How to build a standard application

De Wiki
Aller à : navigation, rechercher
How to build a standard application

Sommaire

Why such a widget ?

The used widgets are relatively complex high level widgets as they allow to build a standard application GUI managing:

- context file loading and saving
- computation launching
- result files saving
- displaying input data as well as output ones

Using GDataPanelAbstract class

First, we will have to define how will be displayed data necessary to the application. To do that, we will create a class extending the GDataPanelAbstract one. In the following code example, we have assume:

- data needed for the computation are gathered in a single object corresponding to the Vehicle class. More generally, data could be dispatched in several objects but we have preferred to simplify the example using only one.
- the corresponding widget is defined via the WidVehicle class which owns a method getCalcObject() returning a Vehicle object.
- we added a console where output of the application will be displayed

```
public class WidSWDataPanel extends GDataPanelAbstract {

    private final WidVehicle widVehicle;

    /**
   * Constructor.
   * @throws GException   GENIUS exception
     */
    public WidSWDataPanel () throws GException {

        // Calling for the super constructor
        // "Data" is the name of the root XML name for the context files
```

```
        super("Data");

        // Creating a vehicle data widget
        widVehicle =  new WidVehicle();
        // Adding it in a tabbedpane
        this.addTab("Vehicle", widVehicle);

        // Adding an output console
        this.addConsoleTab("Output");

    }

    /**
   * Method returning a vehicle object needed by the application.
   * @return  object vehicle
     */
    public Vehicle getVehicle () { return widVehicle.getCalcObject(); }

    @Override
    public void display () throws GException { super.display(); }

    @Override
    public void clear() throws GException {}

}
```

Using GMainFrameAbstract class

Then we will create the main frame by extending the GMainFrameAbstract class.

By calling the super constructor of this class we will add as arguments:

- the title as a String
- the previous WidSWDataPanel object
- a GContextFileManagement object that will manage how to search for the context files (see specific GContextFileManagement topic)
- a GAboutDialog object that will display a pop up window for some information about the application.
- a GSaveResults object that will manage how to save result files.
- some dimension data
- a flag to display or not a progress bar

Then, the fact to extend from GMainFrameAbstract class imposes to fill the following methods:

- saveFilesManagement() that will define how to save result files (and eventually context file too)
- customPreProcessManagement() to execute some preprocessing before launching the application (see specific Process Management topic)
- customPostProcessManagement to execute some postprocessing after the end of the application (see specific Process Management topic)

Here is a code example:

```
public class WidSW extends GMainFrameAbstract<WidSWDataPanel> {

    // File Names
    /** Prefix for context file names */
    private static final String INI_FILE_PREFIX = "INI_";
    /** By default context file names */
    private static final String INI_FILE = "INIT.xml";
    /** Prefix for output file names */
    private static final String EPH_FILE_PREFIX = "EPH_";
    /** By default EPHEM file names */
    private static final String EPH_FILE = "EPHEM.txt";

    // SIZES

    /** Data panel height */
    private static final int DATAPANEL_HEIGHT = 400;
    /** Error console height */
    private static final int ERRCONSOLE_HEIGHT = 80;
    /** Icon size */
    private static  final int ICON_SIZE = 12;

    /**
   * Constructor
   * @throws GException GENIUS exception
     */
    public WidSW() throws GException {

        super("S/W",
              new WidSWDataPanel(),
              new GContextFileManagement(".", "SW", new
GFileFilter(INI_FILE_PREFIX, ".xml", "S/W Files") ),
              new GAboutDialog("About S/W", "Example S/W ...", "CNES",
"Vx.x ; xx/xx/2017", "/images/logoCNES.jpg"),
              new GSaveResults("Saving SW results", new File("results/"),
".txt", ".xml"),
              DATAPANEL_HEIGHT, ERRCONSOLE_HEIGHT, ICON_SIZE, true);

    }

    /**
   * Method managing the results (and context) files saving.
     */
    @Override
    protected void saveFilesManagement() throws GException {

        final File ini = new File("data/", INI_FILE);
        final File res = new File("results/", EPH_FILE);

        // The context file INIT.xml will be saved in INI_....xml
        this.getSaveResultsDialog().setContextFile("../data/"+ini.getName(),
INI_FILE_PREFIX, true);
```

```
        // Result files consist in a single one named by default "EPHEM"
        this.getSaveResultsDialog().clearResultFileList();
        this.getSaveResultsDialog().addSingleResultFile(res.getName(),
EPH_FILE_PREFIX, true);

        this.getSaveResultsDialog().show();

    }

    /**
   * Method for pre processing management just before running computation.
     */
    @Override
    protected void customPreProcessManagement() throws
GFileManipulatorException {

        // We write a context file with data coming from the data panel
        GFileManipulation.writeConfig("data/"+INI_FILE, "SW",
this.getDataPanel(), true);
        // We initialize the JavaCommandLauncher
       final String classPath = System.getProperty("java.class.path");
        this.getJavaCommandLauncher().setJavaCommand(classPath, new String[]
{"GStandardApplication.ihm.BatchSW"});
        // We display the console above the other tabbedpanes
        this.getDataPanel().selectConsoleTab();

    }

    /**
   * Method for post processing management.
     */
    @Override
    protected void customPostProcessManagement() {
        // Nothing to do ...
    }

}
```

The application launched by the GContextFileManagement object may be the following one.

Note that a GConsoleLogger class is now available to display exception messsages.

```
public class BatchSW {

    private Vehicle veh;
    private String ephemFileName;

    /**
   * Constructor
   * @param nomFicData    name of the context file.
```

```
   * @param nomFicEphem   name of the output file.
   * @throws IOException  GENIUS exception.
     */
    public BatchSW (final String nomFicData, final String nomFicEphem) throws
IOException {

        WidSWDataPanel dataPan = null;
        try {
            dataPan = new WidSWDataPanel();
            // Data reading inside the XML file
            GFileManipulation.readConfig(nomFicData, "SW", dataPan, true);
            // SW initialization
            veh = dataPan.getVehicle();
            }
        } catch (GException err) {
            GConsoleLogger.getLogger().log(Level.SEVERE, err.getMessage());
        }
        this.ephemFileName = nomFicEphem;

    }

    /**
   * Method to compute.
     */
    public void compute () {

        // Writing in the result EPHEM file
        FileWriter ephemFile = null;
        try {
            ephemFile = new FileWriter(ephemFileName);
            ephemFile.write("Initial total mass: " +
vehicle.getTotalMass()+"\n");
            ephemFile.close();
        } catch (IOException err) {
            GConsoleLogger.getLogger().log(Level.SEVERE, err.getMessage());
        }

        // Console display
        System.out.println("Initial total mass:" + vehicle.getTotalMass());
        for (int i = 0; i < 10; i++) {
            try {
                Thread.sleep(500);
            } catch (InterruptedException err) {
                GConsoleLogger.getLogger().log(Level.SEVERE,
err.getMessage());
            }
            System.out.println("@"+(int)((i+1)*10)+"@");
            System.out.println(i);
        }

    }
```

```
    public static void main(String[] args) {

        BatchSW batch;
        try {
            batch = new BatchSW("data/INIT.xml", "results/EPHEM.txt");
            batch.compute();
        } catch (IOException err) {
            GConsoleLogger.getLogger().log(Level.SEVERE, err.getMessage());
        }

    }

}
```

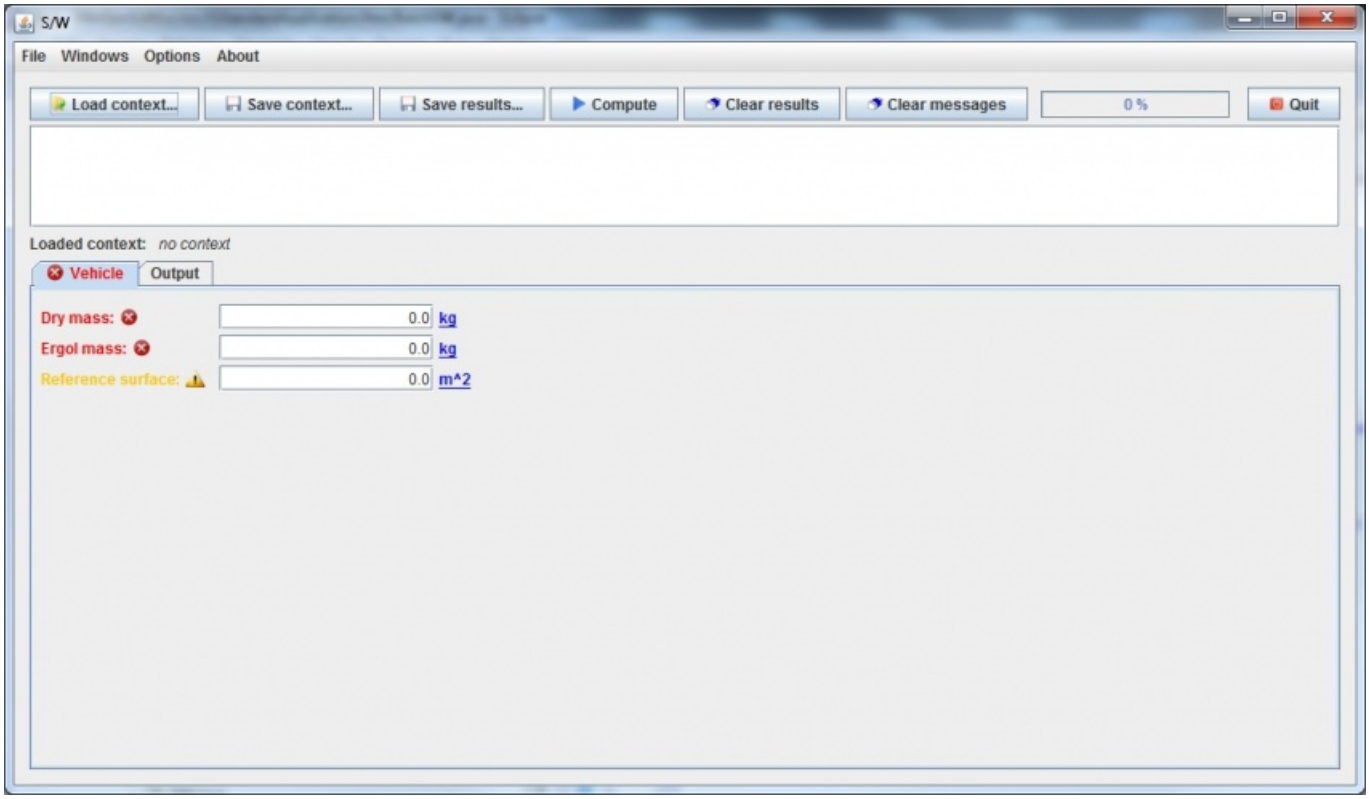
Calling the main frame

At last, you will just have to call in a main method, the previously defined WidSW object as below:

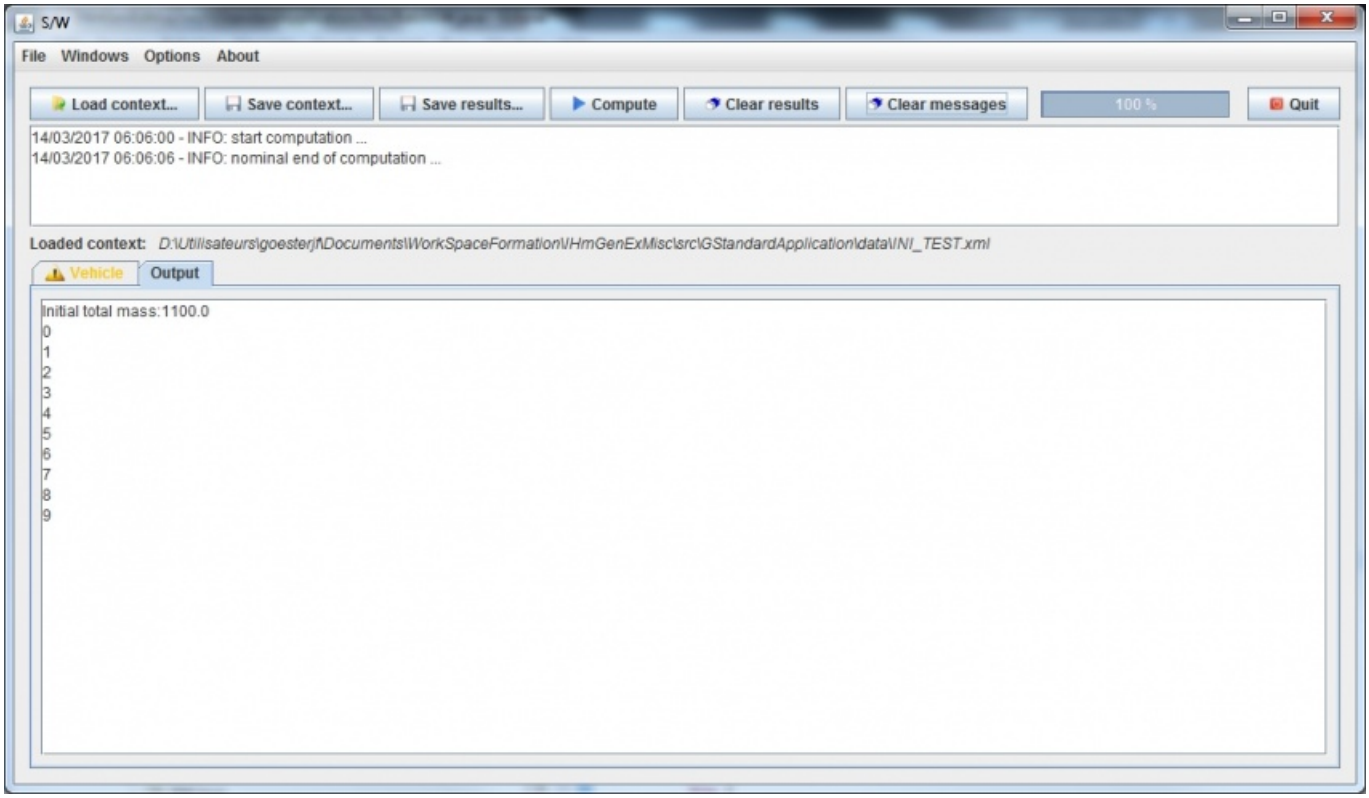
```
    public static void main(String[] args) throws GException {

        final WidSW pan = new WidSW();
        pan.displayMainFrame();

    }
```

!!! Warning: be careful to call the displayMainFrame() method and not the display() one !!!

... and you will obtain this:



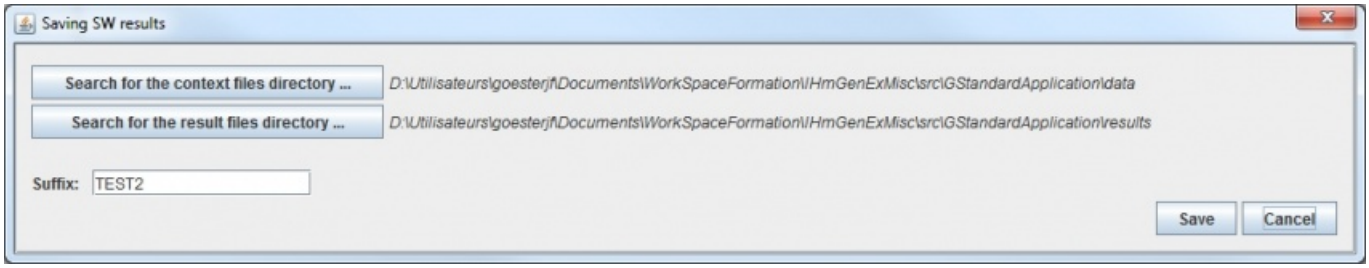
... and after loading a context file and executing the job:



We could see on the next image how saving results is managed:

- it is possible to choose a different directory for context and results files
- the common characteristic name of the saved files will be specified via a specific entry string.

Thus, in the example, INIT.xml will be saved as INI_TEST2.xml in data/ folder as EPHEM.txt file will be saved as EPH_TEST2.txt in the result/ folder.



Return to the introduction ↑ Go to the next page →

Menu de navigation

Outils personnels

- 52.14.130.13
- Discussion avec cette adresse IP
- Créer un compte
- Se connecter

Espaces de noms

- Page
- Discussion

Variantes

Affichages

- Lire
- Voir le texte source
- Historique
- Exporter en PDF

Plus

Rechercher

Rechercher Lire

GENIUS

- Welcome
- Quick Start
- News

Basic principles

- GFrame and GPanel
- Main widgets
- Links with Swing
- GLayout
- Conditional Display
- GListener interface

More deeper in the concept

- Units management
- GContainer
- GReadWrite interface and data files management
- Modified data
- Process management

Still more ...

- Validity controls
- Menu bar
- Icons
- GClear interface

Still more again ...

- Tooltips
- Shortcuts
- Copy & paste
- Plots
- Results File Management
- GPlotPanel
- GGroundPlotPanel
- Internationalization
- Log file
- Update data

Some other widgets

- GTabbedPane

- GTable1D
- GTable2D
- GComponentList
- GDialog and GDetachedPanel
- GContextFileManagement
- How to build a standard application
- GPanTest
- Create your own widget

Evolutions

- Main differences between V1.11.4 and V1.12.1
- Main differences between V1.10.1 and V1.11.4
- Main differences between V1.10 and V1.10.1
- Main differences between V1.9.1 and V1.10
- Main differences between V1.9 and V1.9.1
- Main differences between V1.8 and V1.9
- Main differences between V1.7 and V1.8
- Main differences between V1.6.2 and V1.7
- Main differences between V1.6.1 and V1.6.2
- Main differences between V1.6 and V1.6.1
- Main differences between V1.5 and V1.6
- Main differences between V1.4.1 and V1.5
- Main differences between V1.3 and V1.4.1

Training

- Training slides
- Tutorials package for V1.12.1
- Tutorials package for V1.11.4
- Tutorials package for V1.10.1
- Tutorials package for V1.9.1
- Training & tutorials package for V1.8
- Training & tutorials package for V1.7
- Training & tutorials package for V1.6

Links

- CNES freeware server

Outils

- Pages liées
- Suivi des pages liées
- Pages spéciales
- Adresse de cette version
- Information sur la page

- Citer cette page

- Dernière modification de cette page le 30 septembre 2020 à 12:48.

- Politique de confidentialité
- À propos de Wiki
- Avertissements
- Powered By MediaWiki