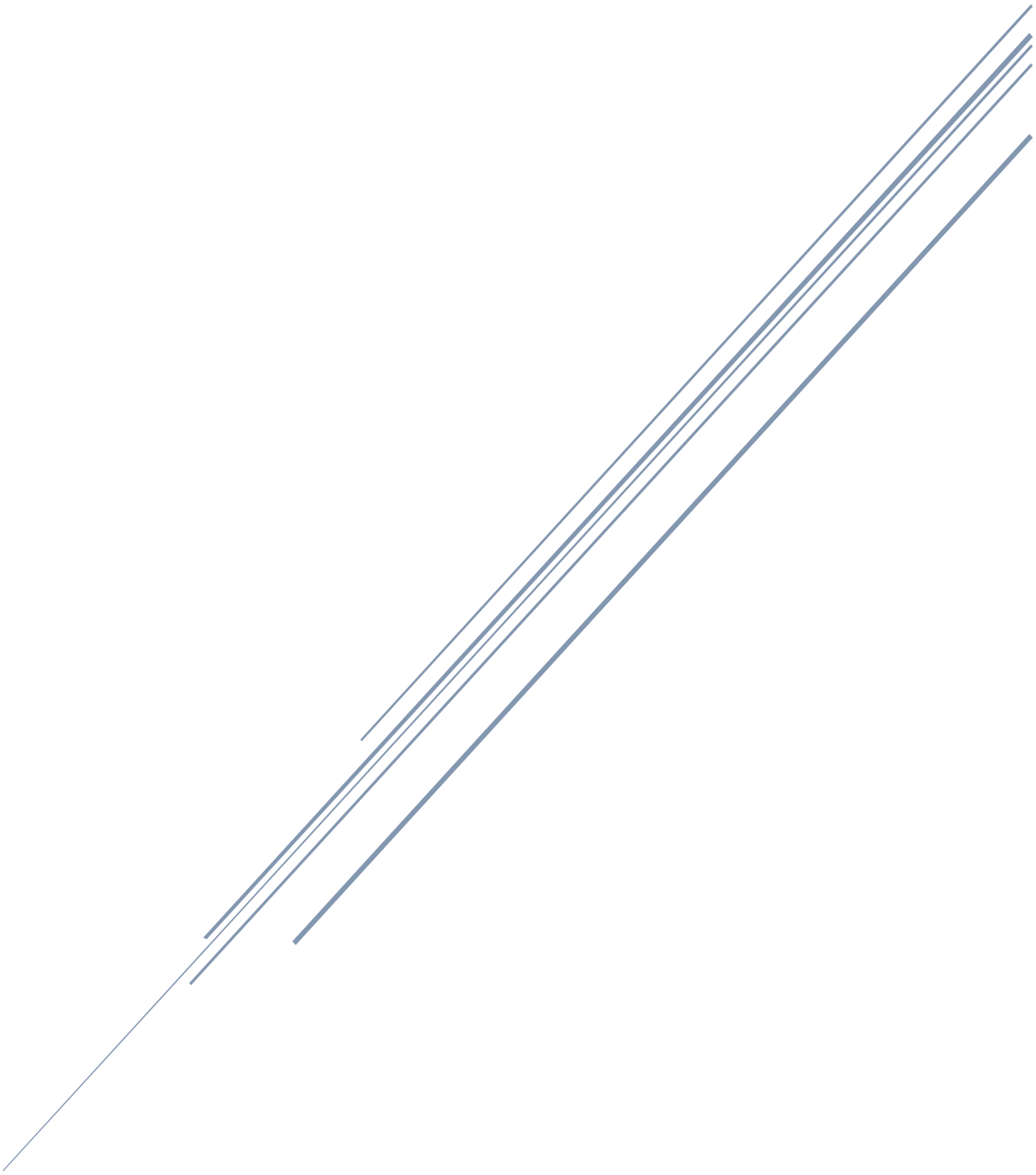
DEEPEN

atom-to-Device Explicit simulation Environment for Photonics and Electronics Nanostructures



GRIDBEAN HOW TO

TABLE OF CONTENTS

GRIDBEAN IN 10 MINUTES 2
UNICORE App Model 3
CREATE UNICORE GRIDBEAN MODEL 4
ADD A GUI TO GRIDBEAN 4
ADDING UNICORE APP PARAMETERS 5
INSTALL APP 6
SERVER CONFIGURATION 6
BIBLIOGRAPHY 7

GRIDBEAN IN 10 MINUTES

UNICORE applications are developed using Maven (https://maven.apache.org/). We recommend doing the same, because it will make the resolution of dependent libraries much easier. But it can be used any build automation system for java like ant, gradle, etc.

In order to start developing, you will have to setup a Maven project. A short introduction that should get you started with Maven can be found at (https://maven.apache.org/run.html). We will guide you through the essential steps here.

```
mvn archetype:create -DgroupId=eu.unicore.app.deepen.omen -DartifactId=DEEPEN-omen
```

That will generate an almost empty project with the right structure and a `pom.xml` file representing your project object model (POM), under the root of the project.

Then there is a `src` directory that splits up in two hierarchies, `main` and `test`. The `main` part will be included in the generated artifact, the `test` part contains testing code that will be executed at build time to ensure your classes behave as expected. Let's focus on the essential part, being the main sources. Remove the `App.java` and `AppTest.java` files from your project, they are just examples.

The UNICORE App will depend on specific UNICORE modules that are configured in the POM. Therefore, it is also required to add these and the UNICORE Maven repository to the POM. You can achieve this by adding the following to the pom.xml of your project.

```
<project ...>
  ...
  <dependencies>
    ...
    <dependency>
      <groupId>com.intel.gpe.clients</groupId>
      <artifactId>gpe-gridbeans-swt</artifactId>
      <version>2.2.5</version>
    </dependency>
    <dependency>
       <groupId>eu.deepen.cdf</groupId>
        <artifactId>cdf</artifactId>
        <version>1.0</version>
        <scope>system</scope>
        <systemPath>
          ${project.basedir}/src/main/resources/lib/cdf.jar
        </systemPath>
     </dependency>
    ...
  </dependencies>
  ...
  <repositories>
    <repository>
      <id>unicore-dev</id>
      <url>https://unicore-dev.zam.kfa-juelich.de/maven</url>
    </repository>
  </repositories>
  ...
</project>
```

UNICORE Apps are packaged as JAR archives, which Maven can generate for you. In order to provide a fully functional UNICORE App, one additional file has to be packaged inside the JAR file. This file only needs to be placed in a special location of your sources and it will be packaged along with the JAR. Under your project root, create a file at

```
src/main/resources/META-INF/gridbean.xml
```

and put the following inside:

```
<gb:GridBeansInfo xmlns:gb="http://gpe.intel.com/gridbeans">
   <gb:Name>OMEN</gb:Name>
   <gb:Author>tiberlab</gb:Author>
   <gb:Version>1.0</gb:Version>
   <gb:Application>OMEN</gb:Application>
   <gb:Description>Quantum transport solver</gb:Description>
   <gb:GridBean>eu.unicore.app.deepen.omen.OmenGridBean</gb:GridBean>
   <gb:Plugin type="gb:Swing">eu.unicore.app.deepen.omen.plugin.OmenPlugin</gb:Plugin>
   <gb:ApplicationName>OMEN</gb:ApplicationName>
   <gb:ApplicationVersion>1.0</gb:ApplicationVersion>
</gb:GridBeansInfo>
```

This is the UNICORE App configuration file. The meaning of the elements is as follows.

`Name`: The name of this UNICORE App.

`Author`: The author of the UNICORE App. Usually, that would be your name.

`Version`: The version of this UNICORE App. It is shown along with the name in the applications palette of the workflow editor. It is also shown in the list of applications when creating a new job on a site.

`Application`: The name of this UNICORE App. Usually the same as the plain Name element.

`Description`: A description of this UNICORE App.

`GridBean`: The fully-qualified name of the class in your UNICORE App that extends the AbstractGridBean class.

`Plugin`: The fully-qualified name of the class in your UNICORE App that implements the `IGridBeanPlugin<Component>` interface. The `type` attribute must be either "`gb:Swing`" or "`gb:SWT`". There are abstract base classes available for each of these toolkits. These are `com.intel.gpe.gridbeans.plugins.swing.GridBeanPlugin` and `com.intel.gpe.gridbeans.plugins.swt.SWTGridBeanPlugin` respectively.

`ApplicationName`: The name of your application as configured in the server. This will be used for matching computational resources that provide the application. Thus, `gb:ApplicationName` in gridbean.xml must match `idb:ApplicationName` in the server side IDB.

`ApplicationVersion`: The version of your application as configured in the server. This will be used for matching computational resources that provide the application. If this App works with any version of the server side application, the `ANY_VERSION` keyword can be used. `gb:ApplicationVersion` in gridbean.xml must match `idb:ApplicationVersion` in the server side IDB unless any version is sufficient.

Now you have setup the basic project for your UNICORE App. The following sections will describe how to create the UNICORE App Model and a corresponding GUIPlugin.

UNICORE App Model

The center of the UNICORE App is the UNICORE App Model that provides a generic way to store and retrieve information about a particular job configuration. The parameters to be stored in a UNICORE App can be categorized in input parameters, output parameters, and additional data. Most of this data will be used to create a job description for submission to a UNICORE target system.

The generic panels are always present in any UNICORE App and they contain generic user interface panels to edit and view parameters of the model.

The UNICORE App Plugin is the application specific GUI portion of the UNICORE App. Beside defining application specific parameters and adding them to the UNICORE App Model, a UNICORE App developer would typically create a set of application specific input and output panels and add them to the UNICORE App. GUI elements within these panels will then be linked with the model elements, such that the user settings will be stored in the UNICORE App Model automatically.

CREATE UNICORE GRIDBEAN MODEL

Model object defines the logic of the Grid Bean and is independent of any graphical user interfaces 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 `QNames`.

ADD A GUI TO GRIDBEAN

UNICORE Apps may provide one or more graphical user interfaces, called **plugins**, for user interaction. Plugins are responsible for the communication between an user and the UNICORE App and the possibility to modify the UNICORE App Model

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 UNICORE App 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, but still call `super.initialize()`
- Within the initialize method, add all your panels with the `addInputPanel()` or `addOutputPanel()` methods

Each UNICORE App panel will have a number of controls, usually created in its `buildComponents()` method. For linking a control to a parameter, at least a `ValueTranslator` must be specified. Additionaly, you can add a `ValueValidator` to ensure the correct format of the input.

VALUE TRANSLATORS

A *translator* is an object implementing `IValueTranslator` that translates the original value contained in the graphical component into the UNICORE App's internal value representation.
Translators are defined in package
`com.intel.gpe.gridbeans.plugins.translators`

VALUE VALIDATORS

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.
Validators are defined in package
`com.intel.gpe.gridbeans.plugins.validators`

ADDING UNICORE APP PARAMETERS

In the UNICORE App Model, all input and output parameters for the job to be created must be declared. This step refers to input and output files that the application takes/produces, as well as environment variables that will 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
  `QName (String namespaceURI, String localPart, String prefix)`
  - Environment variable: the local part of the QName is used as the variable's name. Therefore, you need to make sure that even the local part of the name is unique within the application.
  - 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 UNICORE App Model with
  `getInputParameters().add(param)` or `getOutputParameters().add(param)`

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

TYPES OF PARAMETERS

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

- **File**
  A file at a local or remote location. The file is represented by an object of class `InputFileParameterValue`. As an input parameter, it is a file to be transferred to the job's working directory.

- **File Set**
  A set of files at a local or remote location. The file set is represented by an object of class `InputFileSetParameterValue`. File sets are collections of files. Their addresses may also contain wildcards.

- **Environment Variable**
  A job parameter, represented by an object of class EnvironmentVariableParameterValue. The value is either fixed of can be taken from workflow variables whenever the job is embedded within a UNICORE workflow.

- **Resources**
  Resource requirements of the job.

Output parameters can be the following:

- **File**
  A file at a remote location represented by an object of class `OutputFileParameterValue`. The file is accessed via the `getGpeFile()` method. Before rendering the UNICORE App'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 represented by an object of class `OutputFileSetParameterValue`. At the UNICORE App's job design time (before the submission) the methods of the object may be used to specify which files to include into the fileset (`addFile()`, `removeFile()`, `setFiles()`). During the rendering time the list of fetched files is accessed via the `getOutputFiles()` method of the UNICORE App Model.

INSTALL APP

Now you have setup the basic project for your UNICORE App and you can export it into jar file using Maven with:

```
mvn package
```

UNICORE Apps can directly be installed in the UNICORE Rich Client (URC) by putting their JAR file into the applications folder of the URC installation directory. The default applications folder is located in the installation directory of the URC and called `applications`.

SERVER CONFIGURATION

The UNICORE IDB (incarnation database) contains information on how abstract job definitions are to be mapped onto real executables. This process is called "incarnation".

The second IDB function is advertising target system capabilities and allowing to check client resource requests against these.

The IDB is (in a simple configuration) a XML files, which by default is called simpleidb (by default …/unicorex/conf/simpleidb).

The most important functionality of the IDB is providing executables for abstract applications. An abstract application is given by name and version, whereas an executable application is given in terms of executable, arguments and environment variables.

So, for each GridBean App that run inside the Client, we need a server side executable. This executable can be written with any language, but if you want to use DEEPEN CDF format taking advantage of the API developed during the project, you need to develop such applications in Java.

Here is an example entry for *QuantumEspresso* tool:

```
<idb:IDBApplication>
  <idb:ApplicationName>Quantum-Espresso</idb:ApplicationName>
  <idb:ApplicationVersion>1.0</idb:ApplicationVersion>
  <jsdl:POSIXApplication xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl-
posix">
    <jsdl:Executable>/usr/bin/qe.sh</jsdl:Executable>
    <jsdl:Argument>-n $NUMOFCPUS</jsdl:Argument>
    <jsdl:Argument>$EXECUTABLE?</jsdl:Argument>
    <jsdl:Argument>$INPUTFILENAME?</jsdl:Argument>
     <jsdl:Argument>$OUTPUTFILENAME?</jsdl:Argument>
  </jsdl:POSIXApplication>
 </idb:IDBApplication>
```

Conditional arguments are indicated by “?” char appended to the argument value.
The job submission from a client usually contains environment variables to be set when running the application. It often happens that a certain argument should only be included if a corresponding environment variable is set. This can be achieved by using “conditional arguments” in the incarnation definition.

BIBLIOGRAPHY

“Maven Getting Started Guide” - http://maven.apache.org/guides/getting-started/index.html

“GridBean Development” - https://sourceforge.net/p/unicore/wiki/GridBean_Development/

“UNICOREX Manual” - https://www.unicore.eu/docstore/unicorex-7.7.0/unicorex-manual.html