Exercise 5 - BEAM Modules

Learning Targets

- Introduction to BEAM Modules
- Create a simple module
- Use the Module Manager to investigate the module
- Add an action to the module
- Use the module in VISAT

Introduction to BEAM Modules

The module.xml is the file which will describe your module. It contains information like the name, version, description and dependencies and also some advanced stuff like extensions and extension points.

<table>
<thead>
<tr>
<th>module.xml (Minimal content)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;module&gt;</code></td>
</tr>
<tr>
<td><code>&lt;manifestVersion&gt;1.0.0&lt;/manifestVersion&gt;</code></td>
</tr>
<tr>
<td><code>&lt;symbolicName&gt;minimal-module&lt;/symbolicName&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/module&gt;</code></td>
</tr>
</tbody>
</table>

The tag `manifestVersion` defines the version of the module.xml file.
The `symbolicName` is used as a unique identifier in the runtime environment.

Create a simple module

Add VISAT to BEAM library

The VISAT API is a selfstanding BEAM module which needs to be added to the classpath.
This is similar to exercise 3 but this time we add `beam-ui` and `beam-visat-rcp` to our library.

Create a module.xml

Create a file named `module.xml` in the source folder of your project.
As a first step you should only add the `symbolicName`, the `version` and the `name` of your module and also a short `description`.

<table>
<thead>
<tr>
<th>module.xml</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;module&gt;</code></td>
</tr>
<tr>
<td><code>&lt;manifestVersion&gt;1.0.0&lt;/manifestVersion&gt;</code></td>
</tr>
<tr>
<td><code>&lt;symbolicName&gt;aatsr-cloud-mask&lt;/symbolicName&gt;</code></td>
</tr>
<tr>
<td><code>&lt;version&gt;1.0&lt;/version&gt;</code></td>
</tr>
<tr>
<td><code>&lt;name&gt;AATSR-TOA Cloud Mask&lt;/name&gt;</code></td>
</tr>
<tr>
<td><code>&lt;description&gt;Creates an AATSR TOA cloud mask&lt;/description&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/module&gt;</code></td>
</tr>
</tbody>
</table>

Now create a JAR file as described in previous exercise: Cloud Masking Tool - Create a JAR

Use the Module Manager to investigate the module

Copy the JAR you have just created into the `BEAM4_HOME/modules` directory and start VISAT.
Open the Module Manager which you can find in the main menu. Select Help/Module Manager....
You can see the name on the left of the table, the version number and the state of the module. The column on the right hand side of the table shows the action which will be performed on the corresponding module. The action can be one of Install, Uninstall and Update. Below the table you'll see also the description of the module you've typed in.

But for now the module can't do anything useful. Close VISAT and return to the IDE.

**Add an action to the module**

**Create an action class**

To use the operator from within VISAT you have to define an action, which will be invoked when the user selects it from the menu. In order to create a menu item, which can invoke your operator you have to implement a little action. Create a new class and name it `CloudMaskOpAction`. This class must extend from the `AbstractVisatAction` class.

```java
import org.esa.beam.visat.actions.AbstractVisatAction;

public class CloudMaskOpAction extends AbstractVisatAction {
}
```

**Implement the action with User Interface**

In order to let the action do something useful, you have to override the `actionPerformed(CommandEvent event)` method. This method will be called each time the user has selected the menu item.
The `DefaultSingleTargetProductDialog` generates automatically a default dialog for operators based on the annotations of the operator class. The dialog contains two tabs. The "I/O Parameters" tab allows changing the input product and the name of the output product. The "Processing Parameters" tab allows editing of the parameters of the operator.

### Implement the action without User Interface

Instead of using the `DefaultSingleTargetProductDialog` it is sometimes required to implement a custom `actionPerformed(CommandEvent event)` method.

```java
private ModelessDialog dialog;

@Override
public void actionPerformed(final CommandEvent event) {
    if (dialog == null) {
        DefaultSingleTargetProductDialog dstpDialog =
            new DefaultSingleTargetProductDialog("MaskClouds",
            getAppContext(),
            "AATSR TOA Cloud Mask Computation",
            null);
        dstpDialog.setTargetProductNameSuffix("_cloudmask");
        dialog = dstpDialog;
    }
    dialog.show();
}
```

First the currently selected product is retrieved from VISAT. The source product is put by its name into a map. The threshold parameter is put into a separate map. To call the operator we use the alias name `MaskClouds` and pass it along with the two maps into the `GPF.createProduct(String operatorName, Map<String, Object> parameters, Map<String, Product> sourceProducts)` method. After the target product is created, it is made known to VISAT by adding it to the product manager.

### Change the state of the action

If you choose to implement a custom `actionPerformed(CommandEvent event)` method, you should also override the `updateState(CommandEvent event)` method. This method is called every time the state of the user interface might change and has to be updated. In this method you should at least check if a product is selected and if it is an AATSR_TOA product and enable the menu item, if so. Otherwise you can disable it, so the user can't invoke the action.
CloudMaskOpAction.class (Implementation of updateState)

```java
@override
public void updateState(CommandEvent event) {
    final VisatApp visatApp = VisatApp.getApp();
    final Product selectedProduct = visatApp.getSelectedProduct();
    if(selectedProduct != null && EnvisatConstants.AATSR_L1B_TOA_PRODUCT_TYPE_NAME.equals(selectedProduct.getProductType())) {
        setEnabled(true);
    } else {
        setEnabled(false);
    }
}
```

Define an extension

In order to make your action known to VISAT, you have to add an extension to the `module.xml`.

```xml
<module>
...
  <extension point="beam-ui:actions">
    <action>
      <id>cloudMaskOp</id>
      <class>CloudMaskOpAction</class>
      <text>Cloud Mask Operator...</text>
      <shortDescr>Creates an AATSR TOA cloud mask</shortDescr>
      <parent>tools</parent>
    </action>
  </extension>
...
</module>
```

The `<extension>` tag defines the point where the extension should plug in. In this case it is `beam-ui:actions`. Set an Id to the action and specify the implementing class. Also set a short text for the menu item and a short description which is used as a tool tip. Finally you should specify the parent menu where the action shall be placed.

Add dependencies to the module

Additionally you have to define the dependencies the module has to other modules. These are the `beam-gpf` module and the `beam-envisat-reader` module and an optional dependency on the `beam-visat` module. An optional dependency is useful if the module is also usable without this dependency.

The `cloud_mask` module can be used in batch mode where the `beam-visat` module is not required. All the other dependencies, you've previously added to the classpath, are not necessary, when using a module. The dependency to `beam-core`, for example, is implicitly defined by the `beam-gpf` module and therefor already on the classpath.

```xml
<module>
...
  <dependency>
    <module>beam-gpf</module>
  </dependency>
  <dependency>
    <module>beam-envisat-reader</module>
  </dependency>
  <dependency>
    <optional>true</optional>
    <module>beam-visat</module>
  </dependency>
...
</module>
```
Now build the JAR again and drop it in the `%BEAM4_HOME%/modules` directory. Overwrite the previously module.

**Use the module in VISAT**

When you now start VISAT, and try directly to invoke the operator from the **Tools** menu, you'll see that the menu item is disabled.

Open an AATSR TOA product, and try again to invoke the operator. Now the item is enabled and you can select it. The created product will be shown in the product tree of VISAT and you can examine the result.

**More information**

For more information on BEAM modules please refer to

1. Extending BEAM
2. Module Deployment