un *mvn install* de *org.gvsig.landregistryviewer.swing.impl*.

La aplicación de pruebas

Preparando una aplicación de pruebas

Modificaremos :

- Main
- pom.xml

1. Abriremos en el arbol de proyectos *org.gvsig.landregistryviewer.main* para trabajar con él.
2. Modificaremos la clase:

```
public class Main {

  private static final Logger LOG = LoggerFactory.getLogger(Main.class);

  private static final String SHOWINFO_TOOL_NAME = "LandRegistryViewer.infotool";

  private LandRegistryViewerManager manager;

  private MapControlManager mapControlManager;

  private MapControl mapControl;

  public static void main(String args[]) throws MapControlCreationException,
LocatorException, LoadLayerException {
      new DefaultLibrariesInitializer().fullInitialize();
      Main main = new Main();
      main.show();
  }

  @SuppressWarnings("serial")
  public void show() throws MapControlCreationException, LocatorException,
LoadLayerException {
    mapControlManager = MapControlLocator.getMapControlManager();
    manager = LandRegistryViewerLocator.getManager();

    mapControl = mapControlManager.createJMapControlPanel();
    mapControl.addBehavior(
      "zoom",
      new Behavior[] {
        new RectangleBehavior(new ZoomInListenerImpl(mapControl)),
        new PointBehavior(new ZoomOutRightButtonListener(mapControl))
      }
    );
    mapControl.addBehavior(
      "pan",
      new MoveBehavior(
         new PanListenerImpl(mapControl)
      )
    );
    mapControl.setTool("pan");


    Action exit = new AbstractAction("Exit") {
        public void actionPerformed(ActionEvent e) {
            System.exit(0);
        }
    };

    JFrame frame = new JFrame("LandRegistryViewer example app");
    frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

    // Create the menu bar.
    JMenuBar menuBar = new JMenuBar();
```

```
    // Build the menu.
    JMenu menuFile = new JMenu("File");
    menuFile.add(new JMenuItem(exit));

    menuBar.add(menuFile);

    JToolBar toolBar = new JToolBar();
    toolBar.add(
          new JButton(
             new AbstractAction("Pan") {
                public void actionPerformed(ActionEvent e) {
                    mapControl.setTool("pan");
                }
            }
          )
    );
    toolBar.add(
          new JButton(
             new AbstractAction("Zoom") {
                public void actionPerformed(ActionEvent e) {
                    mapControl.setTool("zoom");
                }
            }
          )
    );
    toolBar.add(
          new JButton(
             new AbstractAction("Zoom all") {
                public void actionPerformed(ActionEvent e) {
                    zoomAll();
                }
            }
          )
    );

    toolBar.add(
          new JButton(
             new AbstractAction("Info") {
                public void actionPerformed(ActionEvent e) {
                  mapControl.setTool(SHOWINFO_TOOL_NAME);;
                }
            }
          )
    );

    toolBar.add(new JButton(exit));
    frame.setPreferredSize(new Dimension(400, 300));
    frame.setJMenuBar(menuBar);
    frame.add(toolBar, BorderLayout.PAGE_START);
    frame.add(mapControl,BorderLayout.CENTER);

    // Display the window.
    frame.pack();
    frame.setVisible(true);

    LOG.info("Providers: " +
DALLocator.getDataManager().getStoreProviders().toString());

    // Add the blocks layer
    manager.initialize(
        getResource("data/properties.shp"),
        getResource("data/blocks.shp")
    );
    FLyrVect layer = (FLyrVect) MapContextLocator.
      getMapContextManager().createLayer(
      "Blocks",
      manager.getBlocks()
    );
```

```
    mapControl.getMapContext().getLayers().addLayer(layer);

    PropertiesOfBlockListener listener = new PropertiesOfBlockListener();
    mapControl.addBehavior(SHOWINFO_TOOL_NAME, new PointBehavior(listener));

    zoomAll();
  }

  private void zoomAll() {
    MapContext mapContext = mapControl.getMapContext();
    Envelope all = mapContext.getLayers().getFullEnvelope();
    LOG.info("Full extents "+all.toString());
    mapContext.getViewPort().setEnvelope(all);
    mapContext.invalidate();
  }

  private File getResource(String pathname) {
    URL res = this.getClass().getClassLoader().getResource(pathname);
    return new File(res.getPath());
  }

  public class PropertiesOfBlockListener extends AbstractPointListener {

    public void point(PointEvent event) throws BehaviorException {
      LandRegistryViewerSwingManager swingManager =
LandRegistryViewerSwingLocator.getSwingManager();

      LandRegistryViewerBlock block;
      try {
        block = swingManager.getManager().getBlock(event.getMapPoint());
        if( block == null ) {
          return;
        }
        JPanel panel = swingManager.createJLandRegistryViewerBlockPanel(block);
        swingManager.getWindowManager().showWindow(panel, "Block information",
LandRegistryViewerWindowManager.MODE_TOOL);
      } catch (LandRegistryViewerException e) {
        // FIXME: Process exception
        throw new RuntimeException("Can't show properties of selected block.",e);
      }
    }

  }

}
```

3. Añadiremos una serie de dependencias de compilación y ejecución adicionales al *dependencyManagement* en el pom del proyecto *org.gvsig.landregistryviewer*:

```
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.fmap.control</artifactId>
  <version>2.0-SNAPSHOT</version>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.symbology</artifactId>
  <version>2.0-SNAPSHOT</version>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.symbology</artifactId>
  <version>2.0-SNAPSHOT</version>
  <classifier>impl</classifier>
  <scope>runtime</scope>
</dependency>
```

4. A continuación añadiremos al apartado *dependencies* del pom.xml del proyecto *org.gvsig.landregistryviewer.main* todas las dependencias de compilacion y ejecución que necesita, ya sin versión:

```
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.fmap.dal</artifactId>
  <classifier>impl</classifier>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.fmap.dal.file</artifactId>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.fmap.dal.file</artifactId>
  <classifier>store.dbf</classifier>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.fmap.dal.file</artifactId>
  <classifier>store.shp</classifier>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.fmap.geometry</artifactId>
  <classifier>impl</classifier>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.fmap.geometry</artifactId>
  <classifier>operation</classifier>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.metadata</artifactId>
  <classifier>simple</classifier>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.projection</artifactId>
  <classifier>cresques-impl</classifier>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.fmap.dal</artifactId>
  <classifier>spi</classifier>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.compat</artifactId>
</dependency>
<dependency>
  <groupId>org.gvsig</groupId>
  <artifactId>org.gvsig.compat</artifactId>
  <classifier>se</classifier>
  <scope>runtime</scope>
</dependency>
<dependency>
```

```
    <groupId>org.gvsig</groupId>
    <artifactId>org.gvsig.fmap.control</artifactId>
  </dependency>
  <dependency>
    <groupId>org.gvsig</groupId>
    <artifactId>org.gvsig.symbology</artifactId>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.gvsig</groupId>
    <artifactId>org.gvsig.symbology</artifactId>
    <classifier>impl</classifier>
    <scope>runtime</scope>
  </dependency>
```

5. Haremos un *mvn eclipse-eclipse* del proyecto *org.gvsig.landregistryviewer.main*.
6. Haremos un *mvn install* para cerciorarnos de que compila (no sería necesario ya que no depende nadie de él).
7. Ejecutaremos el Main, *Run as->Java aplication*.
8. Comprobaremos que la aplicación hace lo que toca.

Usar la funcionalidad en la extensión

Modificaciones a la extensión para que use la librería

Añadiremos :

- PropertiesOfBlockListener

Y modificaremos :

- LandRegistryViewerExtension
- config.xml

1. Abriremos en el arbol de proyectos *org.gvsig.landregistryviewer.app.extension* para trabajar con él.
2. Modificaremos la clase, para que use la librería y registre la nueva *tool* sobre la vista :

```
public class LandRegistryViewerExtension extends Extension  implements
ExclusiveUIExtension  {

  private static final Logger LOG =
LoggerFactory.getLogger(LandRegistryViewerExtension.class);

  private static final String VIEW_NAME = "Land registry viewer";

  private static final String TOOL_NAME = "LandRegistryViewer.infotool";
  private static final String ACTION_SETINFOTOOL = "SetInfoTool";

  private LandRegistryViewerManager manager;
  private LandRegistryViewerSwingManager swingManager;

  private Set<String> extensionsToHidde = new HashSet<String>();

  public void initialize() {
    this.initializeVisibilityControl();
  }

  public void postInitialize() {
    try {
      manager = LandRegistryViewerLocator.getManager();
      swingManager = LandRegistryViewerSwingLocator.getSwingManager();
      swingManager.registerWindowManager(new
GvSIGLandRegistryViewerWindowManager());

      this.initializeStores();
      this.initializeView();

    } catch (LoadLayerException e) {
      LOG.error("Error in postinitialize.",e);
    }
  }

  /**
   * Execute the actions associated to this extension.
   */
  public void execute(String actionCommand) {
    if( ACTION_SETINFOTOOL.equalsIgnoreCase(actionCommand) ) {
      // 1. Set the tool in the mapcontrol of the active view.

      // 1.1 Get the application object
        ApplicationManager application = ApplicationLocator.getManager();

      // 1.2 Get the active window
      IView view = (IView) application.getUIManager().getActiveWindow();
```

```
        // 1.3 Set the tool in the mapcontrol
        view.getMapControl().setTool(TOOL_NAME);
      }
    }

    /**
     * Check if tools of this extension are enabled.
     */
    public boolean isEnabled() {
      //
      // By default the tool is always enabled
      //
      return true;
    }

    /**
     * Check if tools of this extension are visible.
     */
    public boolean isVisible() {
      //
      // The tools only is visible when is active my view.
      //

      ApplicationManager application = ApplicationLocator.getManager();

      // Get the active window
      IWindow win = application.getUIManager().getActiveWindow();

      // check if the active window is a View
      if( win == null || !(win instanceof IView) ) {
        return false;
      }

      // Check if the active view is my view.
            String name = ((IView)win).getDocument().getName();
            if( name!= null && name.startsWith(VIEW_NAME) ) {
        return true;
      }
      return false;
    }


    /**
     * Initialize the set of extension's names and set this
     * class to control the visibility of all extensions.
     */
  private void initializeVisibilityControl() {

    // Prepare a set with the names of the extensions to hide.
    String[] extensions = new String[] {
        "org.gvsig.editing.CreateNewLayer",
        "org.gvsig.editing.StartEditing",
        "org.gvsig.editing.StopEditing",
        "org.gvsig.editing.ExportTo",
        "org.gvsig.editing.UndoViewExtension",
        "org.gvsig.editing.RedoViewExtension",
        "org.gvsig.editing.ViewCommandStackExtension",
        "org.gvsig.editing.TableCommandStackExtension",
        "org.gvsig.editing.CADExtension",
        "org.gvsig.editing.ExploitExtension",
        "org.gvsig.editing.MoveGeometryExtension",
        "org.gvsig.editing.InsertPointExtension",
        "org.gvsig.editing.InsertMultiPointExtension",
        "org.gvsig.editing.InsertLineExtension",
        "org.gvsig.editing.InsertPolyLineExtension",
        "org.gvsig.editing.InsertPolygonExtension",
        "org.gvsig.editing.JoinExtension",
        "org.gvsig.editing.InternalPolygonExtension",
        "org.gvsig.editing.StretchExtension",
```

```
      "org.gvsig.editing.ComplexSelectionGeometryExtension",
      "org.gvsig.editing.SelectionGeometryExtension",
      "org.gvsig.editing.MatrixExtension",
      "org.gvsig.editing.AutoCompletePolygonExtension",
          "org.gvsig.editing.SplitGeometryCADToolExtension"
    };
    for( int i=0; i<extensions.length ; i++ ) {
      this.extensionsToHidde.add(extensions[i]);
    }

    // Set me to manage the visibility of all extensions
    PluginServices.setExclusiveUIExtension(this);
}

/**
 * Check if an extension is enabled.
 */
public boolean isEnabled(IExtension extension) {
  // Is the extension's name is in the set of extension to hide
  // always return false to disable the extension.
  if( this.extensionsToHidde.contains(extension.getClass().getName()) ) {
    return false;
  }
  // Relies in the extension to verify if is enabled
  return extension.isEnabled();
}

/**
 * Check if an extension is visible.
 */
public boolean isVisible(IExtension extension) {
  // Is the extension's name is in the set of extension to hide
  // always return false to hide the extension.
  if( this.extensionsToHidde.contains(extension.getClass().getName()) ) {
    return false;
  }
  // Relies in the extension to verify if is visible
  return extension.isVisible();
}

/**
 * Create the view in the project. Add the blocks layer
 * to the view, and register the tool for get info of the
 * blocks.
 */
private void initializeView() throws LoadLayerException {
    ApplicationManager application = ApplicationLocator.getManager();

    ProjectManager projectManager = application.getProjectManager();

    // 1. Create a new view and set the name.
    ViewManager viewManager = (ViewManager)
projectManager.getDocumentManagers(ViewManager.TYPENAME);
    ViewDocument view = (ViewDocument) viewManager.createDocument();
    view.setName(VIEW_NAME);

    // 2. Create a new layer with the blocks
    FLyrVect layer = (FLyrVect)
application.getMapContextManager().createLayer("Blocks",
this.manager.getBlocks());

    // 3. Add this layer to the mapcontext of the new view.
    view.getMapContext().getLayers().addLayer(layer);

    // 4. Add the view to the current project.
    projectManager.getCurrentProject().add(view);

    // 5. Force to show the view's window.
    IView viewPanel = (IView) viewManager.getMainWindow(view);
```

```
      application.getUIManager().addCentredWindow((IWindow) viewPanel);
      try {
        application.getUIManager().setMaximum((IWindow) viewPanel, true);
      } catch (PropertyVetoException e) {
        LOG.error("Can't maximize view.",e);
      }

      // 6. Register my tool in the mapcontrol of the view.
      PropertiesOfBlockListener listener = new PropertiesOfBlockListener();
      viewPanel.getMapControl().addBehavior(TOOL_NAME, new PointBehavior(listener));

    }

    /**
     * Open the stores and initialize the logic manager whit this
     * stores.
     */
    private void initializeStores() {
      manager.initialize(
          getResource("data/properties.shp"),
          getResource("data/blocks.shp")
      );
    }

    /**
     * Get a resource as a File from a path name in the
     * class path.
     *
     * @param pathname
     *
     * @return resource as a File
     */
    private File getResource(String pathname) {
      URL res = this.getClass().getClassLoader().getResource(pathname);
      return new File(res.getPath());
    }

  }
```

3. Añadiremos la clase :

```
  public class PropertiesOfBlockListener extends AbstractPointListener {

    public void point(PointEvent event) throws BehaviorException {
      LandRegistryViewerSwingManager swingManager =
  LandRegistryViewerSwingLocator.getSwingManager();

      LandRegistryViewerBlock block;
      try {
        block = swingManager.getManager().getBlock(event.getMapPoint());
        if( block == null ) {
          return;
        }
        JPanel panel = swingManager.createJLandRegistryViewerBlockPanel(block);
        swingManager.getWindowManager().showWindow(panel, "Block information",
  LandRegistryViewerWindowManager.MODE_TOOL);
      } catch (LandRegistryViewerException e) {
        // FIXME: Process exception
        throw new RuntimeException("Can't show properties of selected block.",e);
      }
    }

  }
```

4. Editaremos el config.xml :

```
  <?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<plugin-config>
  <depends plugin-name="org.gvsig.app" />
  <resourceBundle name="text"/>
  <libraries library-dir="lib"/>
  <extensions>
    <extension class-
name="org.gvsig.landregistryviewer.app.extension.LandRegistryViewerExtension"
      description=""
      active="true"
      priority="1">
      <menu text="Vista/Land registry view info"
        position="11"
        action-command="SetInfoTool"/>
      <tool-bar name="LandRegistryViewer" position="1">
        <action-tool icon="view-query-information" tooltip="Land registry view
info"
          action-command="SetInfoTool" position="1"/>
      </tool-bar>
    </extension>
  </extensions>
</plugin-config>
```

5. Ejecutaremos un *mvn install* para desplegar los cambios sobre gvSIG, tanto sobre el proyecto *org.gvsig.landregistryviewer* como sobre el proyecto *org.gvsig.landregistryviewer.app*.
6. Probaremos a arrancar de nuevo la aplicación desde el launcher de gvSIG.