GReadWrite interface

De Wiki
Aller à : navigation, rechercher
GReadWrite interface

Sommaire

The GreadWrite Interface

GENIUS proposes a simply way to read and write into files, consistent with the display:

- By calling the GReadWrite interface
- By definition of the read() and write() methods calling the put() method, the mechanism being the same as for display (for example, using if/switch for conditionality).

Thus, if we consider that the data must be read or written with the same logic/order as for the display, we will put all inside the generic() method and the display(), read() and write() methods will only have to call the generic() one !

```
public class MyContainer extends GContainer implements GDisplay, GReadWrite {

  GEntryReal   valR;
  GEntryInt    valI;
  GEntryString valS;

  public MyContainer () {
    valR = new GEntryReal("Real value" , 0.);
    valI = new GEntryInt("Integer value", 0);
    valS = new GEntryString("Chain", "");
  }

  public void generic() throws GException {
    put(valR);
    put(valI);
    if ( valI.getValue() > 0 ) {
      put(valS);
    }
  }

  public void display() throws GException { generic(); }
```

```
  public void read()    throws GException { generic(); }
  public void write()   throws Gexception { generic(); }

}
```

So, in this example above:

1. the widget valS will be displayed only if the value of valI is strictly positive.
2. Moreover, the value of valS will be read only if the value of valI is strictly positive.
3. Again, the value of valS will be written into a file only if the value of valI is strictly positive.

We see that all have been centralized in the same code bloc.

On the contrary, if the logic is different for reading, writing and displaying data, we allways have the possibility to dissociate the treatments, coding them respectively inside the read(), write() and display() methods.

In the example below, for reading and writing, all data will be respectively read and written as, for display, valS will only be shown if the value of valI is strictly positive.

```
public class MyContainer extends GContainer implements GDisplay, GReadWrite {

  GEntryReal   valR;
  GEntryInt    valI;
  GEntryString valS;

  public MyContainer () {
    valR = new GEntryReal("Real value" , 0.);
    valI = new GEntryInt("Integer value", 0);
    valS = new GEntryString("Chain", "");
  }

  public void generic() throws GException {
    put(valR);
    put(valI);
    put(valS);
  }

  public void read()    throws GException { generic(); }
  public void write()   throws Gexception { generic(); }

  public void display() throws GException {
    put(valR);
    put(valI);
    if ( valI.getValue() > 0 ) {
      put(valS);
    }
  }

}
```

Data file management

To read (or write) using **GENIUS** tools, we only need to open a file and store inside the **GENIUS** corresponding object the data contained in this file. To do it:

- We use static methods from class GFileManipulation
- The file will be in XML MADONA format

Remark #1 : the **GENIUS** object may contain itself other **GENIUS** objects and so on ...

Remark #2 : in that example, the flag set for the last argument of these methods is to tell to **GENIUS** to add a "*" character after reading (if values changed) and to remove it after writing the configuration. For more details about modified data, see specific Modified data topic.

```
  MyGeniusObject obj = new MyGeniusObject (…); // Must implement the
GReadWrite interface

  GFileManipulation.readConfig  (fileName, XMLRootName, obj, false );
  GFileManipulation.writeConfig (fileName, XMLRootName, obj, true );
```

It is possible to differentiate the label displayed on the screen and the XML variable name using method setNameInConfigFile().

```
valR.setNameInConfigFile("nomXML");
```

It is also possible to have data structures, using beginOfElement() and endOfElement() methods:

```
  public void generic() {
    beginOfElement("typeOfStruct", "nameOfStruct");
      put(valR);
      put(valI);
      put(valS);
    endOfElement(); }

  public void read() { generic(); }
  public void write() { generic(); }
```

Note that the first argument of beginOfElement() may be an enum (implementing GItemTypeInterface) rather than a string which could be more robust:

```
public enum StructType implements GItemTypeInterface {

   Orbit,
   Potential,
   Atmosphere,
   Maneuver,
   Attitude;
```

```
}
```

At the end, the format of the file will be something like that:

```
 <Potential name=Earth">
      <Real name="mu" unit="km^3/s^2">398600.64</Real>
      <Real name="g0" unit="m/s^2">9.80665</Real>
      <Real name="rt" unit="km">6378.139</Real>
      <Real name="ze" unit="km">120.0</Real>
      <Real name="wt" unit="deg/s">0.004178071267451</Real>
 </Potential>
```

Missing data

From 1.4 version there is the possibility to read such files even if some data are missing. Indeed, this may occur when you want to read files from a previous version of your tool when these data where not initially foreseen.

The way to do it is very simple. In fact, you have just to call for the setDefaultValue method. For example:

```
    valG0 = new GEntryReal("G0" , 9.80665);
    valG0.setDefaultValue(9.80665);
  }
```

So, in that case, we will be able to read this file even if g0 does not appear !

```
 <Potential name=Earth">
      <Real name="mu" unit="km^3/s^2">398600.64</Real>
      <Real name="rt" unit="km">6378.139</Real>
      <Real name="ze" unit="km">120.0</Real>
      <Real name="wt" unit="deg/s">0.004178071267451</Real>
 </Potential>
```

Multiple search

First, let us consider such a data structure as shown in this XML file :

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
  <Vehicle name="vehicle">
    <!--Dry mass:-->
    <Real name="dryMass" unit="kg">1.0E0</Real>
    <!--Ergol mass:-->
    <Real name="ergMass" unit="kg">2.0E0</Real>
    <!--Reference surface:-->
    <Real name="sref" unit="m^2">3.0E0</Real>
```

```
  </Vehicle>
  <SubData name="subdata">
          <Vehicle name="vehicle">
            <!--Dry mass:-->
            <Real name="dryMass" unit="kg">11.0E0</Real>
            <!--Ergol mass:-->
            <Real name="ergMass" unit="kg">22.0E0</Real>
            <!--Reference surface:-->
            <Real name="sref" unit="m^2">33.0E0</Real>
          </Vehicle>
  </SubData>
</data>
```

We see that the **Vehicle** structure, including three real data (**drymass**, **ergmass** and **sref**) is existing in both places in the file. The code to create such an object is the following one:

```
public class VehicleData extends GPanel implements GReadWrite {

    public static final GUnit[] UNITS_MASS = {new GMetricUnit("kg"), new
GMetricUnit("t")};
    public static final GUnit[] UNITS_SURFACE = {new GMetricUnit("m^2")};

    private final GEntryReal dryMass;
    private final GEntryReal ergMass;
    private final GEntryReal sref;

    public VehicleData () {

        super();

        dryMass = new GEntryReal("Dry mass:", 0., UNITS_MASS);
        dryMass.setNameInConfigFile("dryMass");

        ergMass = new GEntryReal("Ergol mass:", 0., UNITS_MASS);
        ergMass.setNameInConfigFile("ergMass");

        sref = new GEntryReal("Reference surface:", 0., UNITS_SURFACE);
        sref.setNameInConfigFile("sref");

        // Do not forget this line ...
        this.setNameAndCommentInConfigFile("vehicle");

    }

    @Override
    public void generic() throws GException {
        beginOfElement("Vehicle", "vehicle");
        put(dryMass);
        put(ergMass);
        put(sref);
```

```
        endOfElement();
    }

    @Override
    public void display() throws GException { generic(); }
    @Override
    public void read() throws GException { generic(); }
    @Override
    public void write() throws GException { generic(); }

}
```

The idea is to know how many **Vehicle** (or how many **dryMass**, ...) we may have in the file. To do it, we have several possibilities ...

Using searchCompatibleComponents() method

With this solution, we have to call for the searchCompatibleComponents() method like below:

```
final List<GComponent> list =
GFileManipulation.searchCompatibleComponents(fileName, new VehicleData(),
true);
```

... where:

- the first argument, "fileName" is the name of the file (*note that we may also use a **GENIUS** GComponent object rather than a file where searching elements*)
- the second argument is an object of the same type to search. Note that it is mandatory that this object owns a constructor without arguments (to be able to duplicate it).
- the last argument is a boolean to indicate if the filter will consider the name of the object (if **true**) and not only itstype. Note that is the reason why, we have added the following line in order to initialize (when the constructor is called) correctly this name ...

```
this.setNameAndCommentInConfigFile("vehicle");
```

Calling this method will return a list of GComponent that could be treated for example like this:

```
System.out.println("Amount of compatible components: " + list.size());
for ( GComponent comp : list ) {
    System.out.println(comp.getGClassName());
    System.out.println(comp.getPathInConfigFile());
}
```

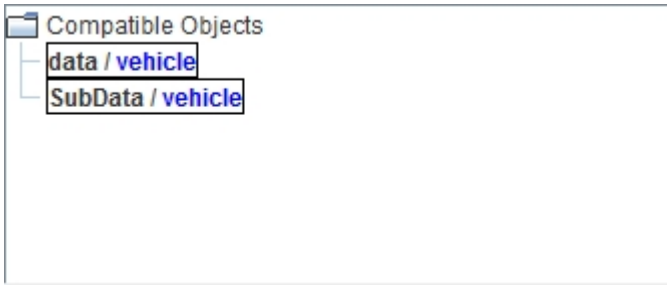
Using the GMultipleStructureChoice widget

With this solution, we have to create such a widget as is:

```
final GMultipleStructureChoice msc = new GMultipleStructureChoice(fileName,
```

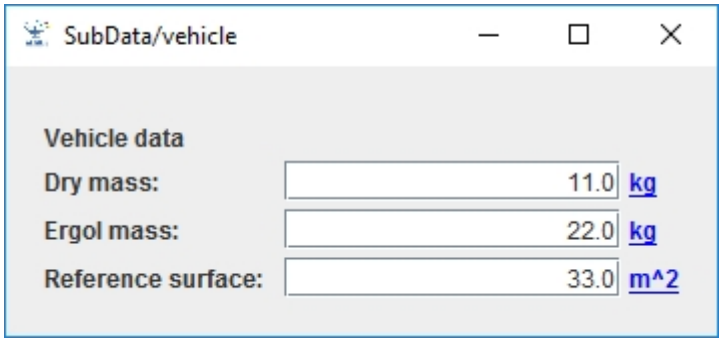
```
new VehicleData(), true);
```

The widget will be displayed like that:



After selecting one of the found objects, user will have the possibility to copy it (in the clipboard) or to display it ...

Display option will open a pop up window displaynig the object as it has been defined internally:



TBW ...

Return to the introduction ↑ Go to the next page →

Menu de navigation

Outils personnels

- 18.216.123.120
- 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 21 août 2019 à 12:03.

- Politique de confidentialité
- À propos de Wiki
- Avertissements
-