

GridBean Developer's Guide

Sandra Bergmann, May 2009



1 Introduction

A *GridBean* is one of the main concepts of the GPE (Grid Programming Environment).

A GridBean is an object responsible for:

- generating job description for grid services;
- providing graphical user interface for input data;
- providing graphical user interface for output data;
- providing interface for user's interaction with grid services.

GridBeans are divided into several modules:

- one job description generation module;
- one or more user interface modules.

GridBeans are developed using the GridBean SDK and deployed at GridBean Services. GPE clients contact GridBean Services for available GridBeans and download the selected ones. Workflows may comprise many job descriptions created by different GridBeans.

2 Developing GridBeans

2.1 GPEJob

One of the main data structures used in GridBeans is the abstraction of Jobs.

The `JSDLJob` is a Java interface for creating a JSDL document. As this document is generic the only things it can specify are:

- resource requirements;
- files to stage-in/stage-out;
- job name.

The `GPEJob` is a Java interface for a GPE atomic job description. It is an extension of `JSDLJob`.

The following additional attributes may be specified in one `GPEJob`:

- application name and version;
- named application parameters;
- application working directory;
- names of stdout and stderr files.

The application name and version uniquely select the application to be run on the target system. Named application parameters are substituted into the selected application.

2.2 Typical GridBean structure

A typical GridBean consists of 2 modules:

- one job description generation module;
- one user interface module.

The job description generation module (*GridBean Model*) extends `AbstractGridBean` which inherits 2 interfaces and thus provides the following functions:

- storing named GridBean parameters (from the interface `IGridBeanModel`);
- generating job description using these stored parameters (from the interface `IGridBean`).

There may be several user interface modules. They inherit from `IGridBeanPlugin` and provide the following:

- the set of input panels (from the interface `IGridBeanPanel`);
- the set of output panels (from the interface `IGridBeanPanel`);
- methods for loading data from and storing data to the GridBean Model;
- validating input data.

2.3 Create a GridBean Model

Each GridBean has an underlying model object. You need this model object for:

- storing data
- and creating the actual job (usually in JSDL), that may be submitted to a target system or will be used within a workflow definition

This object defines the logic of the GridBean and is independent of any graphical user interface that the GridBean provides. In order to implement the GridBean Model, create a new class that inherits from the class `com.intel.gpe.gridbeans.AbstractGridBean`.

The GridBean Model can be used similarly to a Hashmap. Use its `set` method for storing arbitrary data in it. Use the `get` method for retrieving stored data. The keys for storing and retrieving data should always be globally unique.

The client application requests the GridBean to generate a job description by calling `IGridBean.setupJobDefinition(Job)`.

The typical code for generating a job description is as follows:

```
public void setupJobDefinition(Job job) throws GridBeanException {
    super.setupJobDefinition(job);
    if (job instanceof GPEJob) {
        GPEJob gpeJob = (GPEJob) job;
        gpeJob.setApplicationName("Date");
        gpeJob.setApplicationVersion("1.0");
        gpeJob.setWorkingDirectory(
            GPEConstants.JobManagement.TEMPORARY_DIR_NAME );
    }
    else { throw new GridBeanException("Invalid job type."); }
}
```

2.4 Define all input and output parameters

In the GridBean Model, all input and output parameters for the job to be created should be declared. This step refers to input and output files that the application takes/produces and environment variables that should be set during job execution. For declaring an input or output parameter, do the following:

- Choose and declare a qualified name (QName) for the parameter.
 - Environment variable: the local part of the QName is used as the variable's name
 - Input or output file: an environment variable with this name will also be created. The value of this environment variable will be assigned the name of the file in the job's working directory
- Define the parameter by creating an Object of type `com.intel.gpe.gridbeans.IGridBeanParameter`. Have a look at `com.intel.gpe.gridbeans.parameters.ParameterUtils` for convenient ways to do this. This object carries information like the parameter name, type (e.g. single file, fileset, environment variable) and whether it is an input or output parameter.
- Add the `IGridBeanParameter` object to either the input or output parameters of the GridBean-Model:

```
getInputParameters().add(param)
```

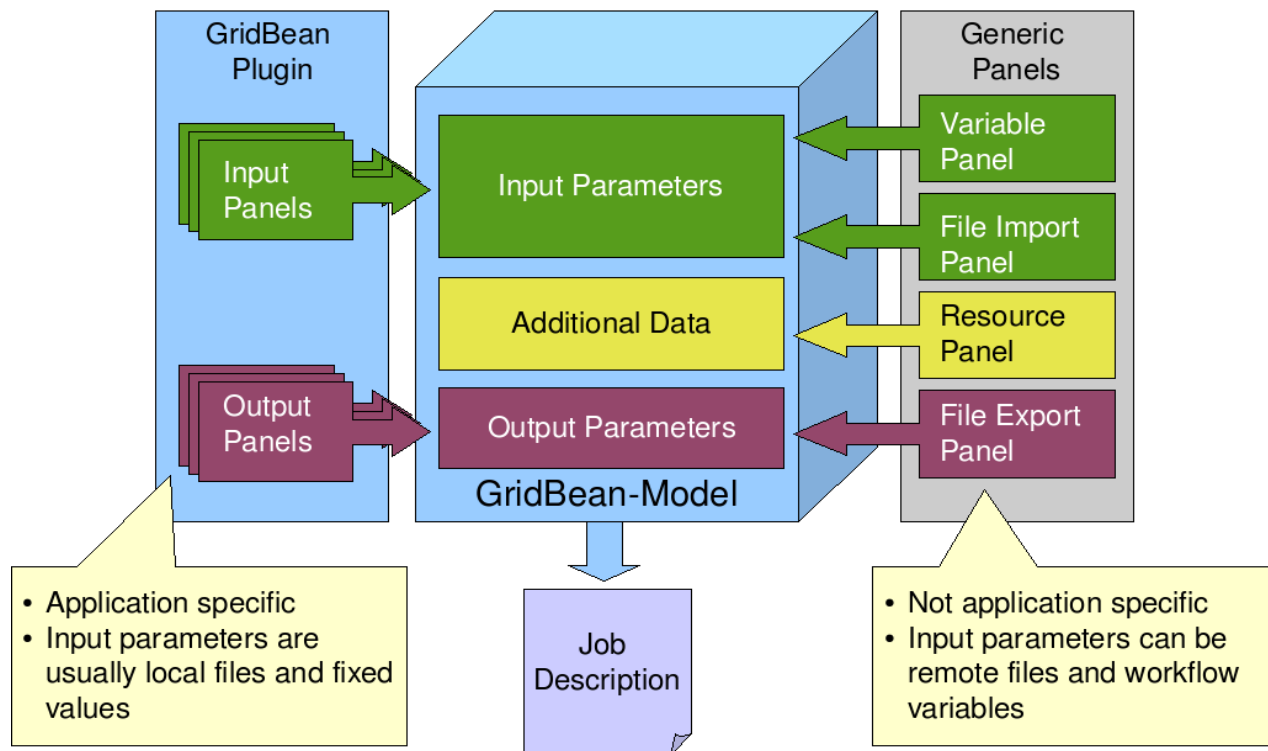
- Set the initial value of the parameter in the GridBean Model (using the chosen QName as a key and an Object of type `com.intel.gpe.gridbeans.IGridBeanParameterValue` as value).

Consider the following example from the Povray GridBean:

```
QName[] envVariables = {WIDTH,HEIGHT,ANTIALIASING_THRESHOLD, ANIMATION,  
    INITIAL_FRAME_NUMBER,FINAL_FRAME_NUMBER,SUBSET_START,  
    SUBSET_END,INITIAL_CLOCK_VALUE,FINAL_CLOCK_VALUE};  
String[] initialValues = {"320","200","0.5","false","0","0","0","0","0","0"};  
getInputParameters().addAll(ParameterUtils.createEnvParameters(envVariables));  
List<IGridBeanParameterValue> values = ParameterUtils.createEnvParameterValues(envVariables,  
    initialValues);  
  
for (int i = 0; i < initialValues.length; i++) {  
    set(envVariables[i],values.get(i));  
}
```

First, an array is created, containing all chosen QNames for the environment variables known to POVray. Then an array of initial values for these variables is provided. The `ParameterUtils` class is used for creating the environment variable parameters and the objects representing their initial values. The parameters are added to the GridBean's input parameters, their values are stored in the model.

The GridBean



Input and output parameters belong to one of the following types (see `GridBeanParameterType`):

- File
- File Set
- Environment variable
- Resources

Table 1. Input parameters

Parameter type	Description	Atomic job
File	A file at a local or remote location	The object of class <code>InputFileParameterValue</code> representing a file to be transferred to the job's working directory.
File Set	A set of files at a local or remote location	The object of class <code>InputFileSetParameterValue</code> representing a fileset (file addresses in this value may also contain wildcards)
Environment variable	A job parameter	The object of class <code>EnvironmentVariableParameterValue</code> representing the value of an environment variable or a location from where the actual value can be fetched

Table 2. Output parameters

Parameter type	Description	Atomic job
File	A file at a remote location	The object of class <code>OutputFileParameterValue</code> representing a remote file. The file is accessed via the <code>getGpeFile</code> method. Before rendering the <code>GridBean</code> 's output panels (where the output parameters are usually accessed) the files are fetched from the job's working directory.
File Set	A set of files at a remote location	The object of class <code>OutputFileSetParameterValue</code> representing a fileset. At the <code>GridBean</code> 's job design time (before the submission) the methods of the object may be used to specify which files to be included into the fileset (<code>addFile</code> , <code>removeFile</code> , <code>setFiles</code>). During the rendering time the list of fetched files is accessed via the <code>getOutputFiles</code> method of the <code>GridBean-Model</code> .

2.5 Create GridBean user interface

GridBeans may provide one or more graphical user interfaces, called "plugins", for user interaction.

Plugins are responsible for:

- the communication between a user and the GridBean and the possibility to modify the GridBean Model. (e.g. a user can set the value of an environment variable that is used as a parameter for the job definition that the GridBean produces).
- the support for additional platforms and windowing toolkits. E.g. a GridBean may also provide a portal plugin defining a GUI which can be displayed inside a web portal.

A plugin provides:

- A set of input panels;
- A set of output panels.

The following steps need to be taken in order to write a Swing based GridBean plugin

- Create one or more panels that inherit from `com.intel.gpe.gridbeans.plugins.swing.panels.GridBeanPanel`.
- Create a new class that extends `com.intel.gpe.gridbeans.plugins.swing.GridBeanPlugin`.
- Override its parent's initialize method (still call `super.initialize()`).
- Within the initialize method, add all your panels with the `addIn/OutputPanel` methods.

Each GridBean panel will have a couple of controls, usually created in its `buildComponents()` method. For linking a control to a parameter, a value translator and a value validator should be specified.

A *translator* is an object implementing `IValueTranslator` that translates the original value contained in the graphical component into the GridBean's internal value representation. E.g. the value of the text field component that is used for specification of some numerical data is of type `String` but it may be translated and stored in the GridBean model as `EnvironmentVariableParameterValue`.

A *validator* is an object implementing `IValueValidator` that validates the translated value of the graphical component and provides the error message in the case of invalid value. E.g. the validator may check that an input value is not empty.

The typical code for creating input panel components looks like this:

```
private void buildComponents() throws DataSetException {
    setLayout(new GridBagLayout());

    JTextField nameTextField = new JTextField();
    add(new JLabel("Name:"), LayoutTools.makegbc(0, 0, 1, 0, false));
    GridBagConstraints c = LayoutTools.makegbc(1, 0, 1, 0, true);
    add(nameTextField, c);
    linkJobNameTextField(POVRayGridBean.JOBNAME, nameTextField);

    JTextField widthField = new JTextField();
    add(new JLabel("Width:"), LayoutTools.makegbc(1, 1, 1, 0, false));
    add(widthField, LayoutTools.makegbc(2, 1, 1, 0, true));
    linkTextField(POVRayGridBean.WIDTH, widthField);
    setValueTranslator(POVRayGridBean.WIDTH, StringValueTranslator.getInstance());
    setValueValidator(POVRayGridBean.WIDTH, IntegerValueValidator.getInstance());

    JTextField heightField = new JTextField();
    add(new JLabel("Height:"), LayoutTools.makegbc(3, 1, 1, 0, false));
    add(heightField, LayoutTools.makegbc(4, 1, 1, 0, true));
    linkTextField(POVRayGridBean.HEIGHT, heightField);
    setValueTranslator(POVRayGridBean.HEIGHT, StringValueTranslator.getInstance());
    setValueValidator(POVRayGridBean.HEIGHT, IntegerValueValidator.getInstance());
}
```

In this example, the textfields are linked to the parameter with a special QName (e.g. `POVRayGridBean.WIDTH`, this is an environment variable parameter for telling POVRay the width of the image to be rendered).

2.6 Define a configuration file `gridbean.xml`

Each GridBean has a configuration file `gridbean.xml`. In this file you should define the name of your gridbean, your name, version, the name of the application, a description for your GridBean and the paths to your developed GridBean Model and your GridBean Plugins.

You also should declare an application name and an application version. These names should be the same as the application name and version which you define in your GridBean Model.

In the following example you will see an example for the configuration file.

```
<gb:GridBeansInfo xmlns:gb="http://gpe.intel.com/gridbeans">
  <gb:Name>Script</gb:Name>
  <gb:Author>Valentina Huber</gb:Author>
  <gb:Version>2.0</gb:Version>
  <gb:Application>Script</gb:Application>
  <gb:Description>Script GridBean</gb:Description>
  <gb:GridBean>de.fzj.gpe.gridbeans.script.ScriptGridBean</gb:GridBean>
  <gb:Plugin type="gb:Swing">de.fzj.gpe.gridbeans.script.plugin.ScriptPlugin</gb:Plugin>
  <gb:ApplicationName>Perl</gb:ApplicationName>
  <gb:ApplicationVersion>1.0</gb:ApplicationVersion>
</gb:GridBeansInfo>
```

2.7 The GridBean Model may have multiple views

When writing a GridBean you should always bear in mind that your GUI plugin is only **one** view on the GridBean Model. There might be additional GUI elements in a client that alter the model. For instance, the Eclipse based client provides generic panels for defining file imports and exports for jobs. These panels will operate on your GridBean Model and change the values of file input and output parameters! In order to keep up with the current state of the GridBean Model, it is necessary to use its built-in property change support. After adding a property change listener to the model, you will be notified of any changes that occur.

Glossary

Atomic job. A one-step computational job. It can be submitted to a target system and executed there. The expected outcome is a set of output files including standard output and standard error. The job is specified in the form of GPE Atomic JSDL extension.

Atomic services. The services implementing the corresponding port types defined by the Unigrids project. These port types include:

- TargetSystem
- JobManagement
- StorageManagement
- FileTransfer
- TargetSystemFactory

The WSRF resources behind these services perform atomic operations like job submission, file transfer, other file operations.

Atomic Target System. The target system resource that can execute atomic jobs.

Endpoint Reference. The essential element of Web Services Addressing specification. The endpoint reference is an XML structure providing the information for unique location of a network entity (web-service, resource, etc.).

GPE Atomic JSDL extension. The JSDL extension used to specify atomic jobs. It is based syntactically on the JSDL POSIX extension but uses the environment section of the Application element to pass the named parameters of the abstract application instead of actual environment variable values.

GPE Clients API. The set of client classes for work with atomic and higher level services. Refer also the section GPE Clients API.

GridBean SDK. A set of libraries and tools provided with GPE to develop GridBeans. Visit the following links for javadoc on the GPE API:

- GPE Clients API
- GridBeans API

GridBean service. The service used to publish GridBeans on the Grid.

Higher level services. The services built on top of atomic services. They include:

- Registry
- Workflow Engine
- other services like Streaming Service, Visualization Service, etc.

Job Management Resource. The type of resources managed through the job management service - one of the atomic services. This resource allows to manage a single job submitted to a target system. The list of operations in particular includes:

- start job;
- abort job;
- destroy job.

Job Submission Description Language. The XML language that defines how to describe the submission of a single job. The description includes:

- job specification;
- request for resources;
- file stage in/stage out specification.

Refer Job Submission Description Language (JSDL) Specification for JSDL specification details.

JSDL POSIX Extension. The normative extension of JSDL. Provides a method for specifying an executable invocation in a POSIX environment. Refer Job Submission Description Language (JSDL) Specification for details on JSDL and its normative extensions.

Registry. One of the higher level services used to store and provide the information on the available Grid resources such as target systems.

Storage Management Resource. The type of resources managed through the storage management service - one of the atomic services. Such resources are the abstractions of the remote file storages and provide the file operations and operations for establishing file transfers.

Target System. The type of resources managed through the target system service - one of the atomic services. The target system has only one operation: submit a job. The list of available properties includes:

- the list of submitted jobs;
- system performance characteristics;
- available storage resources.

Web Services Resource Framework. The set of standards and guidelines defining an open framework for modeling and accessing stateful resources using Web services. Visit WSRF TC at OASIS homepage for more information.

WS-NotificationProducer. The port type defined in Web Services Base Notification specification. This port type provides operation to subscribe for notifications from some entity.

WS-ResourceLifetime. The set of additional requirements for the WSRF resource in order to enable its lifetime management. This includes the definition of 2 port types:

- ImmediateResourceTermination
- ScheduledResourceTermination

Refer Web Services Resource Lifetime 1.2 (WS-ResourceLifetime) for more information.

WS-ResourceProperties. The set of additional requirements for the WSRF resource in order to enable its properties querying and management. This includes the definition of the following port types:

- GetResourcePropertyDocument
- GetResourceProperty
- GetMultipleResourceProperties
- PutResourcePropertyDocument
- SetResourceProperties
- InsertResourceProperties
- UpdateResourceProperties
- DeleteResourceProperties
- QueryResourceProperties

Not all the listed operations are available through the GPE Clients API. For more information on WS-Resource Properties refer to Web Services Resource Properties 1.2 (WS-ResourceProperties).

WSRF resource. A statefull entity that may be managed through a web-service interface following the guidelines of the WSRF standard. Refer to Web Services Resource 1.2 (WS-Resource) for more information.

XPath. The language to query the data from XML documents. GPE uses XPath of version 1.0. Refer to XML Path Language (XPath) for more details.