NEST (Next ESA SAR Toolbox) Developer Training Course

CONTENTS

- Building the Source Code
- General Design
- Creating an Operator
- Creating a Reader
- Creating a Writer

Building the NEST Source Code

OPEN SOURCE

The NEST software and full source code is distributed freely under the GNU GPL license.

- Open source makes software inherently independent of specific vendors, programmers and suppliers. The software can be freely distributed and shared by large communities including the source code and the right to modify it.
- This ensures that there isn't a single entity on which the future of the software depends on and allows for unlimited improvements and tuning of the quality and functionality of the software.
- By making NEST open source, future evolution and growth of the toolbox will be possible by the community of users and developers that contribute back to the project.

PREREQUISITES

- Java JDK
  - http://java.sun.com/javase/downloads/
- Apache Maven
  - http://maven.apache.org/
- Integrated Development Environment
  - IntelliJ http://www.jetbrains.com/idea/
  - Eclipse http://www.eclipse.org/
  - Wordpad

SETUP ENVIRONMENT VARIABLES

- Add the environment variables
- If you installed Java, maven to a folder ~/bin or c:\bin\ then the paths should look like these

- Linux
  - In ~/.bashrc
  - export JAVA_HOME=~/bin/java/jdk1.6.0_17
  - export MAVEN_HOME=~/bin/maven
  - export PATH=$PATH:$JAVA_HOME/bin:$MAVEN_HOME/bin

- Windows
  - In ControlPanel -> System -> Advanced Settings -> Environment Variables
  - New JAVA_HOME c:\bin\java\jdk1.6.0_17
  - New MAVEN_HOME c:\bin\maven
  - New PATH %PATH%;%JAVA_HOME%\bin;%MAVEN_HOME%\bin

GET THE CODE

- Unzip nest-3C-src.zip from the USB disk
- Or download and unzip source from a stable release from the website

  http://earth.esa.int/nest

- Or checkout the latest nightly development build from the CVS repository
  - cvs -d :pserver:anonymous@www.array.ca:2401/repos co nest

SETUP BUILD ENVIRONMENT

- Apache Maven is a build manager that lets you create and automate the build environment from a Project Object Model (POM)
- The NEST pom.xml list all modules and dependencies to include in the build
- Dependencies are downloaded as needed from a repository
- Maven commands
  - mvn compile
  - mvn compile idea:idea
  - mvn package assembly:assembly
- Other useful commands
  - mvn compile eclipse:eclipse
  - Mvn install
  - mvn clean

PRODUCING A RELEASE SOFTWARE

- mvn package assembly:assembly
- This creates a folder Nest-3C/target that contains the software for each platform
- Copy the C:\bin\java\jdk1.6.0_17\jre folder into the folder appropriate for your system.
  - E.g. copy the jre into nest-3C/target/nest-3C-bin-win.dir

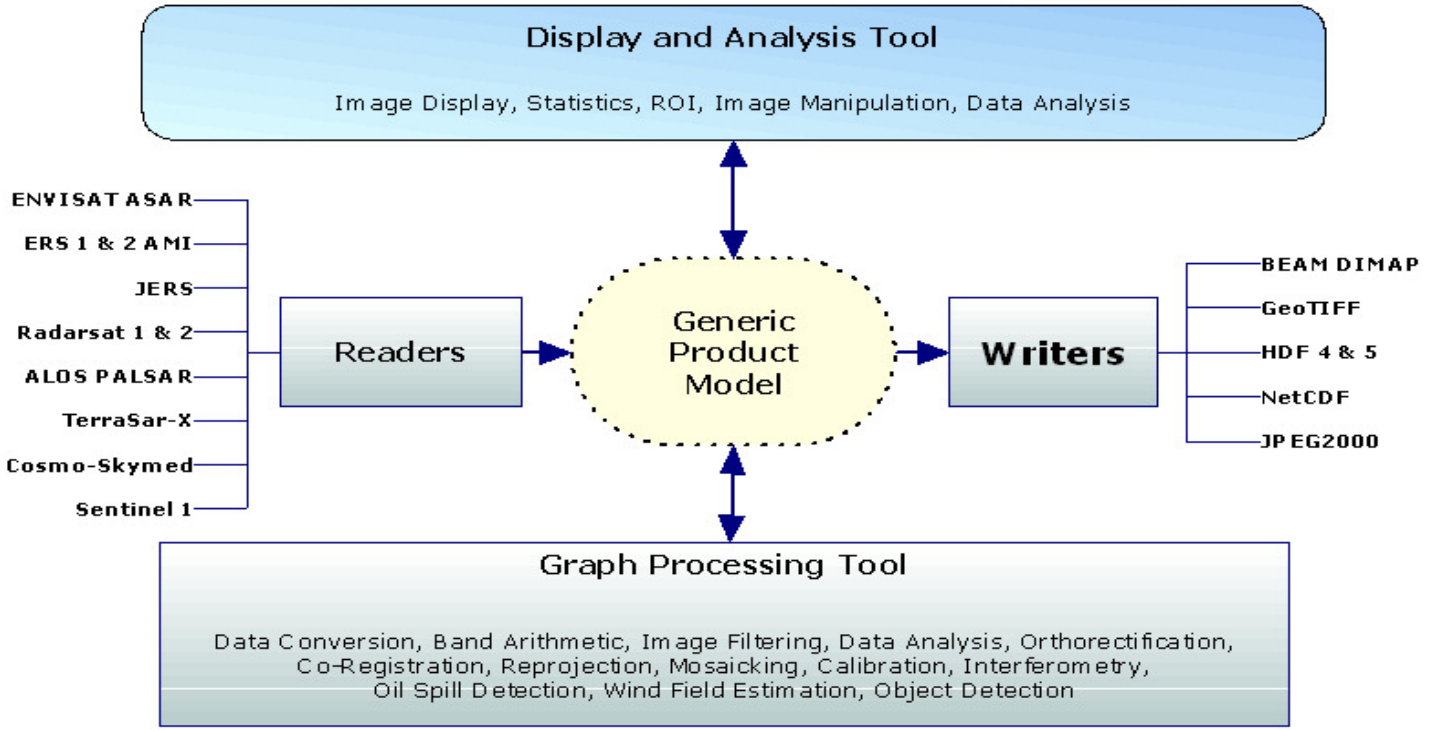
NEST General Design

MODULAR DEVELOPMENT

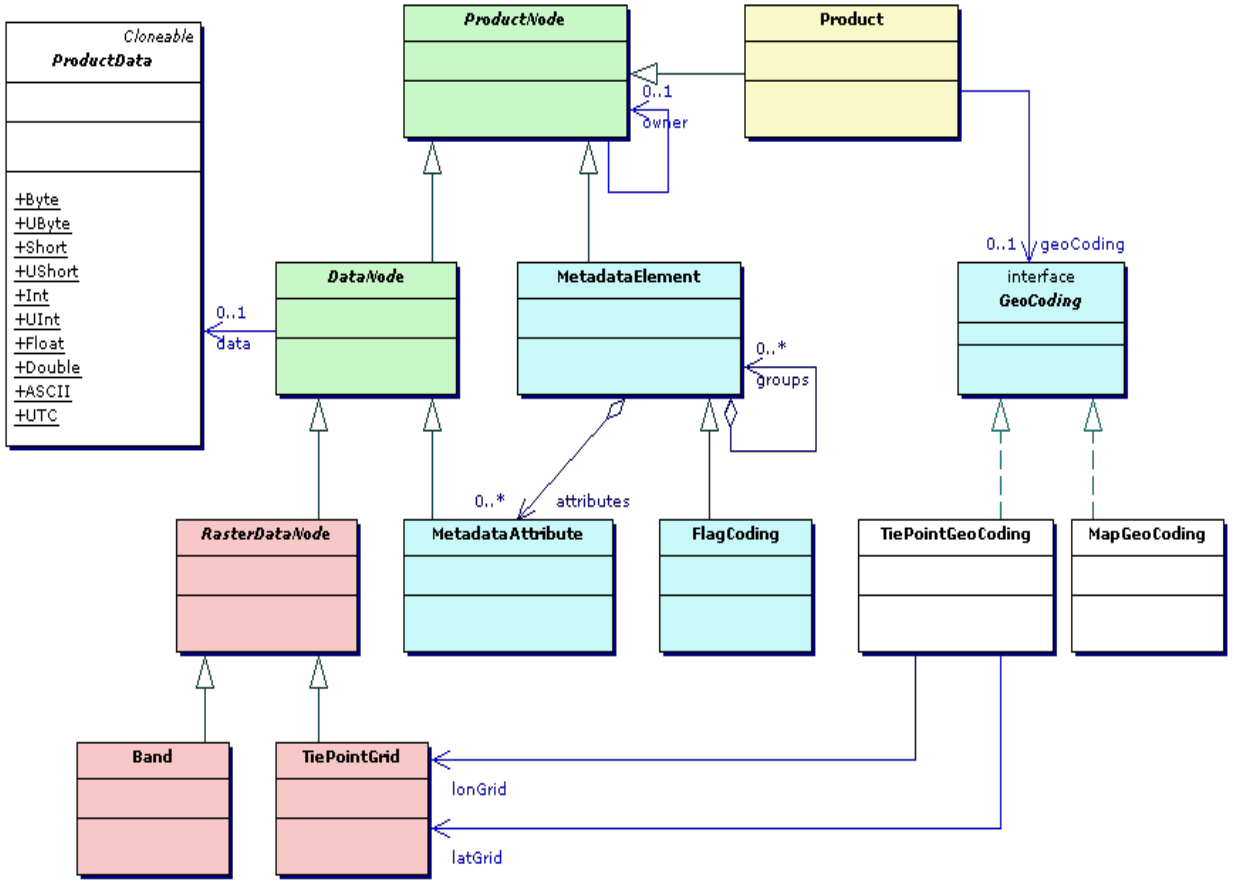
- API provided with NEST allows for easy extension by users to add data readers/writers of other formats and to support data formats of future missions.
- Plug-in modules can be developed separately and shared by the user community
- Processors can be easily extended without needing to know about the complexities of the whole software

GENERIC PRODUCT MODEL (GPM)

The GPM is a common, unified data model designed so that all data readers convert data into this data model and all analysis and processing tools use this data model exclusively.



PRODUCT MODEL



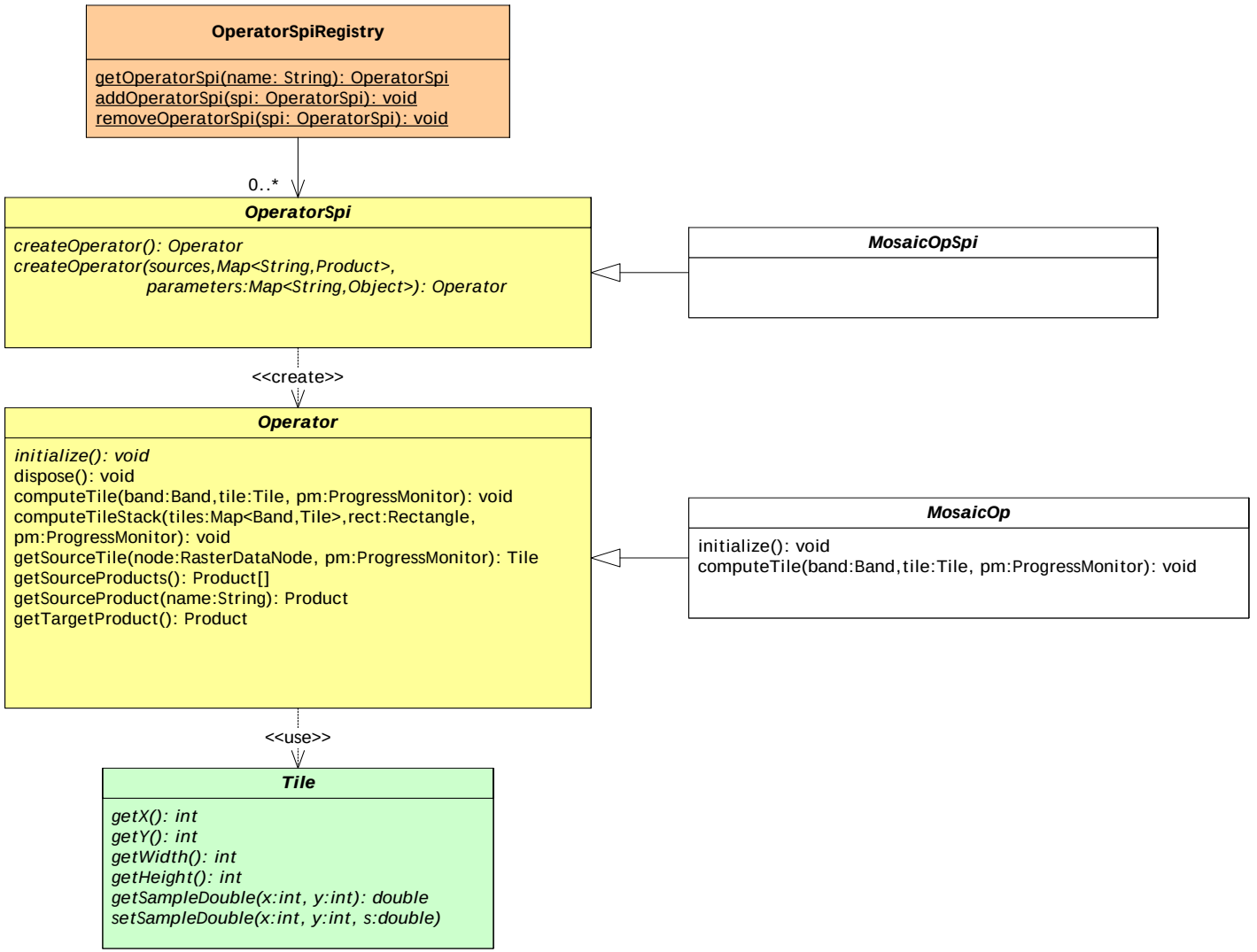
- Tie Point Grid
  - Sub-sampled raster data
- Band
  - Data at scene resolution
- Flag Coding
  - Product type specific
  - Land cover etc.
- Geocoding
  - Tie point geocoding
  - Map geo coding
- Metadata Elements
  - Header data
- DataNode
  - Raster data as int, float,...

GRAPH PROCESSING FRAMEWORK

- Processing chains are created as Directed Acyclic Graphs.
- The sources of the graph are the data product readers, and the sinks can be either a product writer or an image displayed on the DAT.

- The GPF uses a Pull Model, wherein a request is made from the sink backwards to the source to process the graph. This request could be to create a new product file or to update a displayed image. Once the request reaches a source, the image is pulled through the nodes to the sink. Each time an image passes through an operator, the operator transforms the image, and it is passed down to the next node until it reaches the sink.

THE GRAPH OPERATOR



Creating An Operator

MAVEN GPF ARCHETYPE

- A Maven Archetype is a template toolkit for generating a new module package that is already integrated and ready to use.
- Run mvn install to install the archetypes into the Maven repository
- Using the create_gpf_op script a maven archetype will create a new operator

```
create_gpf_op nest-op-segmentation
```

- The new operator will be added to the pom.xml and contain everything needed for an operator to work.
- Add it to your project

```
mvn compile idea:idea
```

- You can then begin to modify it to your needs.

THE GPF OPERATOR

- The Operator basically takes a source product as input and creates a new target product within initialize().
- The algorithm implementation for what your operator does will go inside computTile() or computeTiles().

```
public interface Operator {
    OperatorSpi getSpi();
    Product initialize(OperatorContext context);
    void computeTile(Tile targetTile, ProgressMonitor pm);
    void computeTiles(Rectangle targetTileRectangle,
            ProgressMonitor pm);
    void dispose();
}
```

INITIALIZE

- Get source metadata
- Create target product
- Add selected bands
- Update target metadata

```
public void initialize() throws OperatorException {
    getSourceMetadata();

    // create target product
    targetProduct = new Product(sourceProduct.getName(),
            sourceProduct.getProductType(),
            sourceProduct.getSceneRasterWidth(),
            sourceProduct.getSceneRasterHeight());

    // Add target bands
    addSelectedBands();

    // copy or create product nodes
    OperatorUtils.copyProductNodes(sourceProduct,
                                   targetProduct);

    // update the metadata with the affect of the processing
    updateTargetProductMetadata();
}
```

ABSTRACTED METADATA

- The Abstracted Metadata is an extract of information and parameters from the actual metadata of the product.
- The reader convert product specific metadata into this common format for all Operators to understand
- The Abstracted Metadata is the only metadata that an Operator should read from and write to.

[2] Abstracted Metadata

Name	Value	Type	Unit	Description
PRODUCT	ASA_IMP_1PNPDE20061201_055712_00...	ascii		Product name
PRODUCT_TYPE	ASA_IMP_1P	ascii		Product type
SPH_DESCRIPTOR	Image Mode Precision Image	ascii		Description
MISSION	ENVISAT	ascii		Satellite mission
PROC_TIME	15-DEC-2006 07:52:09.000000	utc	utc	Processed time
Processing system identifier	ASAR/4.02	ascii		Processing system identifier
CYCLE	53	int32		Cycle
REL_ORBIT	249	int32		Track
ABS_ORBIT	24852	int32		Orbit
STATE_VECTOR_TIME	01-DEC-2006 05:48:09.577867	utc	utc	Time of orbit state vector
VECTOR_SOURCE	FR	ascii		State vector source
NUM_SLICES	1	int32		Number of slices
first_line_time	01-DEC-2006 05:57:12.728119	utc	utc	First zero doppler azimuth time
last_line_time	01-DEC-2006 05:57:27.636332	utc	utc	Last zero doppler azimuth time
first_near_lat	33.199956	float64	deg	
first_near_long	-118.912254	float64	deg	
first_far_lat	33.313646	float64	deg	
first_far_long	-118.152459	float64	deg	
last_near_lat	34.079652	float64	deg	
last_near_long	-119.105236	float64	deg	
last_far_lat	34.192867	float64	deg	
last_far_long	-118.337328	float64	deg	
SWATH	IS6	ascii		Swath name
PASS	ASCENDING	ascii		ASCENDING or DESCENDING
SAMPLE_TYPE	DETECTED	ascii		DETECTED or COMPLEX
mds1_tx_rx_polar	VV	ascii		Polarization
mds2_tx_rx_polar		ascii		Polarization
ALGORITHM	RAN/DOP	ascii		Processing algorithm
AZIMUTH_LOOKS	4	int32		
RANGE_LOOKS	1	int32		
RANGE_SPACING	12.5	float64	m	Range sample spacing
AZIMUTH_SPACING	12.5	float64	m	Azimuth sample spacing
pulse_repetition_frequency	1705.227294921875	float64	Hz	PRF
radar_frequency	5331.004416	float64	MHz	Radar frequency
LINE_TIME_INTERVAL	0.00188045064	float64	s	
total_size	87	int32	Mb	Total product size
num_output_lines	7929	int32		
num_samples_per_line	5755	int32		
srgr_flag	1	uint8	flag	SRGR applied
avg_scene_height	80.80833435058594	float64	m	Average ccene height ellipsoid
map_projection		ascii		Map projection applied
is_terrain_corrected	0	int32	flag	orthorectification applied
dem		ascii		Digital Elevation Model used
geo_ref_system		ascii		geographic reference system
lat_pixel_res	0.0	float64	deg	pixel resolution in geocoded image
lon_pixel_res	0.0	float64	deg	pixel resolution in geocoded image
slant_range_to_first_pixel	979813.938924835	float64	m	Slant range to 1st data sample
ant_elev_corr_flag	1	uint8	flag	Antenna elevation applied
range_spread_comp_flag	1	uint8	flag	range spread compensation applied
replica_power_corr_flag	0	uint8	flag	Replica pulse power correction applied
abs_calibration_flag	0	uint8	flag	Product calibrated
calibration_factor	470087.71875	float64		Calibration constant
range_sampling_rate	19.20768	float64	MHz	Range Sampling Rate
multilook_flag	0	uint8	flag	Product multilooked
external_calibration_file	ASA_XCA_AXVIEC20060717_154125_20...	ascii		External calibration file used
orbit_state_vector_file	AUX_FRO_AXVPDS20061211_093908_2...	ascii		Orbit file used
⊞ Orbit_State_Vectors				
⊞ SRGR_Coefficients				

GET SOURCE METADATA

- final MetadataElement abs =
  AbstractMetadata.getAbstractedMetadata(sourceProduct);

- rangeSpacing = AbstractMetadata.getAttributeDouble(abs,
  AbstractMetadata.range_spacing);

UPDATE TARGET METADATA

- final MetadataElement absTgt =
  AbstractMetadata.getAbstractedMetadata(targetProduct);

- AbstractMetadata.setAttribute(absTgt,
  AbstractMetadata.multilook_flag, 1);

ADD SELECTED BANDS

- Band targetBand = new Band(targetBandName,
  sourceBand.getDataType(),
  sourceBand.getRasterWidth(),
  sourceBand.getRasterHeight());

- ProductUtils.copyRasterDataNodeProperties(sourceBand, targetBand);

- targetProduct.addBand(targetBand);

TILES

- The memory management of NEST allows very large data products, that can not be all stored in available memory, to be handled by the processing tools using a tiled approach.
- The dataset is divided into workable areas called tiles consisting of a subset of the data read from disk in one piece.
- Depending on the tool, data is ingested for a tile or a set of tiles, and processing is applied only to the current set of tiles. The data is then written to a file and released from memory. The process is then repeated on a new set of tiled data from the large data product.
- Each Operator can have ComputeTile method or a ComputeTileStack method.
- The size of the target tile may be dependent on the requests of other Operators in the graph.
- The size of the source tile requested by your operator could be any size (memory permitting)

COMPUTE TILE

- Getting the source data
- Apply your algorithm
- Saving the target data

```
public void computeTile(Band targetBand, Tile targetTile,
                ProgressMonitor pm) throws OperatorException {

      final Rectangle targetTileRectangle = targetTile.getRectangle();
      final Band srcBand =
               targetBandToSourceBandMap.get(targetBand);
      final Tile sourceRaster = getSourceTile(srcBand,
                               targetTileRectangle, pm);

      final int x0 = targetTileRectangle.x;
      final int y0 = targetTileRectangle.y;
      final int w = targetTileRectangle.width;
      final int h = targetTileRectangle.height;
      for (int y = y0; y < y0 + h; y++) {
         for (int x = x0; x < x0 + w; x++) {
            final GeoCoding geoCoding = sourceProduct.getGeoCoding();
            final GeoPos geoPos = geoCoding.getGeoPos(
                                              new PixelPos(x, y), null);

            final double v = sourceRaster.getSampleDouble(x, y);
            final double v1 = 0.1 * v;
            targetTile.setSample(x, y, v1);
         }
      }
}
```

COMPUTE TILE STACK

- Computes all target bands at once

- Needed only if either:
  - Result of one band is dependant on the result of another
  - For efficiency, something common can be computed once for all bands and then applied to each band

- Generate from @Parameter
  - @Parameter(interval = "[1, *))
    int myNumber;

  - @Parameter(label=“Description"))
    String myDescription;

  - Parameter(valueSet = {ITEM1, ITEM2, ITEM3},
        defaultValue = ITEM2, label=“Selected Item")
    String selectedItem;

MENUS AND ACTIONS

- In the modules.xml file found in the resources folder of the package, add an Action to create a menu item in the DAT. State the class of the Action to be called and the text to show in the menu item.

```
<action>
  <id>SegmentationOp</id>
  <class>org.esa.nest.dat.SegmentationOpAction</class>
  <text>Simple Segmentation</text>
  <shortDescr>This will be shown in the tooltip</shortDescr>
  <parent>sartools</parent>
  <helpId>SegmentationOp</helpId>
</action>
```

- If you created your Operator with the GPF Archetype, this should all be done for you already.

ACTIONS

- The Action class defines what will happen when the menu item is clicked.
- In most cases, it should call the Single Target Product Dialog to run the Operator

- The Action class should extend AbstractVisatAction and override the handler for actionPerformed

```
public class SRGROpAction extends AbstractVisatAction {

    private DefaultSingleTargetProductDialog dialog;

    @Override
    public void actionPerformed(CommandEvent event) {

        if (dialog == null) {
            dialog = new DefaultSingleTargetProductDialog("SRGR",
                getAppContext(), "Slant Range to Ground Range", getHelpId());
                dialog.setTargetProductNameSuffix("_GR");
        }
        dialog.show();

    }
```

PUBLISHING AN OPERATOR

- Operator implementations are published via the Java service provider interface (SPI). A JAR publishes its operators in the resource file META-INF/services/org.esa.beam.framework.gpf.OperatorSpi.
- In this file add your operator SPI eg: org.esa.nest.gpf.MultilookOp$Spi

In your Operator package add a class to extend the OperatorSpi interface. This class may also serve as a factory for new operator instances.

```
public static class Spi extends OperatorSpi {
  public Spi() {
     super(MultilookOp.class);
  }
}
```

- If you created your Operator with the GPF Archetype, this should all be done for you already.

Creating A Reader

READERS

- Readers allow you to interpret a file format and abstract the data into the Generic Product Model

- Readers have two main classes
  - Reader Plugin Class
  - Reader Implementation *Class*

READER PLUGIN

```
public interface ProductReaderPlugIn extends ProductIOPlugIn {

    /**
     * Gets the qualification of the product reader to decode a given input object.
     *
     * @param input the input object
     * @return  the decode qualification
     */
    DecodeQualification getDecodeQualification(Object input);

    /**
     * Returns an array containing the classes that represent valid input types for this reader.
     * <p/>
     * <p> Intances of the classes returned in this array are valid objects for the <code>setInput</code> method of the
     * <code>ProductReader</code> interface (the method will not throw an <code>InvalidArgumentException</code> in this
     * case).
     *
     * @return an array containing valid input types, never <code>null</code>
     */
    Class[] getInputTypes();

    /**
     * Creates an instance of the actual product reader class. This method should never return <code>null</code>.
     *
     * @return a new reader instance, never <code>null</code>
     */
    ProductReader createReaderInstance();
}
```

READER

- The Reader extends AbstractProductReader

- Product readProductNodesImpl()
  - Create a new Product object

- void readBandRasterDataImpl(int sourceOffsetX, int sourceOffsetY,
  int sourceWidth, int sourceHeight, int sourceStepX, int sourceStepY,
  Band destBand, int destOffsetX, int destOffsetY, int destWidth,
  int destHeight, ProductData destBuffer, ProgressMonitor pm)
  - Fill the destination buffer with band data for the requested rectangular area

ADD ABSTRACTED METADATA

- For NEST to use the metadata in a generic way, it must be in the Abstracted Metadata so that each Operator has one interface to get at the data.

- final MetadataElement absRoot = AbstractMetadata.addAbstractedMetadataHeader(root);

- AbstractMetadata.setAttribute(absRoot, AbstractMetadata.PRODUCT, prodName);

- AbstractMetadata.setAttribute(absRoot,
  AbstractMetadata.range_spacing,
  mapProjRec.getAttributeDouble("Nominal inter-pixel distance in output scene"));

PUBLISHING YOUR READER

- Reader implementations are published via the Java service provider interface (SPI).
- A JAR publishes its readers in the resource file META-INF/services/org.esa.beam.framework.dataio.ProductReaderPlugIn.
- In this file add your reader SPI eg: org.esa.nest.dataio.radarsat2.Radarsat2ProductReaderPlugIn

ADDING A MENU ITEM

- In the modules.xml file found in the resources folder of the package, add an Action to create a menu item in the DAT. State the class of the Action to be called and the text to show in the menu item.

```
<action>
   <id>importRadarsat2Product</id>
   <class>org.esa.beam.visat.actions.ProductImportAction</class>
   <formatName>Radarsat 2</formatName>
   <shortDescr>Import a Radarsat2 data product or product subset.</shortDescr>
   <description>Import a Radarsat2 data product or product subset.</description>
   <largeIcon>icons/Import24.gif</largeIcon>
   <placeAfter>importRadarsatProduct</placeAfter>
   <helpId>importRadarsat2Product</helpId>
</action>
```

Creating A Writer

WRITERS

- Writers allow you to write the Product data to some other file format

- Writers have two main classes
  - Writer Plugin Class
  - Writer Implementation Class

WRITER PLUGIN

- public interface ProductWriterPlugIn extends ProductIOPlugIn {

```
    /**
     * Returns an array containing the classes that represent valid output types for this writer.
     * <p/>
     * <p> Instances of the classes returned in this array are valid objects for the <code>setOutput</code> method of the
     * <code>ProductWriter</code> interface (the method will not throw an <code>InvalidArgumentException</code> in this
     * case).
     *
     * @return an array containing valid output types, never <code>null</code>
     *
     * @see ProductWriter#writeProductNodes
     */
    Class[] getOutputTypes();

    /**
     * Creates an instance of the actual product writer class. This method should never return <code>null</code>.
     *
     * @return a new writer instance, never <code>null</code>
     */
    ProductWriter createWriterInstance();
}
```

WRITER

- The Writer extends AbstractProductWriter

- Product writeProductNodesImpl()
  - Write the file header

- void writeBandRasterData(Band sourceBand,
  int sourceOffsetX, int sourceOffsetY, int sourceWidth, int sourceHeight,
  ProductData sourceBuffer, ProgressMonitor pm)
  - Write raster data from the given in-memory source buffer into the files band representation using the given source band and region

PUBLISING YOUR WRITER

- Writer implementations are published via the Java service provider interface (SPI).
- A JAR publishes its writers in the resource file META-INF/services/org.esa.beam.framework.dataio.ProductWriterPlugIn.
- In this file add your writer SPI eg: org.esa.beam.dataio.geotiff.GeoTiffProductWriterPlugIn

ADDING A MENU ITEM

- In the modules.xml file found in the resources folder of the package, add an Action to create a menu item in the DAT. State the class of the Action to be called and the text to show in the menu item.

```
<action>
   <id>exportGeoTIFFProduct</id>
   <class>org.esa.beam.dataio.geotiff.GeoTiffExportAction</class>
   <formatName>GeoTIFF</formatName>
   <useAllFileFilter>true</useAllFileFilter>
   <mnemonic>O</mnemonic>
   <text>Export GeoTIFF Product...</text>
   <shortDescr>Export a GeoTIFF data product or subset.</shortDescr>
   <description>Export a GeoTIFF data product or product subset.</description>
   <helpId>exportGeoTIFFProduct</helpId>
</action>
```

Using Integrated Development Environments (IDE)

OPEN PROJECT

- IDEA
  - mvn idea:idea
  - Open Nest-3C/nest.ipr
  - Right click on any project and select module settings and Project
  - If there is no JDK specified, press New and browse for
    - C:\bin\java\jdk1.6.0_17
- Eclipse
  - mvn eclipse:eclipse
  - click on Main Menu/File/Import
    - Select General/Existing Project into Workspace
    - Browse for source folder
  - Open Window/Preferences..., then select Java/Build Path/Classpath Variables
    - Select New... and add variable M2_REPO
    - Select Folder... and choose the location of your Maven local repository, e.g. ~/.m2/repository or c:\users\username\.m2\repository

SETUP RUN PARAMETERS

Add run-time parameters for IDEA and Eclipse

- Add a new Java Application Run Configuration
- Name: DAT
- Main class: com.bc.ceres.launcher.Launcher
- VM parameters: -Xmx1024M -Dceres.context=nest
- Program parameters: none
- Working directory: $MY_PROJECTS/nest/beam
- Use classpath of module nest-bootstrap
  - Eclipse: click on user entry, add project and select nest-bootstrap
  - Eclipse: add external jars, browse for nest-3C\target\nest-3C-bin-dir\lib and select all jar files

Thank You