Norman Fomferra
Thomas Storm
Ralf Quast

BEAM PROGRAMMING TUTORIAL

How to use BEAM in your own programs and how to extend the platform

Brockmann Consult

esa

Sentinel-3 OLCI/SLSTR & MERIS/(A)ATSR
Workshop, ESRIN, Oct 2012

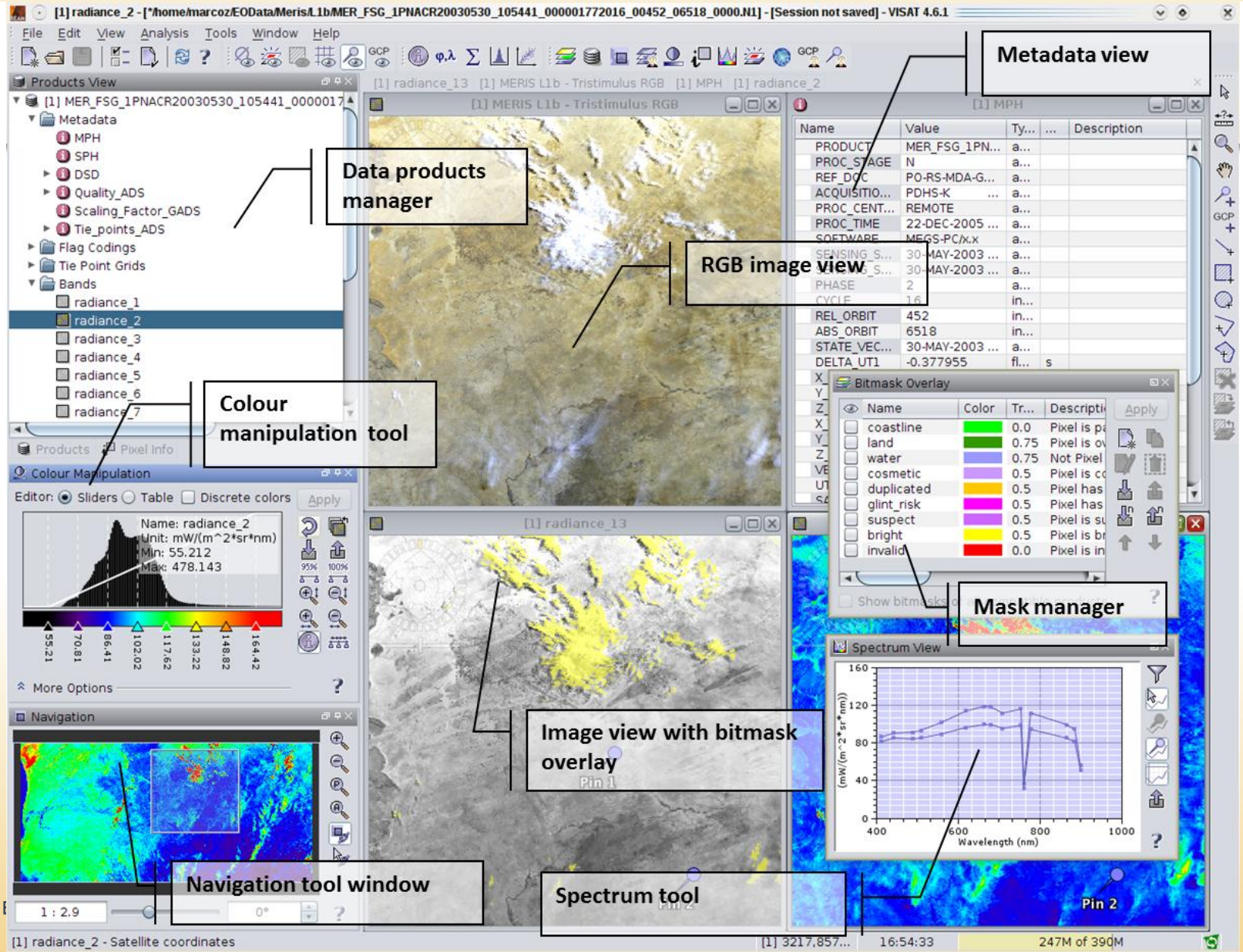
Overview

- Introduction to the BEAM Software (15 min)

- **Exercise 1**:
  Generate a quicklook from MERIS L1b (15 min)
- **Exercise 2**:
  Compute FLH from MERIS L1b (30 min)
- **Exercise 3**:
  Integrated FLH tool extending BEAM (60 min)

BEAM Software Overview and Runtime Configuration

Graphical User Interface - VISAT



Command-Line Interface - GPT

BEAM Command Line

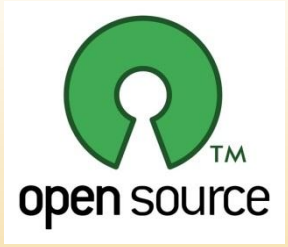
```
  -T<target>=<file>  Defines a target product. Valid for graphs only. <target>
                     must be the identifier of a node in the graph. The node's
                     output will be written to <file>.
  -S<source>=<file>  Defines a source product. <source> is specified by the
                     operator or the graph. In an XML graph, all occurrences of
                     ${<source>} will be replaced with references to a source
                     product located at <file>.
  -P<name>=<value>   Defines a processing parameter, <name> is specific for the
                     used operator or graph. In an XML graph, all occurrences of
                     ${<name>} will be replaced with <value>. Overwrites
                     parameter values specified by the '-p' option.

Operators:
  Aatsr.SST                 Computes sea surface temperature (SST) from (A)ATSR products.
  BandMaths                 Create a product with one or more bands using mathematical expressions.
  Binning                   Performs spatial and temporal aggregation of pixel values into 'bin' cells
  Collocate                 Collocates two products based on their geo-codings.
  EMClusterAnalysis         Performs an expectation-maximization (EM) cluster analysis.
  FlhMci                    Computes fluorescence line height (FLH) or maximum chlorophyll index (MCI).
  KMeansClusterAnalysis     Performs a K-Means cluster analysis.
  Merge                     Allows copying raster data from any number of source products to a specified
ct.
  Meris.Brr                 Compute the BRR of a MERIS L1b product.
  Meris.Case2Regional       Performs IOP retrieval on L1b MERIS products, including atmospheric correctio
  Meris.CorrectRadiometry   Performs radiometric corrections on MERIS L1b data products.
  Meris.GlintCorrection     MERIS atmospheric correction using a neural net.
  Meris.Lakes               Performs IOP retrieval for eutrophic and boreal Lakes on L1b MERIS products,
spheric correction.
  Meris.N1Patcher           Copies an existing N1 file and replaces the data for the radiance bands
  Mosaic                    Creates a mosaic out of a set of source products.
  PixEx                     Extracts pixels from given locations and source products.
  Read                      Reads a product from disk.
  Reproject                 Reprojection of a source product to a target Coordinate Reference System.
  Subset                    Create a spatial and/or spectral subset of a data product.
  Unmix                     Performs a linear spectral unmixing.
  Write                     Writes a data product to a file.
  glint.Flint               Flint Processor.
```

BEAM Installation Directory

- **beam-4.10.x** — *BEAM 's home directory*
  - **bin** — *Application executables*
  - **lib** — *Common, 3rd -party libraries*
  - **modules** — *Application and plug-in modules*
  - **config** — *Application configuration file(s)*
    - beam.config

Why develop with BEAM?

- **Access** EO raster data, metadata, no-data, flags, …
- Simple and effective **programming models**
- **Reuse** a rich software infrastructure
- It is **free**, it is **open source**
- It is **widely used**
- It is very **well tested**
- It is **supported by ESA, used by NASA**

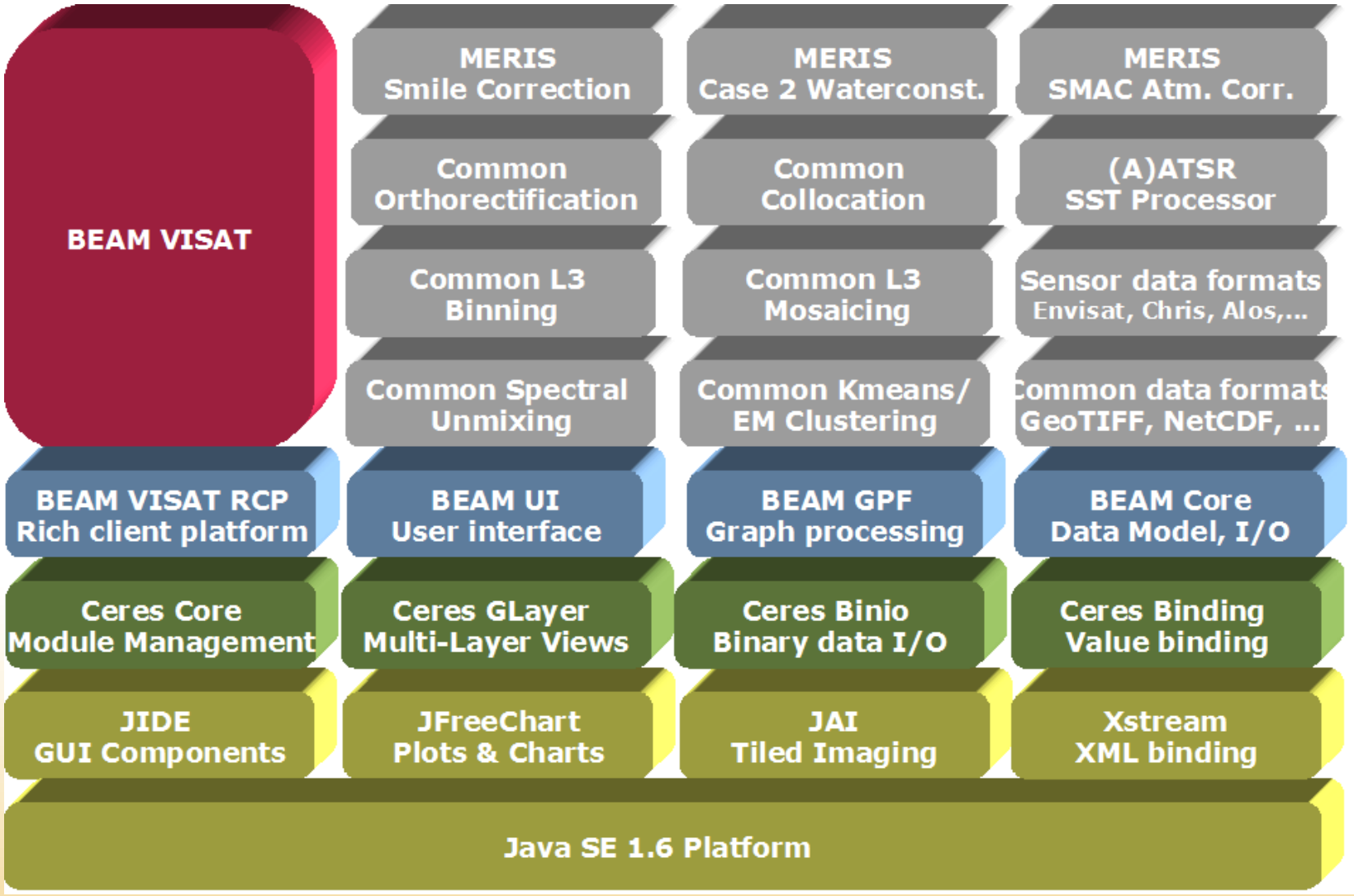


BEAM Development Use Cases

1. **Use or embed BEAM**
   - Write batch-mode scripts
   - Program stand-alone applications
   - Develop web-services
2. **Extend BEAM**
   - Add EO data readers, writers
   - Add EO data processing nodes
   - Add map projections, DEMs
   - Add VISAT actions, tool bars and tool windows
3. **Clone BEAM**
   - Customize and brand VISAT

Architecture and Application Programming Interfaces

Architecture Overview

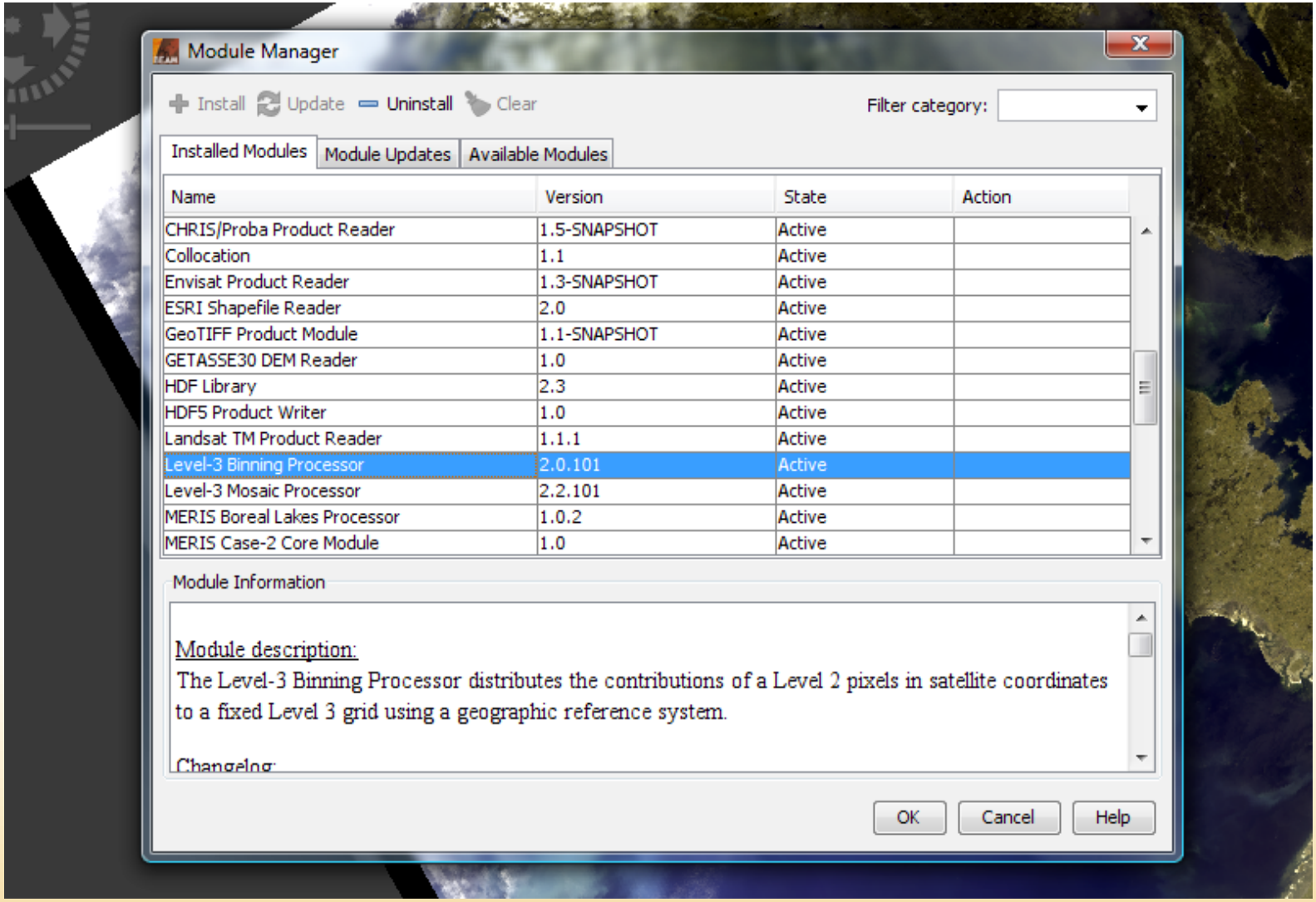


Module-based Architecture

- BEAM module:
  - Name
  - Description
  - Version
  - Authors
  - Changelog
  - Dependencies
  - Extension points
  - Extensions

- Simple extension model:
  - *Host* module provides extension point, e.g. "product-reader"
  - *Client* module provides extension to host's extension point, e.g. "seawifs-reader"
  - Clients can be hosts

VISAT Module Manager



Important Extension Points

- EO Data
  - Data processor plug-ins (GPF operators, Ex3)
  - Data product reader plug-ins
  - Data product writer plug-ins

- VISAT User Interface
  - VISAT actions
  - VISAT tool windows
  - VISAT branding

Important system-level APIs

- Module beam-core — *Commonly used classes*
  - org.esa.beam.framework.datamodel — *Product Model API*
  - org.esa.beam.framework.dataio — *Data I/O API*
- Module beam-gpf — *Graph processing framework*
  - org.esa.beam.framework.gpf — *GPF API*
  - org.esa.beam.framework.gpf.ui — *GPF user interface components*
- Module beam-ui — *User interface*
  - org.esa.beam.framework.ui — *UI components*
- Module beam-visat-rcp — *VISAT Rich Client Platform*
  - org.esa.beam.visat — *VISAT application comp.*
- Module ceres-core — *Module runtime*
  - com.bc.ceres.core — *Module API, System utilities*

Useful Links for Programmers

- Homepage
  www.brockmann-consult.de/beam/
- Programming tutorial
  www.brockmann-consult.de/beam/wiki
- Software downloads
  www.brockmann-consult.de/beam/software
- API documentation
  www.brockmann-consult.de/beam/doc/apidocs
- Source repository
  github.com/bcdev/beam
- User forum
  www.brockmann-consult.de/beam/forum

!

Exercise 1:
Generating a quicklook image from MERIS L1b

Exercise 1 Contents

- Prepare a Java IDE for programming with BEAM
- Inspect a simple Java program
- Introduce some important objects
- Introduce the BEAM generic product model
- Demonstrate that you don't need object-oriented programming skills

Tutorial Requirements

- BEAM 4.10.x binaries and sources
- Java Software Development Kit
  - Java SE JDK 1.6
    www.oracle.com/technetwork/java/javase
- Java Development Environment
  - IntelliJ IDEA 11 Community
    www.jetbrains.com/idea/
  - Used for BEAM development and this tutorial, but Eclipse, NetBeans work fine as well
  - Example IDEA projects: **~/IdeaProjects/Ex1**, **Ex2**, **Ex3**

Ex1: Lead Through (1/2)

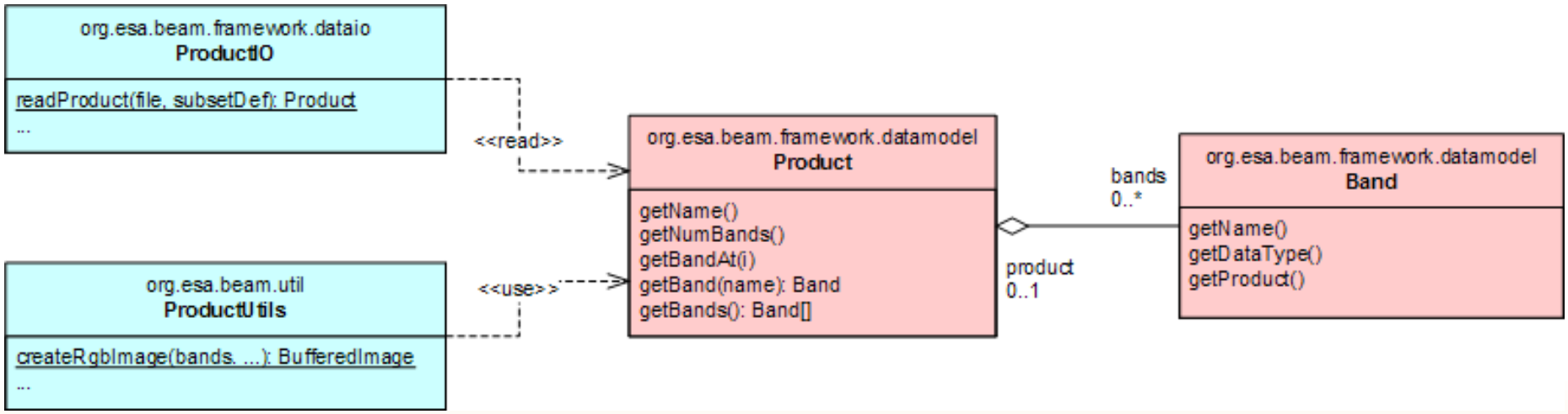
- Start IDEA
- Open project **Ex1**
- Inspect project folder
- Have a look at the Java source file in **src**
- Compile
- Produce syntax error and compile, remove error
- Run the program (right click class)
- Add command-line argument: MERIS L1b file
- Run again
- Sync project folder, open PNG quicklook image

Java Naming Conventions

- Interface / Class names
  - CamelCase, first character upper-case
  - `GeoCoding`, `Product`, `RasterDataNode`
- Variable names
  - CamelCase, first character lower-case
  - `backgroundColor`, `threshold`, `minValue`
- Constant names
  - All letters upper-case, underscore
  - `DIALOG_TITLE`, `BUFFER_SIZE_LIMIT`
- Method / Function names
  - CamelCase, first character lower-case
  - `getSampleValue`, `computeTile` , `isClosed`
- Package names
  - all lower-case
  - `beam`, `syke`, `binning`

Product Data Model (1/2)

- A "Product" object contains objects of type "Band"

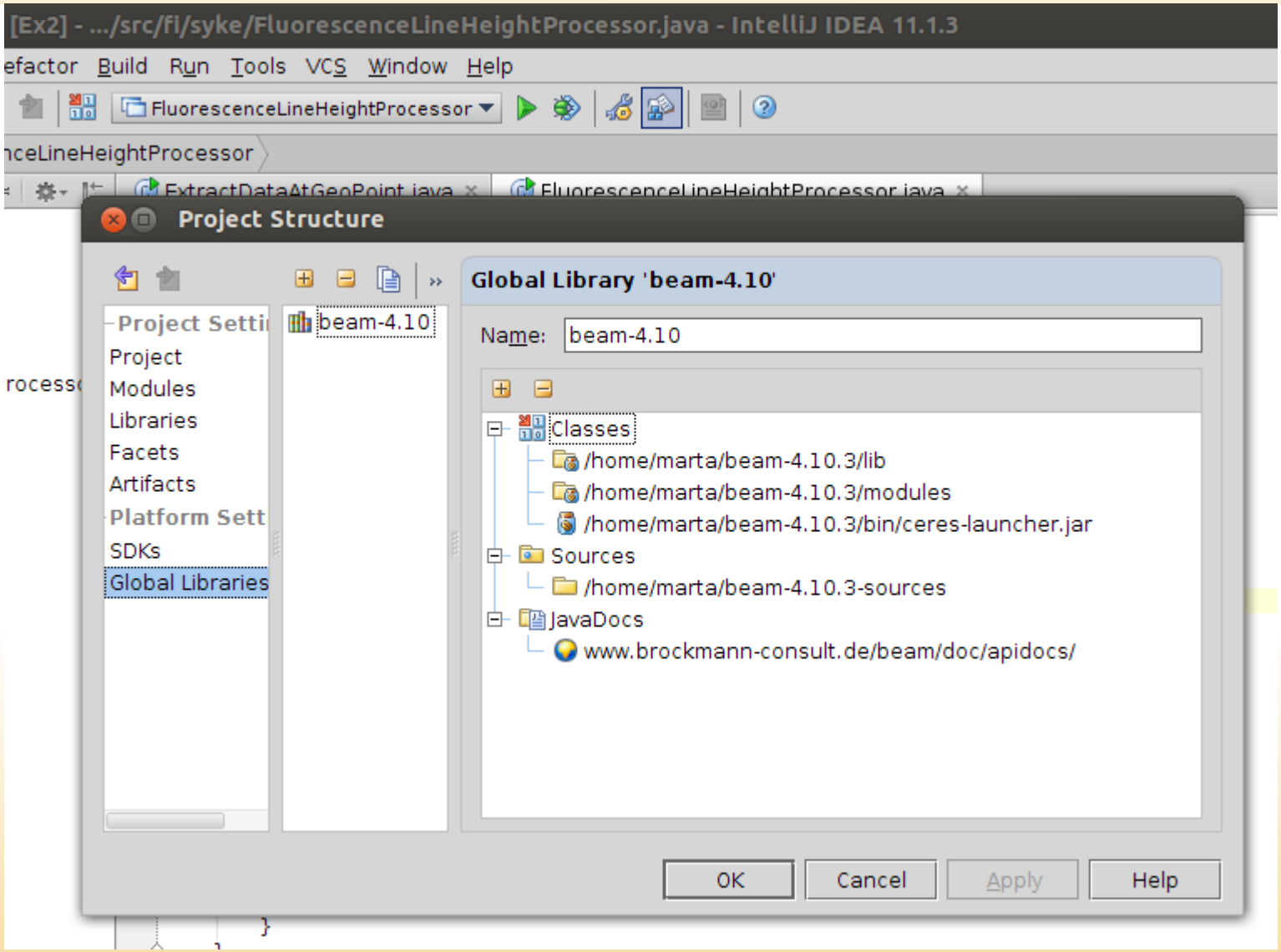


- The ProductIO class can be used to open data product files

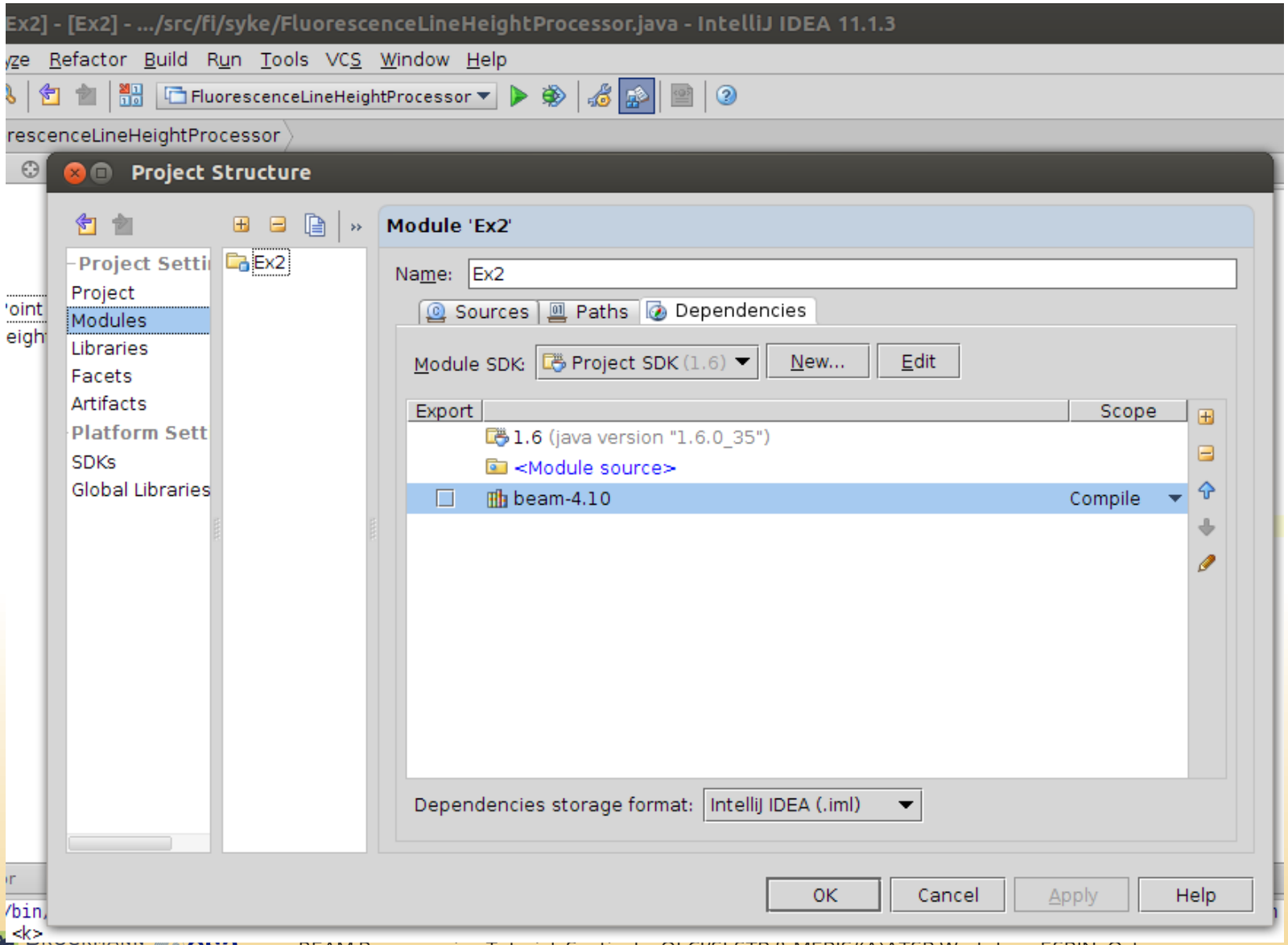
Ex1: Lead Through (2/2)

- Open project settings dialog
- Have a look at Global Library **beam-4.10.3**
  - Library (JAR) containing directories **bin**, **lib** and **modules**
- Have a look at the module dependencies and see **beam-4.10.3**, close project settings dialog
- Inspect used object instances of classes: **ProductIO**, **Product**, **Band**, **ImageIO**, **IOException**
- Change code!
  - to print out band names
  - to process multiple inputs
  - to not handle the exception

IDEA Setup: Add global BEAM Library



IDEA Setup: Add Module Dependency



Exercise 2: Computing FLH from MERIS Level 1b

Exercise 2 Contents

- Create a new product object from scratch
- Add a Fluorescence Line Height (FLH) band object to the product object
- Compute FLH pixels for the new band
- Write a data product in a specific data format
- Run VISAT from the IDE in order to inspect, validate, analyse the output
- Write a script that invokes the program

Fluorescence Line Height (FLH)

- Used as an indicator for the biological activity of the phytoplankton.
- Phytoplankton chlorophyll fluorescence at 680.5 nm is measured and its height above a baseline through the measurements at 664 nm and 708 nm is calculated
- $$FLH = L_2 - k*[L_1 + a*(L_3 - L_1)]$$
  $$a = (\lambda_2 - \lambda_1)/(\lambda_3 - \lambda_1)$$

"Interpretation of the 685 nm peak in water-leaving radiance spectra in terms of fuorescence, absorption and scattering, and its observation by MERIS" J. F. R. Gower, R. Doerffer, G. A. Borstad, Int. J. Remote Sensing, 1999, vol. 20, no. 9, 1771-1786.

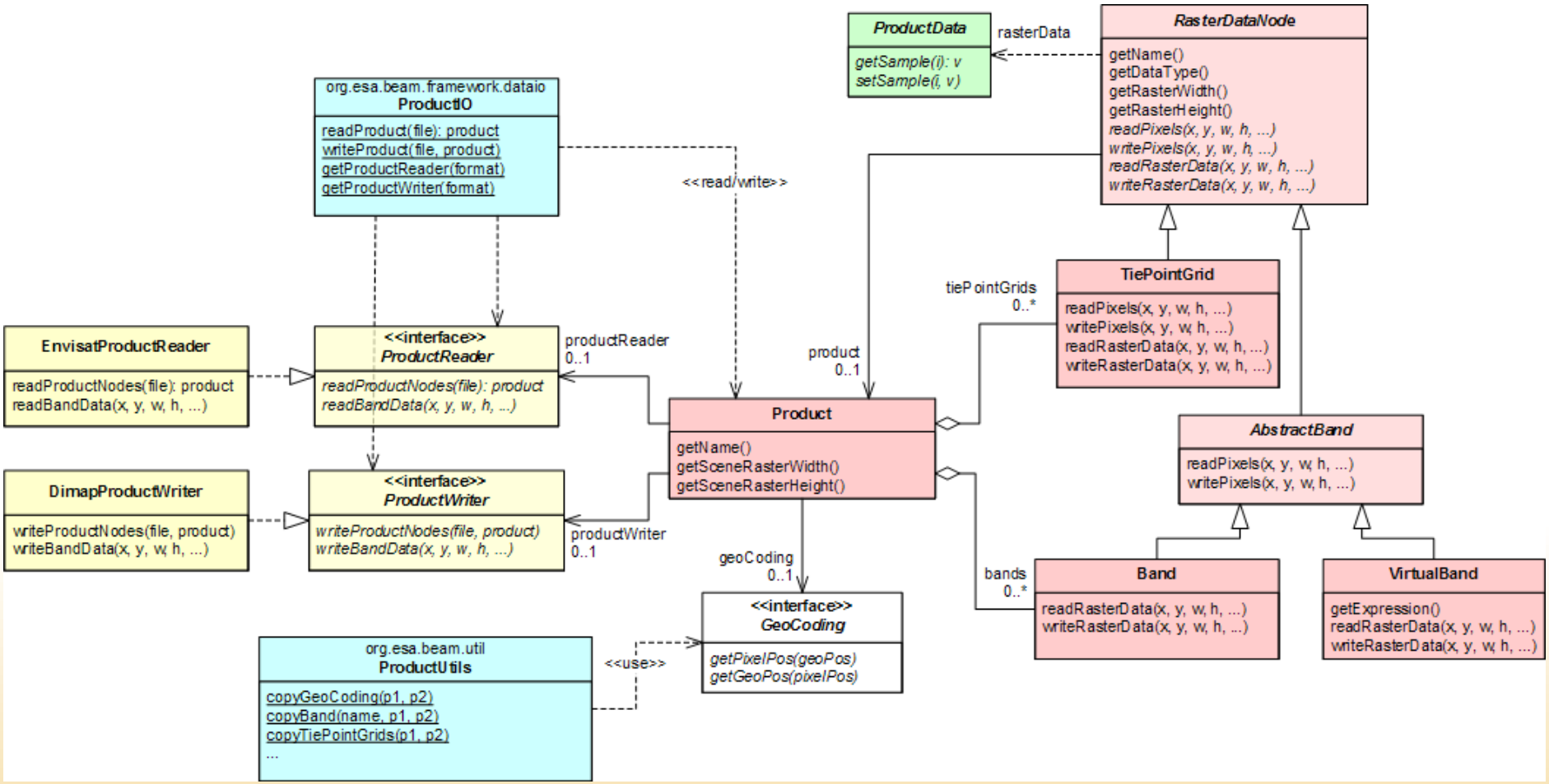
Ex2: Lead Through

- Open project **Ex2**
- Inspect Java code
- Run code, edit configuration in order to specify input file
- Inspect BEAM-DIMAP data product: **FLH.dim**, **FLH.data**
- Run program in debug mode
- Open configuration dialog, and add a VISAT configuration (see next slide) to look into generated FLH
- Change code!
  - Output format as parameter: e.g. NetCDF, GeoTIFF
  - Normalize L1, L2, and L3 using sun spectral flux and cos(VZA) or make code handle MERIS L2 inputs ("radiance_" → "reflec_")
  - Parameterise bands used for provision of L1, L2, and L3

VISAT Invocation Configuration

- Main class:
  - **com.bc.ceres.launcher.Launcher**
- Java VM options:
  - **-Xmx1024M -Dceres.context=beam**
- Program arguments:
  - **~/IdeaProjects/Ex2/FLH.dim**
- Working directory:
  - **~/beam-4.10.3**
- Use classpath of module:
  - **Ex2**

Product Data Model (2/2)



Exercise 3: Integrated FLH tool extending BEAM / VISAT

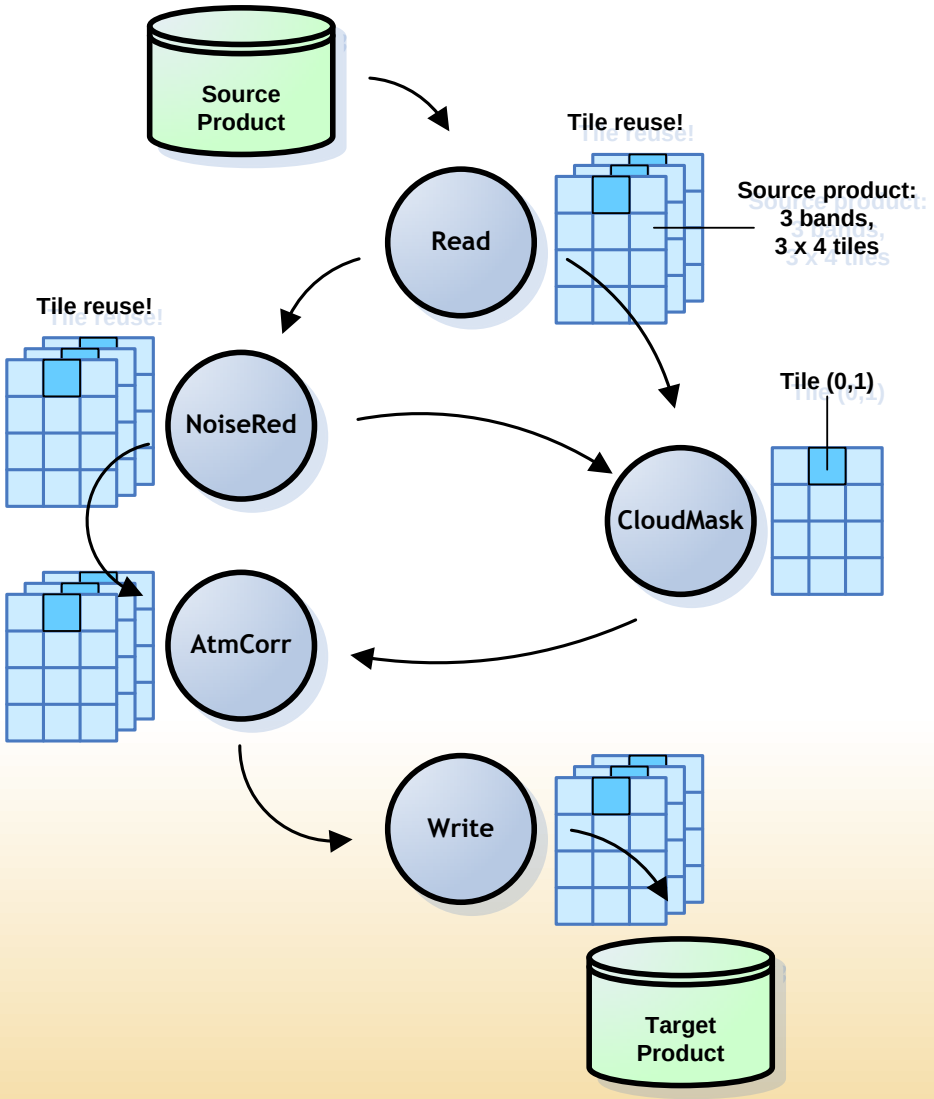
Exercise 3: Contents

- Introduction to BEAM **plug-ins** and the module descriptor file
- Introduction to the *BEAM graph processing framework* - **GPF**
- Turn the FLH code into a GPF **operator**, a special plug-in type for data processors
- Introduce the GPF command-line tool **gpt**
- Demonstrate how the FLH operator integrates into **VISAT** and the command-line tool **gpt**

Processing Framework Requirements

- Processor implementation code shall abstract from
  - physical file format of EO data
  - file I/O
  - configuration & parameterisation, parameter value access
- Avoid I/O overhead between processing steps
- Exploit multi-core CPU architectures
- The framework shall be extendible, e.g. it shall be possible to add new processors as plug-ins
- Create processing chains or even graphs composed of processing nodes. Processing nodes shall
  - be reusable in other graphs, e.g. MERIS Cloud Screening
  - be easily configurable in terms of processing parameters
  - be easily configurable in terms of the algorithm implementation

Processing Graph Example



BEAM Graph Processing Framework

- Simple data processing and programming model
  - For a particular node, clients (Java developers) implement how a raster tile is being computed
  - Parameter values are "injected" by the framework
- Implements the "pull-processing" paradigm
  - A processor is represented by its processing graph
  - Product processing requests are translated into raster-tile-requests propagated backwards through the graph
  - Tile requests are automatically parallelised depending on CPU cores
- Generated user interfaces: Command-line tool **gpt** and **VISAT** GUI
- Operators can be dynamically added to BEAM via plug-in modules
- Allows to construct directed, acyclic processing graphs

Processing Framework

- **Why "Framework"?**
  - BEAM calls your code when it is time to *Don't call us, we call you*.
  - Abstract interfaces comprising **callback** functions clients must implement
- **Rapid processor development**
  - Forget about processing environment
  - Concentrate on data and algorithm

GPF Operator Code

```java
@OperatorMetadata(alias = "NoiseRedOp",
                  version = "1.0", authors = "N.Fomferra, M.Zuehlke",
                  copyright = "(c) 2008 by Brockmann Consult",
                  description = "Performs a noise reduction on CRIS/Proba images")
public class NoiseRedOp extends Operator {

    @SourceProduct Product source;
    @TargetProduct Product target;

    @Parameter(defaultValue = "25", interval = "(0,50]",
               description = "The number of scans to use for smoothing")
    int smoothingOrder;

    @Parameter(defaultValue = "false",
               description = "Whether or not to perform sclit correction")
    boolean slitCorrection;

    @Parameter(defaultValue = "N4", valueSet = {"N2", "N4", "N8"},
               description = "The type of neighbourhood pixel consideration")
    String neighbourhoodType;

    @Override
    public void initialize() throws OperatorException {
        int width = source.getSceneRasterWidth();
        int height = source.getSceneRasterHeight();
        target = new Product("noisered", "", width, height);
        // todo - add bands to target product
    }

    @Override
    public void computeTileStack(Map<Band, Tile> bandTileMap, Rectangle rectangle,
                                 ProgressMonitor progressMonitor) throws OperatorException {
        // todo - compute target tiles in 'bandTileMap' for given 'rectangle' here.
    }
}
```

GPF Operator Configuration

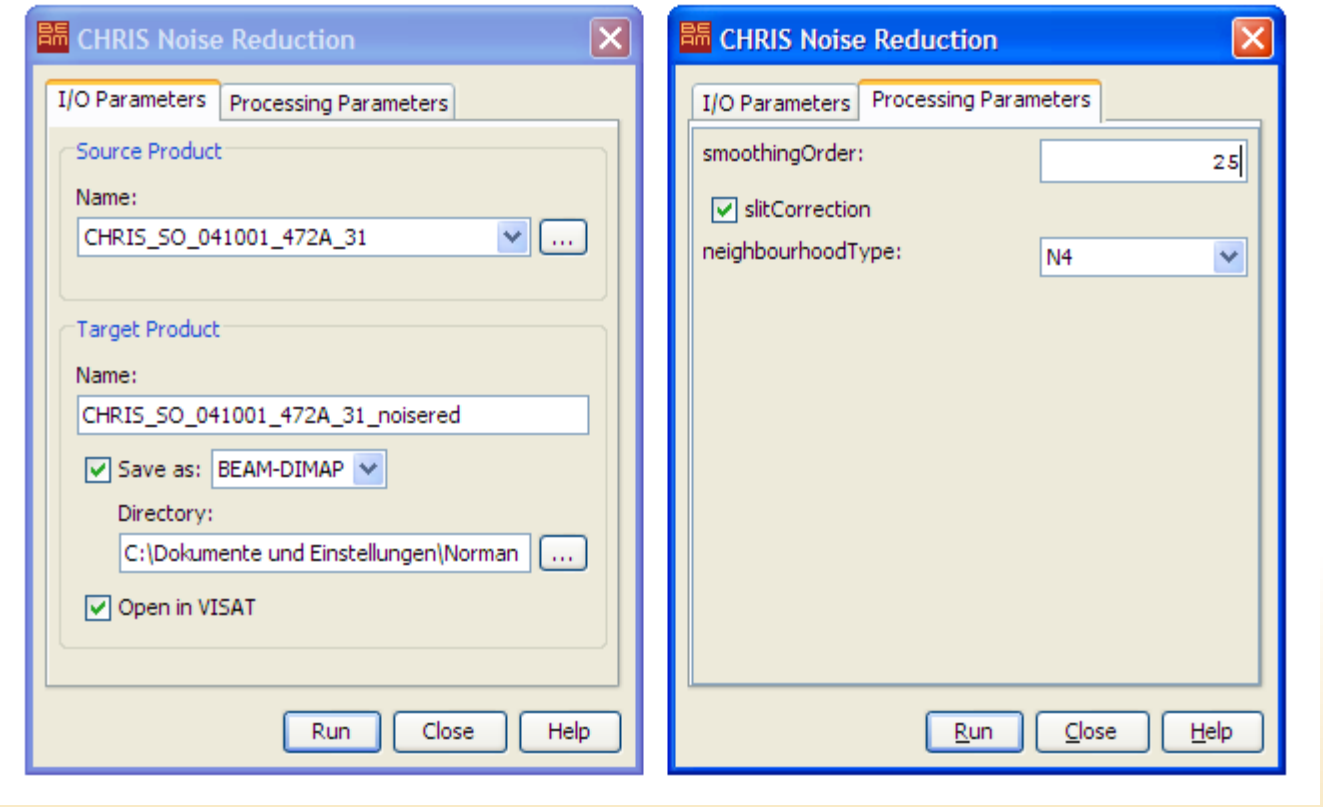
```
<node id="noisered">
    <operator>NoiseRedOp</operator>
    <sources>
        <sourceProduct>${sourceProduct}</sourceProduct>
    </sources>
    <parameters>
        <smoothingOrder>23</smoothingOrder>
        <slitCorrection>true</slitCorrection>
        <neighbourhoodType>N4</neighbourhoodType>
    </parameters>
</node>
```

gpt – The GPF Command-Line Tool

Command Line

```
> gpt -h NoiseRedOp
Usage:
  gpt NoiseRedOp [options]

Description:
  Performs a noise reduction on CRIS/Proba images


Source Options:
  -Ssource=<file>    Sets source 'source' to <filepath>.
                     This is a mandatory source.

Parameter Options:
  -PslitCorrection=<boolean>      Whether or not to perform sclit correction
                                  Default value is 'false'.
  -PneighbourhoodType=<string>    The type of neighbourhood pixel consideration
                                  Value must be one of 'N2', 'N4', 'N8'.
                                  Default value is 'N4'.
  -PsmoothingOrder=<int>          The number of scans to use for smoothing
                                  Valid interval is (0,50].
                                  Default value is '25'.

Graph XML Format:
  <graph id="someGraphId">
    <node id="someNodeId">
      <operator>NoiseRedOp</operator>
      <sources>
        <source>${source}</source>
      </sources>
      <parameters>
        <slitCorrection>boolean</slitCorrection>
        <neighbourhoodType>string</neighbourhoodType>
        <smoothingOrder>int</smoothingOrder>
      </parameters>
    </node>
  </graph>

> gpt NoiseRedOp -PslitCorrection=true -Ssource=in.hdf_
```

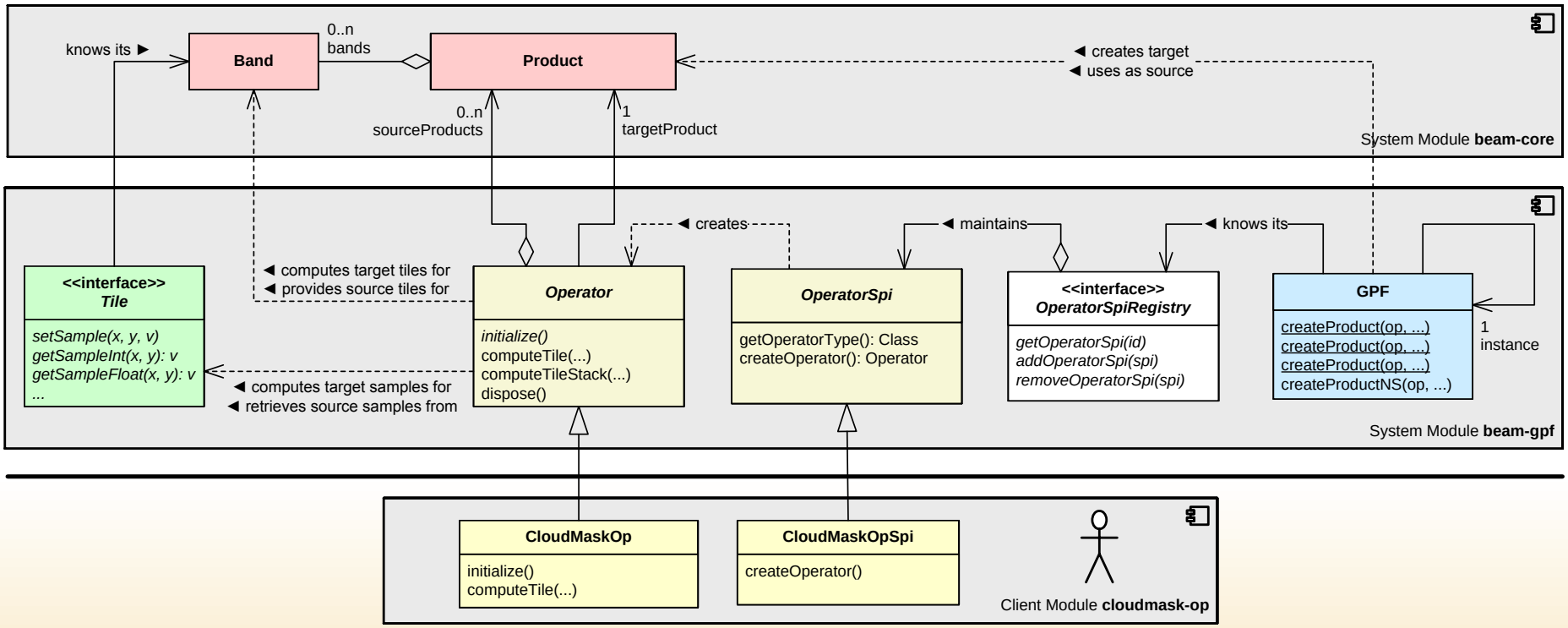
GPF Processor GUI



GPF Programming Model

```
Product p0 = ProductIO.readProduct("MER_1P.N1");

Map<String,Object> params1 = ...;
Map<String,Object> params2 = ...;
Map<String,Object> params3 = ...;

Product p1, p2, p3;
p1 = GPF.createProduct("SmileCorrOp", params1, p0);
p2 = GPF.createProduct("CloudMaskOp", params2, p1);
p3 = GPF.createProduct("NoiseRedOp", params3,
                       p1, p2);
WriteOp.writeProduct(p3, new File("TEST.DIM"),
                     "BEAM-DIMAP",
                     ProgressMonitor.NULL);
```

GPF Architecture (1/2)



GPF Architecture (2/2)

- Clients provide their compiled **Operator** Java code packed into JAR modules
- A JAR module publishes its operator services via metadata (Service Provider Interface, SPI)
  - **META-INF/services/ org.esa.beam.framework.gpf.OperatorSpi**
- Once GPF is started, e.g. by **VISAT** or **gpt**, it loads all operator plug-ins it finds on its dynamic classpath
  - **%BEAM4_HOME%/modules/*.jar**
  - **%BEAM4_HOME%/lib/*.jar**

Ex3: Lead Through (1/3)

- Open **Ex3** project in IDEA
- Inspect the Java source code in the **src** directory
- Understand the abstract Java class **Operator**
  - Special GPF Java *annotations* **@OperatorMetadata**, **@SourceProduct**, **@TargetProduct**, **@Parameter**
  - *Callback* methods **initialize()** and **computeTileStack()**
- Inspect special resource files
  - Inspect module descriptor file **src/module.xml**
  - Inspect the file in **META-INF/services/**

Ex3: Lead Through (2/3)

- Create an IDEA configuration to invoke the standard **gpt** tool → check that the new FLH operator plugged-in
- Run **gpt FLH** <file>
- Create an IDEA configuration to invoke the **VISAT** → check that the new FLH processor plugged-in. Check how parameters translate in GUI components
- Run the FLH processor from VISAT

Ex3: Lead Through (2/3)

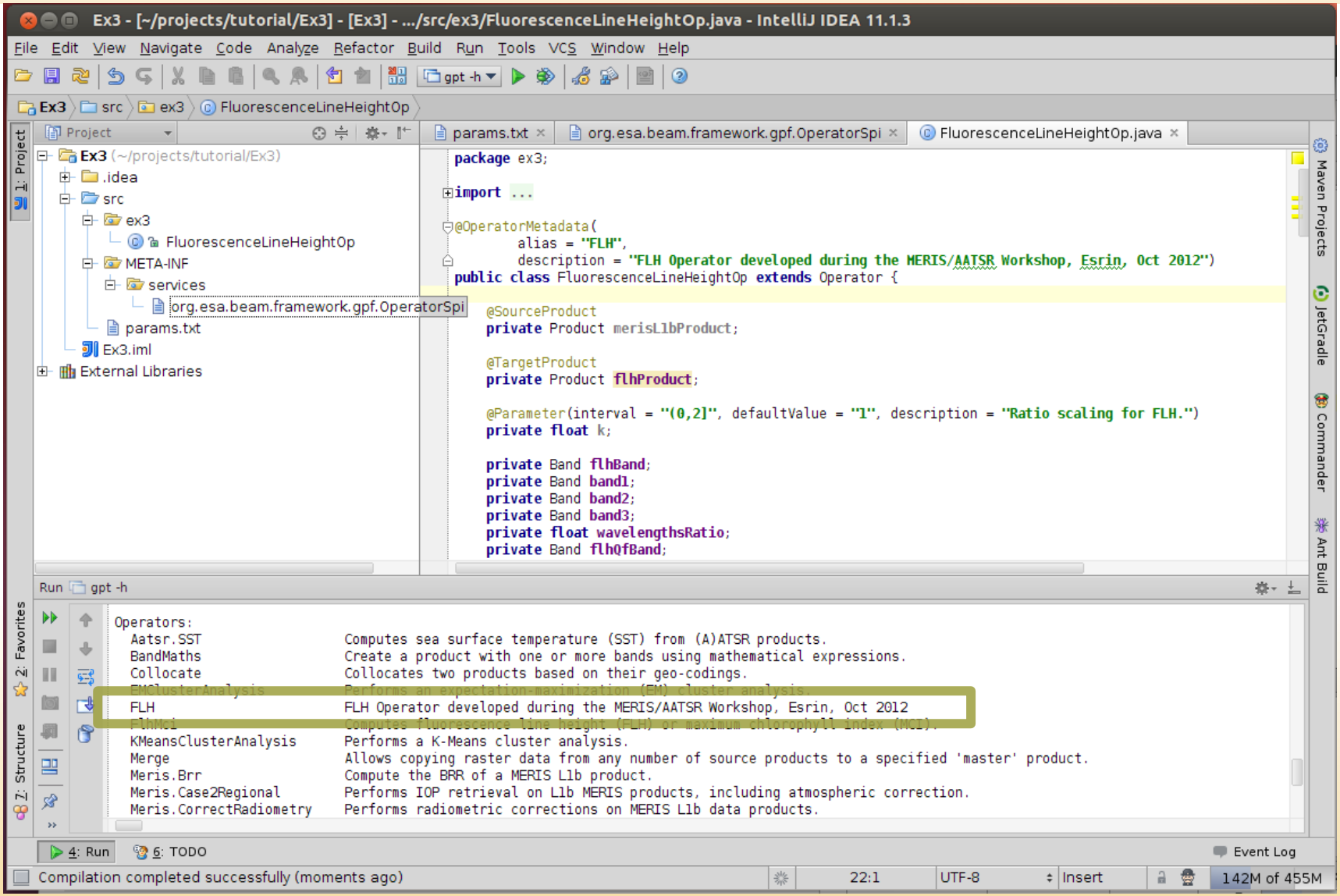
- In IDEA, open project settings, add an artifact **ex3.jar** in order to create a Java JAR module (see next slides)
- Put **ex3.jar** into BEAM's **module** directory, start VISAT → Extension installed
- Check installed module in Module Manager

→ This plug-in module can now be *deployed*

gpt Invocation Configuration

- Main class:
  - **com.bc.ceres.launcher.Launcher**
- Java VM options:
  - **-Xmx1024M -Dceres.context=beam -Dbeam.mainClass=org.esa.beam.framework.gpf.main.GPT**
- Program arguments:
  - **-h** or <*op-name*> <*source-file*>
- Working directory:
  - **~/beam-4.10.3**
- Use classpath of module:
  - **Ex3**

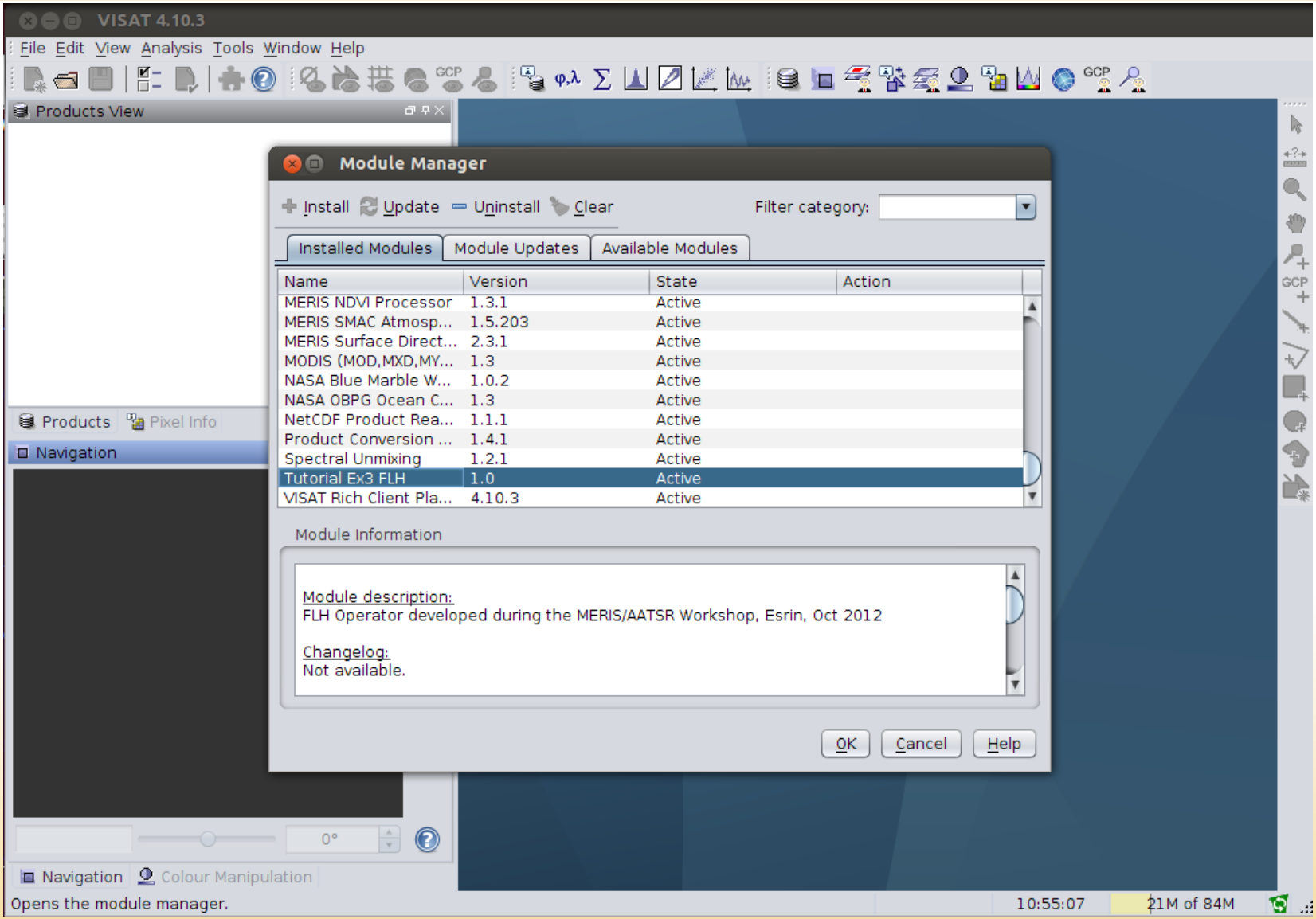
FLH Tool in gpt



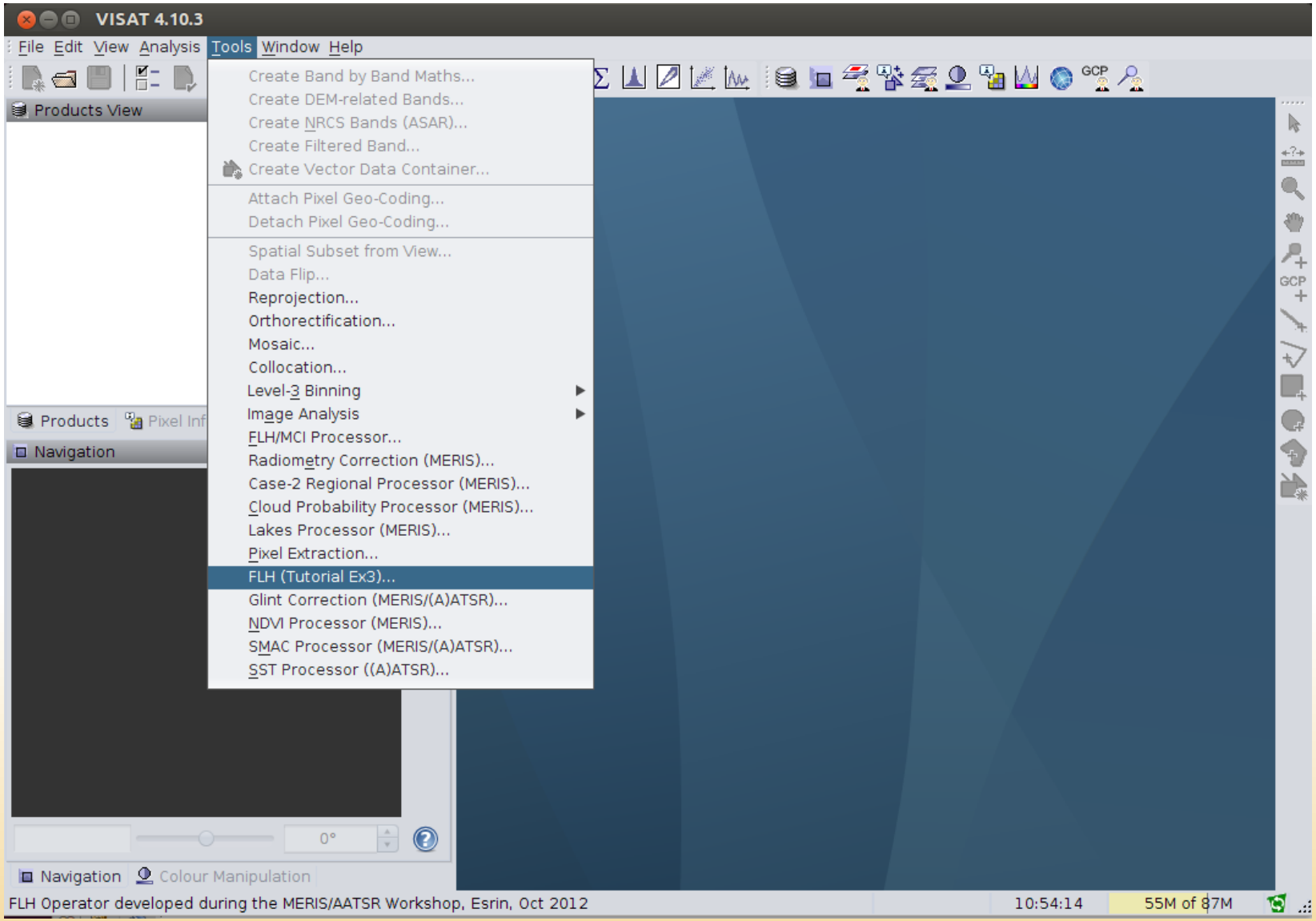
VISAT Invocation Configuration

- Main class:
  - **com.bc.ceres.launcher.Launcher**
- Java VM options:
  - **-Xmx1024M -Dceres.context=beam**
- Program arguments:
  - *none*
- Working directory:
  - **~/beam-4.10.3**
- Use classpath of module:
  - **Ex3**

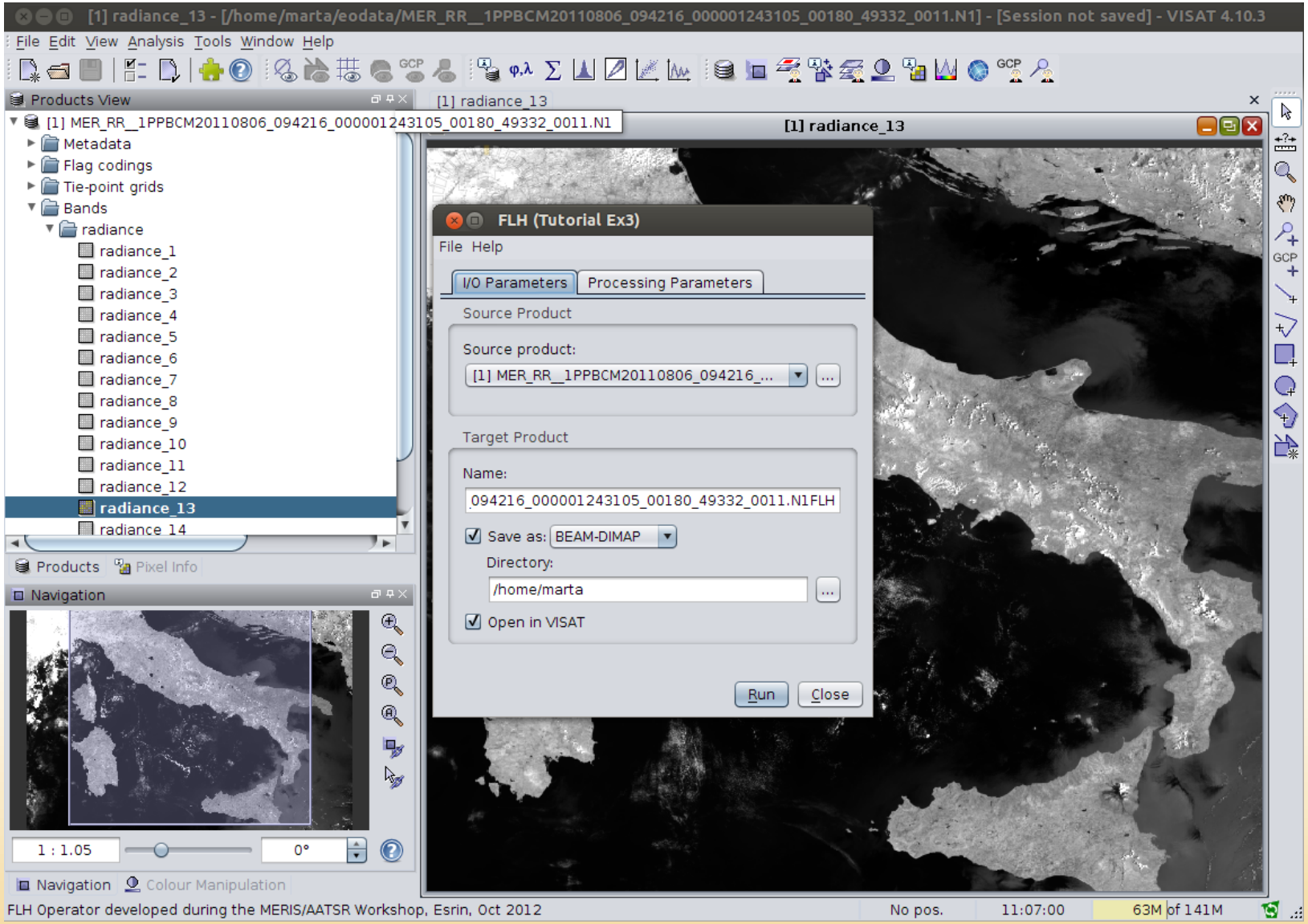
FLH Tool in Module Manager



FLH Tool in Main Menu



FLH Tool GUI


[1] radiance_13 - [/home/marta/eodata/MER_RR__1PPBCM20110806_094216_000001243105_00180_49332_0011.N1] - [Session not saved] - VISAT 4.10.3
File Edit View Analysis Tools Window Help
Products View
[1] MER_RR__1PPBCM20110806_094216_000001243105_00180_49332_0011.N1
Metadata
Flag codings
Tie-point grids
Bands
radiance
radiance_1
radiance_2
radiance_3
radiance_4
radiance_5
radiance_6
radiance_7
radiance_8
radiance_9
radiance_10
radiance_11
radiance_12
radiance_13
radiance_14
Products
Pixel Info
Navigation
1 : 1.05
0°
Navigation
Colour Manipulation
[1] radiance_13
FLH (Tutorial Ex3)
File Help
I/O Parameters
Processing Parameters
Source Product
Source product:
[1] MER_RR__1PPBCM20110806_094216_...
Target Product
Name:
094216_000001243105_00180_49332_0011.N1FLH
Save as:
BEAM-DIMAP
Directory:
/home/marta
Open in VISAT
Run
Close
GCP
FLH Operator developed during the MERIS/AATSR Workshop, Esrin, Oct 2012
No pos.
11:07:00
63M of 141M

Options for BEAM in Batch-Mode

1. Use BEAM's command-line tools
   - from a command-line shell
   - from shell scripts
   - from Python, IDL, MATLAB scripts

2. Use the BEAM Java libraries to directly call BEAM functions
   - from your **Java** program
     → *BEAM Programming Tutorial, Thursday, 15:00-17:00*
   - from your **C** or **Python** program
     → *In progress, 1st version expected Spring 2013*
     → *We are happy to consider your requirements!*

3. Use the VISAT Scripting Console (experimental)
   - Use BEAM libraries within VISAT to automate work
   - **Python** (Jython) and **JavaScript**, see VISAT Help

Outlook

- C & Python Language Support
  - C Processing and Analysis API
  - Python Processing and Analysis API
  - Spring 2013

- Sentinel Data Support
  - S1 SAR
  - S2 MSI
  - S3 OLCI, SLSTR, Synergy
  - Summer 2013

Thanks for your attention!

→ Get instant support in the BEAM user forum

Complementary Material

GPF Operator Anatomy (1/3)

```
@OperatorMetadata alias   “NoiseRedOp",
                  version   "1.0",
                  authors   "Ralf Quast",
                  copyright   "(c) 2008 by Brockmann Consult",
                  description   "Performs a noise reduction on”
                                  + “ CHRIS/Proba images."
public class NoiseRedOp extends Operator {
    @SourceProduct alias   "source"
    private Product sourceProduct
    @TargetProduct
    private Product targetProduct
    @Parameter defaultValue “false”
    private boolean slitCorrection
    @Parameter(interval “(0,50]”, defaultValue=“25”)
    private int smoothingOrder;
    @Parameter(valueSet={“N2”, “N4”, “N8”}, defaultValue=“N2”)
    private String neighbourhoodType;
    // ...
}
```

GPF Operator Anatomy (2/3)

If single target bands can be computed independently of each other, e.g. NDVI or algorithms which perform single band filtering:

```
public class NoiseRedOp extends Operator {
   public void initialize() throws OperatorException {
       // validate and process source products and parameter values
       // create, configure and set the target product
   }
   public void computeTile(Band targetBand,
                           Tile targetTile,
                           ProgressMonitor pm)
                                    throws OperatorException {
       // Obtain source tiles for used bands of source products
       // Process samples of source tiles to samples of target tile
       // Set samples of single target tile
   }
}
```

GPF Operator Anatomy (3/3)

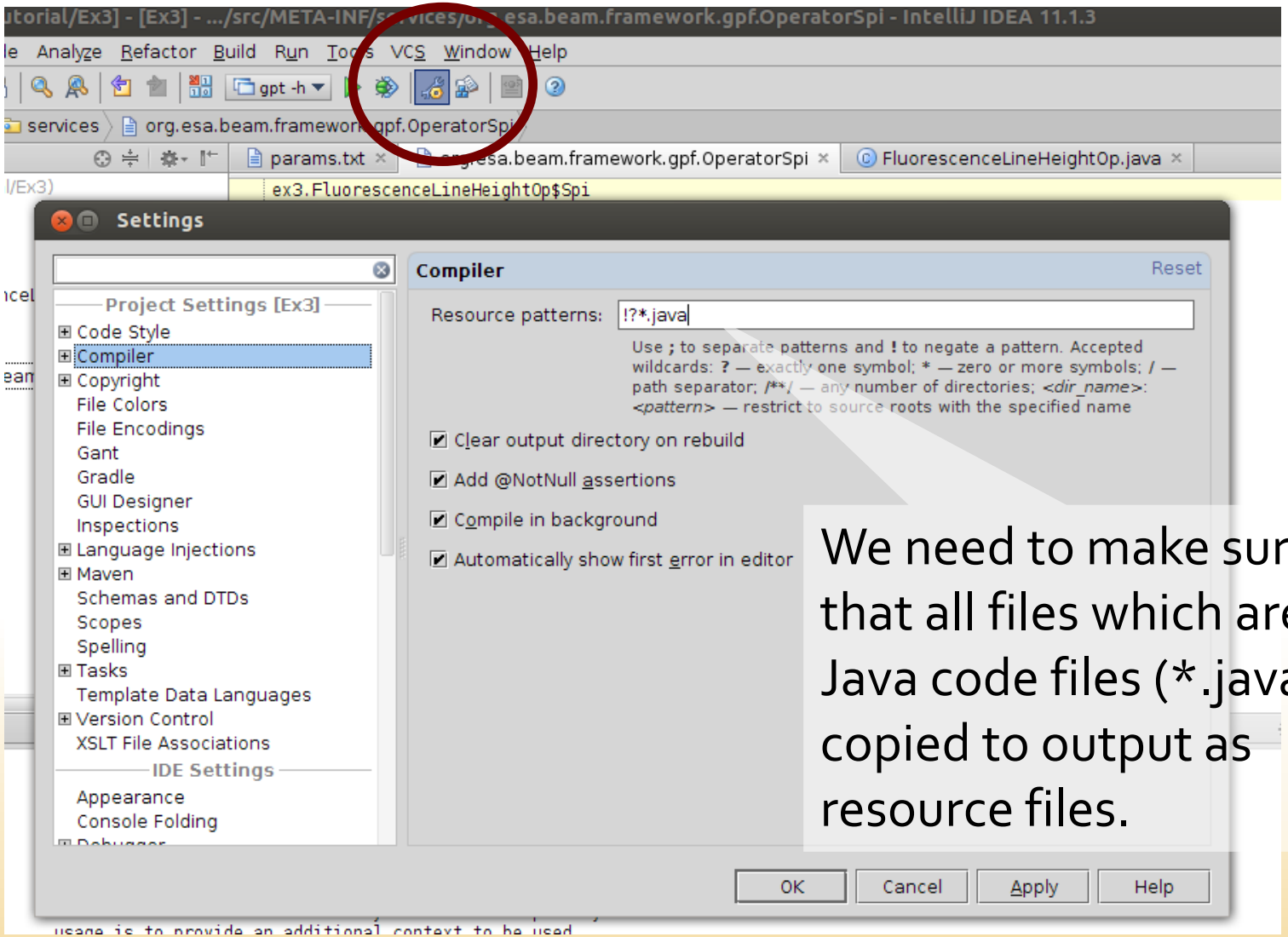
If single target bands cannot be computed independently of each other, e.g. model inversion algorithms based on neural networks:

```
public class NoiseRedOp extends Operator {
   public void initialize() throws OperatorException {
       // validate and process source products and parameter values
       // create, configure and set the target product
   }
   public void computeTileStack(Map<Band, Tile> targetTiles,
                                Rectangle targetTileRectangle,
                                ProgressMonitor pm)
                                        throws OperatorException {
       // Obtain source tiles for used bands of source products
       // Process samples of source tiles to samples of target tiles
       // Set samples of all given target tiles
   }
}
```

How GPF "drives" Operators

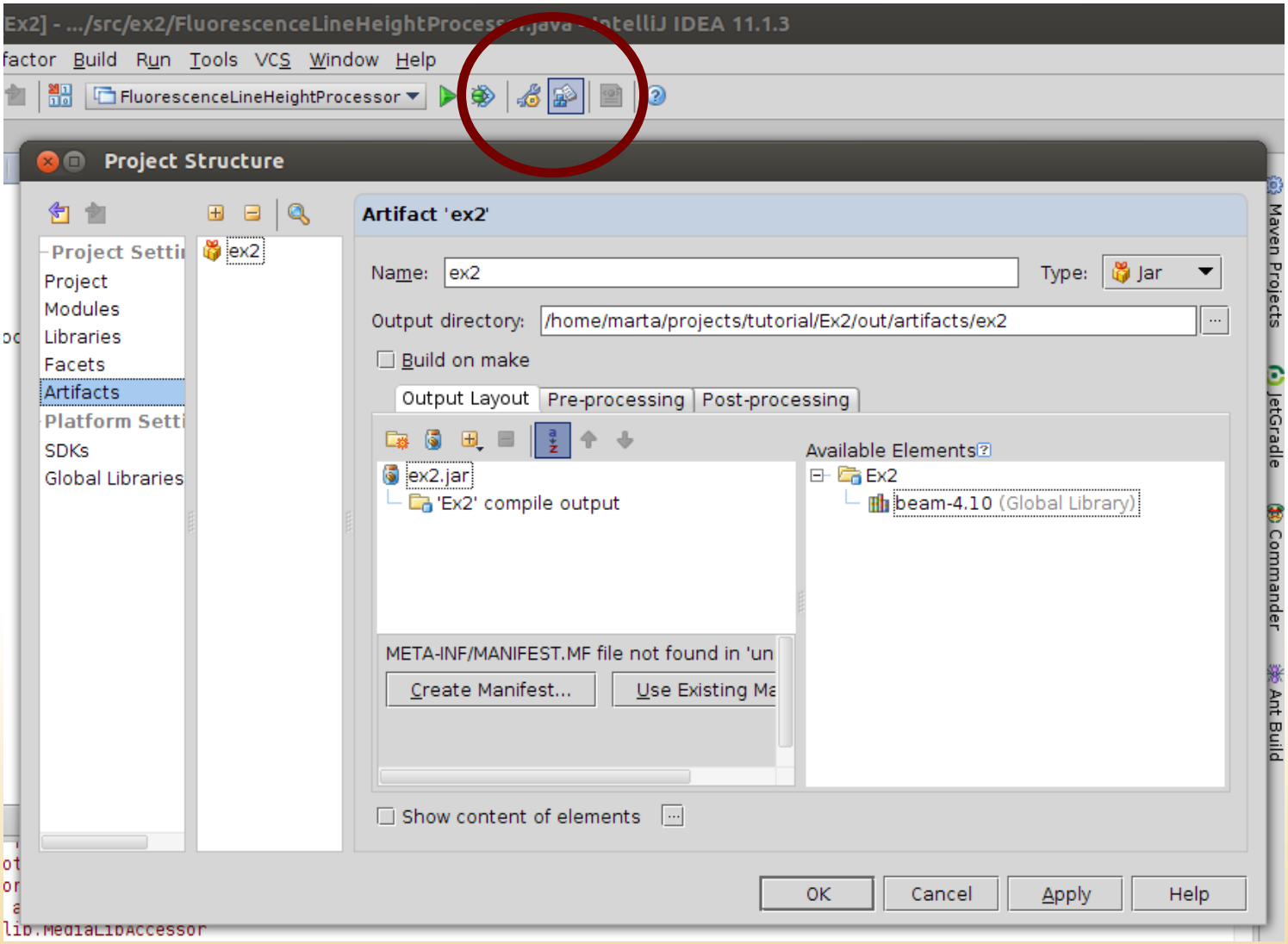
1. GPF knows about all registered Operators (Plug-in)
2. GPF encounters a required node (GUI, CLI, XML graph)
   1. The client's Operator class is found by its alias
   2. The **SourceProduct**, **TargetProduct** and **Parameter** annotations are analysed
   3. The client's Operator object is created
   4. Source products and parameter values are injected
3. GPF calls **Operator.initialize()**.
   In this method, the client provides code to
   1. validate and process source products and parameter values
   2. create, configure and set the target product
4. GPF calls either **Operator.computeTile()** or **Operator.computeTileStack()**in case raster data is required

Configuring Resources in IDEA

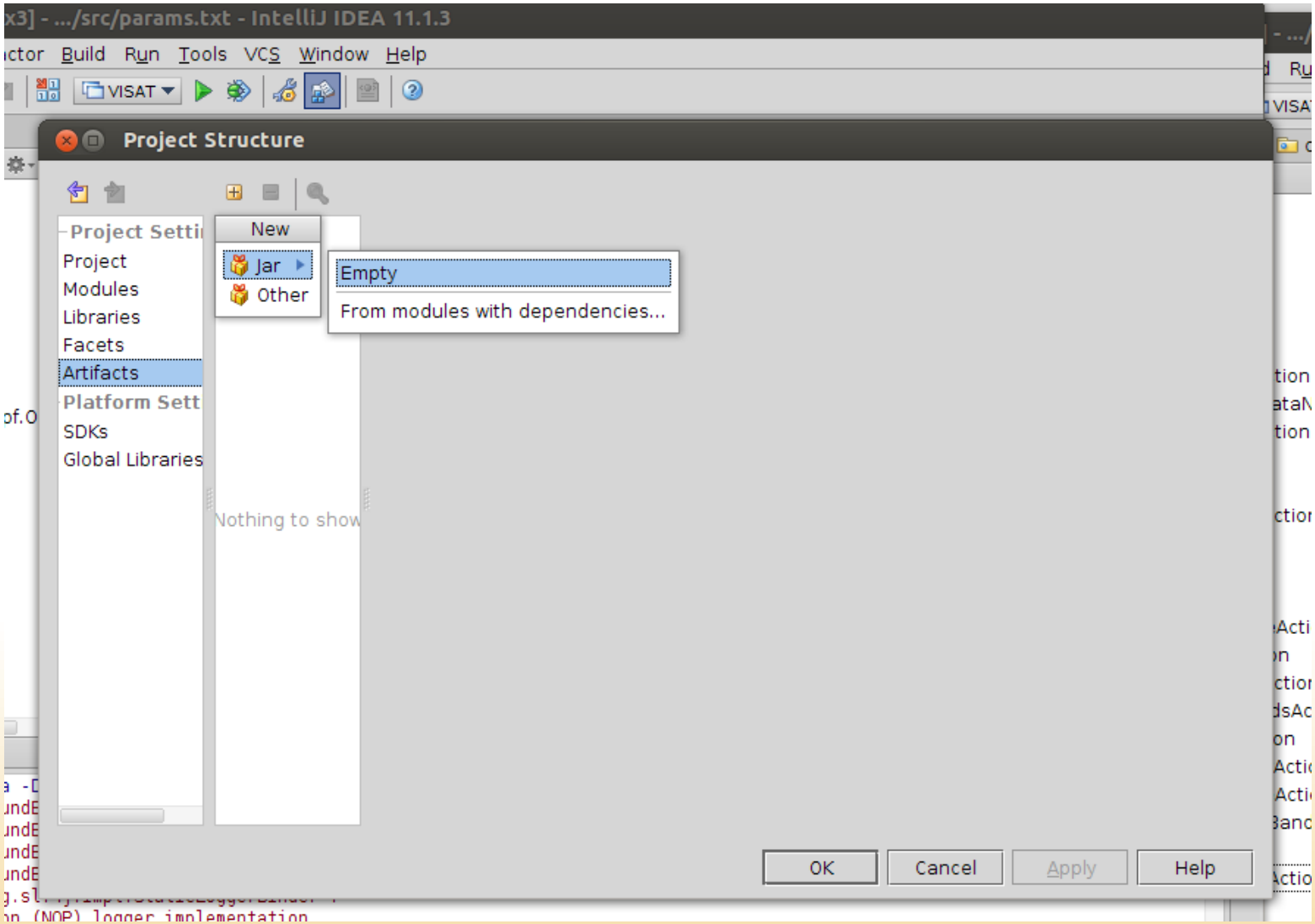


We need to make sure that all files which are not Java code files (*.java) are copied to output as resource files.

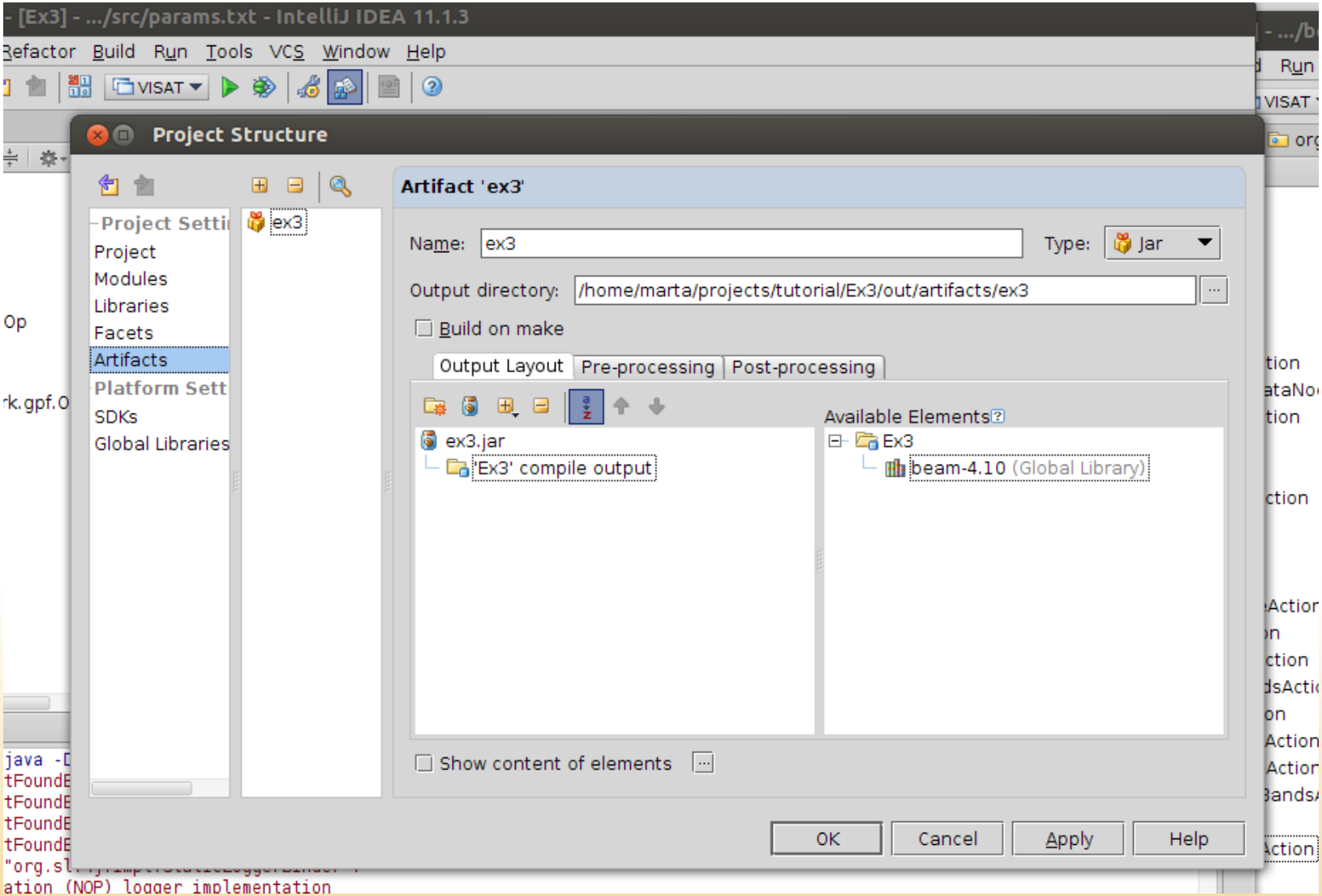
Creating a JAR in IDEA (1/4)



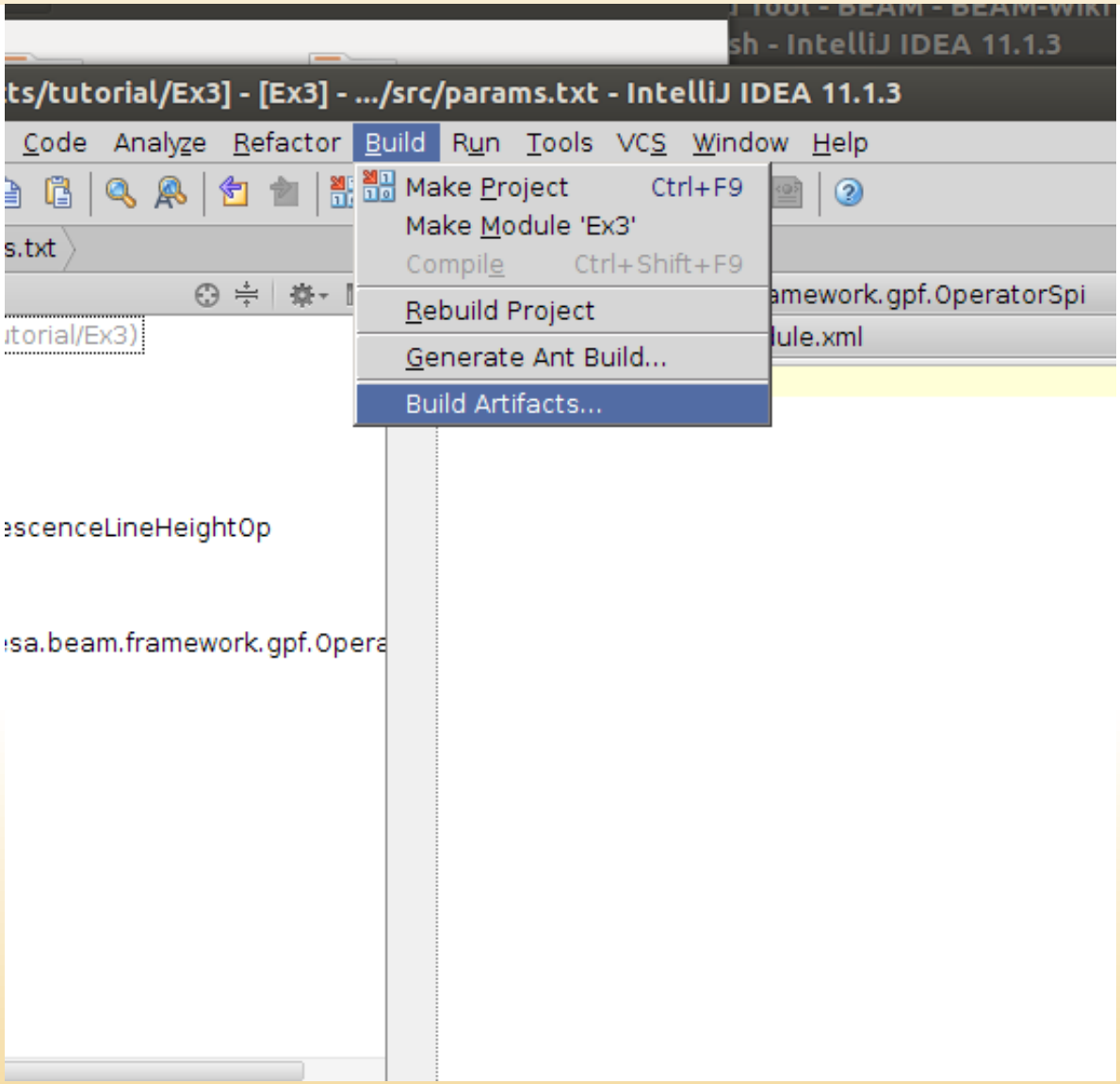
Creating a JAR in IDEA (2/4)



Creating a JAR in IDEA (3/4)



Creating a JAR in IDEA (4/4)



Invocation of Command-line Tools

Invocation of **gpt.bat** via a dedicated launcher program (**ceres-launcher.jar**). All JARs found in BEAM's **lib** and **modules** directories are put on the classpath (see also scripts in **%BEAM4_HOME%/bin**):

```
@echo off

set BEAM4_HOME=C:\Program Files\beam-4.10.3

"%BEAM4_HOME%\jre\bin\java.exe" ^
    -Xmx1024M ^
    -Dceres.context=beam ^
    "-Dbeam.mainClass=org.esa.beam.framework.gpf.main.GPT" ^
    "-Dbeam.home=%BEAM4_HOME%" ^
    "-Dncsa.hdf.hdflib.HDFLibrary.hdflib=%BEAM4_HOME%\...\jhdf.dll" ^
    "-Dncsa.hdf.hdf5lib.H5.hdf5lib=%BEAM4_HOME%\...\jhdf5.dll" ^
    -jar "%BEAM4_HOME%\bin\ceres-launcher.jar" %*
exit /B 0
```