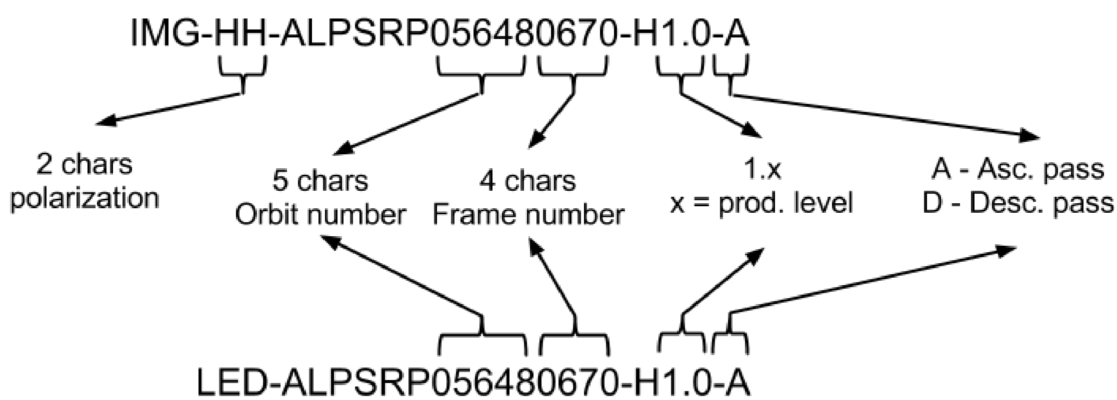
1.Understanding ALOS PALSAR Data Set Names

We'll start with a popular data type: The PALSAR L-band data from Japan's ALOS satellite, which operated between 2006 and 2011. We will describe the names these files are given by the data provider, and how these can be placed in the ISCE input files for processing.

Were you to download PALSAR data from a data provider, each frame comprises an image data file and a image leader file, as well as possibly some other ancillary files that are not used by ISCE. The leader file contains parameters of the sensor that are relevant to the imaging mode, all the information necessary to process the data. The data file contains the raw data samples if Level 1.0 raw data (this is just a different name from what other satellites call Level 0) and processed imagery if Level 1.1 or 1.5 image data. The naming convention for these files is standardized across data archives, and has the following taxonomy:



Files with IMG as prefix are images. Files with LED as prefix are leaders. We will describe how to find and download these data shortly. But first let's see how these filenames are specified in the inputs to ISCE. ISCE at present only supports processing the raw or Level 1.0 PALSAR data.

2. Inserting ALOS PALSAR filenames into the ISCE xml input files

Now it is time to take a look at the input file that we used in lab1 when we ran insarApp.py. First you should change your working directory to the lab1 directory. Recall from lab1 that you use the command `pwd` to see the location of the directory you are currently in and you use the command `cd` to change directories. For your current directory you should see,

```
> pwd
/home/ubuntu
```

Now cd into the lab1 processing directory where you ran the ISCE application insarApp.py.

```
> cd data/lab1/20070215_20061231
```

You can use the `ls` command discussed in lab1 to see the files in this directory. For now we will be concerned only with the three files, `insar_20070215_20061231.xml`, `Master.xml`, and `Slave.xml`.

```
> ls -l insar_20070215_20061231.xml Master.xml Slave.xml
insar_20070215_20061231.xml
Master.xml
Slave.xml
```

Use the Unix command `cat` (for catenate), to see the contents of the input file:

```
> cat insar_20070215_20061231.xml
<insarApp>
<component name="insarApp">
    <property name="sensor name">
        <value>ALOS</value>
    </property>
    <component name="Master">
        <catalog>Master.xml</catalog>
    </component>
    <component name="Slave">
        <catalog>Slave.xml</catalog>
    </component>
</component>
</insarApp>
```

This is an *xml* file. The format of this type of file may seem unfamiliar or strange to you, but with the following description of the basics of the format, it will hopefully become more familiar. The first thing to point out is that the indentations and line breaks seen above are not required and are simply used to make the structure more clear and the file more readable to humans. The xml file provides structure to data for consumption by a computer. As far as the computer is concerned the data structure is equally readable if all of the information were contained on a single very long line, but human readers would have a hard time reading it in that format.

The next thing to point out is the method by which the data are structured through the use of *tags* and *attributes*. An item enclosed in the < (less-than) and > (greater-than) symbols is referred to as a *tag*. The name enclosed in the < and > symbols is the *name* of the *tag*. Every tag in an xml file **must** have an associated closing tag that contains the same name but starts with the symbol </ and ends with the symbol >. This is the basic unit of structure given to the data. Data are enclosed inside of opening and closing tags that have names identifying the enclosed data. This structure is nested to any order of nesting necessary to represent the data. The Python language (in which the ISCE user interface is written) provides powerful tools to parse the xml structure into a data structure object and to very easily "walk" through the structure of that object.

In the above xml file the first and last tags in the file are a tag pair: <insarApp> and </insarApp> (note again, tags **must** come in pairs like this). The first of these two tags, or the *opening tag*, marks the beginning of the contents of the tag and the second of these two tags, or the *closing tag*, marks the end of the contents of the tag. ISCE expects a "file tag" of this nature to bracket all inputs contained in the file. The actual name of the file tag, as far as ISCE is concerned, is user selectable. In this example it is used, as a convenience to the user, to document the ISCE application, named insarApp.py, for which it is meant to provide inputs; it could have been named <foo> and insarApp.py would have been equally happy provided that the closing tag were </foo>.

The next tag is <component name="insarApp">. Its closing tag </component> is located at the penultimate line of the file (one line above the </insarApp> tag). The name of this tag is component and it has an *attribute* called name with value "insarApp". The component tags bound a collection of information that is used by a computational element within ISCE that has the name specified by the name attribute. The name "insarApp" in the first component tag tells ISCE that the enclosed information correspond to a functional component in ISCE named "insarApp", which in this case is actually the application that is run at the command line.

In general, component tags contain information in the form of other component tags or property tags, all of which can be nested to any required level. In this example the insarApp component contains a property tag and two other component tags.

The first tag we see in the insarApp component tag is the property tag with attribute name="sensor name". The property tag contains a value tag that contains the name of the sensor, `ALOS` in this case. The next tag is a `component` tag with attribute `name="Master"`. This tag contains a `catalog` tag containing `Master.xml`. The `catalog` tag in general informs ISCE to look in the named file (`Master.xml` in this case) for the contents of the current tag. The next `component` tag has the same structure with the `catalog` tag containing a different file named `Slave.xml`.

The contents of the `Master.xml` and `Slave.xml` files are the following:

```
> cat Master.xml
<component name="Master">
    <property name="IMAGEFILE">
        <value>../20070215/IMG-HH-ALPSRP056480670-H1.0__A</value>
    </property>
    <property name="LEADERFILE">
        <value>../20070215/LED-ALPSRP056480670-H1.0__A</value>
    </property>
    <property name="OUTPUT">
        <value>20070215.raw</value>
    </property>
</component>

> cat Slave.xml
<component name="Slave">
    <property name="IMAGEFILE">
        <value>../20061231/IMG-HH-ALPSRP049770670-H1.0__A</value>
    </property>
    <property name="LEADERFILE">
        <value>../20061231/LED-ALPSRP049770670-H1.0__A</value>
    </property>
    <property name="OUTPUT">
        <value>20061231.raw</value>
    </property>
</component>
```

The `component` tag that contains the information in each of these files (named “Master” and “Slave”) can be found also in the file `insar_20070215_20061231.xml` surrounding the `<catalog>` entries that specify these filenames. The `Master.xml` and `Slave.xml` files each contain three `property` tags that give the names of the `IMAGEFILE`, `LEADERFILE`, and the `OUTPUT` file. The ALOS PALSAR data are delivered with the `IMAGEFILE` and `LEADERFILE` plus a few other files that are not used by ISCE. You may choose any name you like for the `OUTPUT` filename. The `OUTPUT` filename is the name of the raw file that ISCE creates in its initial steps of processing. In the above example, we have chosen a ROI_PAC style convention of using the date in the format yyyymmdd (year month day). The base of the name you give (the part of the name before the .raw) is also used in the name of the single-look complex files (SLCs) created by ISCE.

The <value> tag for the properties IMAGEFILE and LEADERFILE in Master.xml and Slave.xml contain the symbol / (commonly referred to as slash) in its name, which indicates that these are *paths* in the file system. The <value> tag for the output file does not contain any / symbols, which indicates that the file will be located in the directory from where the processing command is issued, which was the /home/ubuntu/lab1/20070215_20061231 directory in lab1. The paths used in these example files begin with the symbol ../ which indicates that they are *relative paths* from where we are to where the files are located. The other type of path is an *absolute path* and would start with the / symbol without the leading two dots as in the result of the pwd command (see above for example).

To understand how to interpret the relative path consider, for instance, the IMAGEFILE given in the Master.xml file where we find the value,

```
../20070215/IMG-HH-ALPSRP056480670-H1.0__A
```

The ../ part of this name indicates to look one directory above the current directory. Then the 20070215 part indicates to look in the directory 20070215 found relative to there (*i.e.,* the directory 20070215 located in the directory one directory above the current directory). Finally, the IMG-HH-ALPSRP056480670-H1.0__A part names the IMAGEFILE located in that directory. To further help you understand relative paths, try the following commands:

```
> pwd
/home/ubuntu/data/lab1/20070215_20061231
> cd ../
> pwd
/home/ubuntu/data/lab1/
> ls
20061231  20070215  20070215_20061231  DEM
> cd 20070215
> pwd
/home/ubuntu/data/lab1/20070215
> ls
IMG-HH-ALPSRP056480670-H1.0__A  LED-ALPSRP056480670-H1.0__A
```

As you follow these steps you are following the relative path given in the Master.xml file and you see that the IMAGEFILE and LEADERFILE found in that directory are those given in the Master.xml file.

Now use the cd command (in one step) to go back to the processing directory and use the ls command to view the contents of the 20070215 directory without moving to that directory,

```
> cd ../20070215_20061231
> pwd
/home/ubuntu/data/lab1/20070215_20061231
> ls Master.xml
Master.xml
> ls ../20070215
IMG-HH-ALPSRP056480670-H1.0__A  LED-ALPSRP056480670-H1.0__A
```

You can see that the result of this last `ls` command issued from the directory `20070215_20061231` (where `Master.xml` is located) is the same as above where we used the `cd` command to change directories to the `../20070215` directory.

Note, in this example the relative paths involved a single `../` symbol in naming the relative path. A relative path in general may contain any number of `../` symbols and directory names necessary to locate the directory tree where the files are. Each `../` indicates to look one directory above the directory pointed to by any previous chain of `../` symbols. For example,

```
../../dir1/file1
```

points to a file named `file1` located in a directory named `dir1` located two directories *above* the current directory. We say a directory `dir1` is *above* directory `dir2` if `dir1` contains `dir2`, i.e., if the `ls` command used in `dir1` shows `dir2` in its listing of files and directories. Another example indicating a relative path going up and down the directory trees relative to the current directory: the relative path,

```
../../../../dir1/dir2/file1
```

indicates that `file1` is found by going up 4 directories from the current directory and then down from there into `dir1` and then `dir2`.

An alternative to using the relative path would be to use the absolute path, which is the path shown by the `pwd` command above when we changed directories to the `20070215` directory where the `IMAGEFILE` and `LEADERFILE` were found. Using the absolute path, the `IMAGEFILE` tag would look as follows:

```
    <property name="IMAGEFILE">
        <value>
/home/ubuntu/data/lab1/20070215/IMG-HH-ALPSRP056480670-H1.0__A
        </value>
    </property>
```

Remember that the line breaks and indentations in the xml file are not interpreted by the computer and are only used to improve readability for humans. The absolute path method for the `LEADERFILE` would look similar in an obvious way except with the name of the leader file after the final `/` in the path. You are free to choose whether to use absolute paths or relative paths or a combination of both (for whatever reason).

The choice between the use of absolute and relative paths could involve more than a question of style. If you are doing a very small project, such as in this tutorial, then it matters little which you choose. If there ends up being a long chain of `../` symbols to point to the input files, then an absolute path may be more readable. If you are working on a large project involving many processing runs and a complex directory structure, then the use of absolute paths could result in a waste of time and money when the project directory tree is moved within the file system or to another computer and the absolute paths in the input files have to be modified. The benefit of using relative paths is that if an entire project data directory tree were moved from one location to another on the same file system or to another computer, while preserving the internal structure of the data directory tree, then all of the input files that use relative paths that point to paths in the project data directory tree will continue to work without modification. Any input files with absolute paths will have to be modified, which could be a very costly and laborious process.

The ISCE input data in the above example were split between three different files, `insarApp.xml`, `Master.xml`, and `Slave.xml`. An alternative is to use a single file containing all of the needed information as in the following:

```
<insarApp>
<component name="insarApp">
    <property name="sensor name">
        <value>ALOS</value>
    </property>
    <component name="Master">
        <property name="IMAGEFILE">
            <value>../20070215/IMG-HH-ALPSRP056480670-H1.0__A</value>
        </property>
        <property name="LEADERFILE">
            <value>../20070215/LED-ALPSRP056480670-H1.0__A</value>
        </property>
        <property name="OUTPUT">
            <value>20070215.raw</value>
        </property>
    </component>
    <component name="Slave">
        <property name="IMAGEFILE">
            <value>../20061231/IMG-HH-ALPSRP049770670-H1.0__A</value>
```

```
        </property>
        <property name="LEADERFILE">
            <value>../20061231/LED-ALPSRP049770670-H1.0__A</value>
        </property>
        <property name="OUTPUT">
            <value>20061231.raw</value>
        </property>
    </component>
</component>
</insarApp>
```

A final point on relative paths: They are interpreted relative to the current working directory. Thus if you are working in directory A, but you have an xml file in directory B below A that references `../file.dat`, this will resolve to a path *a level above* A, not at level A.

There are many more possible input options for commanding the processing that we will reveal as we go along in these tutorials. In the next step of this tutorial you will pick one of these styles for input files and try processing some ALOS data using ISCE. The details of the different input files for the other types of sensors supported by ISCE can be found at the following links.

3. Processing ALOS PALSAR data with ISCE

It is time to test your understanding of the input files needed to run `insarApp.py` by creating your own input files for a new pair of ALOS PALSAR images. In this exercise, you will create the necessary input files based on the examples provided in Step 2. To create these files you will need to be able to use a text editor on the virtual machine. Many of you are familiar with text editors like “vi” or “emacs” and you are welcome to use them. For those unfamiliar with text editors, the virtual machine instance provides a simple tool call “nano” that has a few basic “control commands” to open and close files, cut and paste text, etc. It is mostly self-explanatory, but you can look at this [tutorial] for more information. So let’s get started. First we need to position ourselves in the directory where these new data reside:

```
> cd
> cd data/lab3
```

The first `cd` command simply sends you back to your home directory. The second positions you at the level where the data for this lab resides. Let’s see what’s in this directory:

```
> ls
alos
```

For the moment, we are interested in ALOS PALSAR, so we will position ourselves there:

```
> cd alos
> ls
20070612  20090802
```

These names are directories containing the ALOS data for two dates, one in 2007 and the other in 2009. We can examine the contents:

```
> ls 20070612
IMG-HH-ALPSRP073630230-H1.0__A  LED-ALPSRP073630230-H1.0__A
IMG-HV-ALPSRP073630230-H1.0__A
> ls 20090802
IMG-HH-ALPSRP187700230-H1.0__A  LED-ALPSRP187700230-H1.0__A
IMG-HV-ALPSRP187700230-H1.0__A
```

Now it is time to create the input files as above. To organize your data, let’s create a new directory where all the results will go:

```
> mkdir 20070612_20090802
> cd 20070612_20090802
```

At this point, you must create the input files. As described above you have a choice to create one input file that contains all information or to spread the information across three files. If you choose to create it all in one input file, start by creating an empty file:

```
> touch insar_allinput.xml
```

The `touch` command simply creates an empty file if that file does not already exist. If it does exist, it simply updates the modification date. If you choose to create three files, start by creating three empty files:

```
> touch insar_input.xml
> touch 20070612.xml
> touch 20090802.xml
```

Whichever style you choose, with the information provided in Step 2 above and armed with your favorite text editor, you should be able to construct your input files with the appropriate information.

Go for it! When you think your input files are ready, you have can either "play it safe" or "play it risky". If you want to play it safe, look at these examples to see what these files should look like. If you want to play it risky, just run the processor script!

```
> insarApp.py insar_allinput.xml
```

or

```
> insarApp.py insar_input.xml
```

Go get another cup of coffee, and come back in about 20 minutes while the processing occurs. If the program terminates unexpectedly because of an input error, compare your files to the examples.

4. Your completed run

After insarApp.py completes, you should see a text message on your screen similar to the following:

```
.
.
.
     runGeocode - Outputs
----------------------------------------------------------------------
-------------------------------
runGeocode.outputs.MINIMUM_GEO_LONGITUDE = 40.3883333333
runGeocode.outputs.MAXIMUM_GEO_LATITUDE = 11.0975
runGeocode.outputs.MAXIMUM_GEO_LONGITUDE = 41.2466666667
runGeocode.outputs.GEO_LENGTH = 2048
runGeocode.outputs.LONGITUDE_SPACING = 0.000833333333333
runGeocode.outputs.LATITUDE_SPACING = -0.000833333333333
runGeocode.outputs.MINIMUM_GEO_LATITUDE = 12.8033333333
runGeocode.outputs.GEO_WIDTH = 1031
######################################################################
###############################
2013-07-10 00:50:24,564 - isce.insar - INFO - Total Time: 709 seconds
```

Note for your run, the date and times will be different, and the `Total Time` may be longer or shorter than 709 seconds, depending on the kind of virtual machine you are running.

Congratulations you have successfully run ISCE for an ALOS data set. You can view the list of output files that were generated by insarApp.py using the ls command. You should see the following list of files:

```
> ls

20070612.xml                                            insar.log
20090802.xml                                            insarProc.xml
azimuthOffset.mht                                       isce.log
azimuthOffset.mht.xml                              lat.rdr
catalog                                                      lon.rdr
dem.crop                                                insarProc.xml
demLat_N11_N14_Lon_E040_E042.dem          rangeOffset.mht.xml
demLat_N11_N14_Lon_E040_E042.dem.wgs84     resampImage.amp
demLat_N11_N14_Lon_E040_E042.dem.wgs84.xml  resampImage.amp.xml
demLat_N11_N14_Lon_E040_E042.dem.xml       resampImage.int
```

```
filt_topophase.flat
resampImage.int.xml
filt_topophase.flat.geo
resampOnlyImage.amp
filt_topophase.flat.geo.xml
resampOnlyImage.int
filt_topophase.flat.xml
resampOnlyImage.int.xml
IMG-HH-ALPSRP073630230-H1.0__A.raw         simamp
IMG-HH-ALPSRP073630230-H1.0__A.raw.aux     topophase.cor
IMG-HH-ALPSRP073630230-H1.0__A.raw.xml     topophase.cor.xml
IMG-HH-ALPSRP073630230-H1.0__A.slc         topophase.flat
IMG-HH-ALPSRP073630230-H1.0__A.slc.xml     topophase.flat.xml
IMG-HH-ALPSRP187700230-H1.0__A.raw         topophase.geo
IMG-HH-ALPSRP187700230-H1.0__A.raw.aux     topophase.geo.xml
IMG-HH-ALPSRP187700230-H1.0__A.raw.xml     topophase.mph
IMG-HH-ALPSRP187700230-H1.0__A.slc         topophase.mph.xml
IMG-HH-ALPSRP187700230-H1.0__A.slc.xml     z.rdr
insar_allinput.xml                         zsch.rdr
insar_input.xml
```

The listing from your processing run may be different from what you see above, as the ISCE is continuously under development, and these labs will use the latest version of the software. However, most should have identical names, and you can using your knowledge of mdx.py from Lab 2 to explore many of these files easily. Similarly, there may be small differences in the displayed images or phase values relative to the examples in these tutorials.

At this point you can continue on to Lab 3.2 to explore in detail the output files you see in the above listing or you can jump ahead to learn about running insarApp on the datasets from the other sensors supported by ISCE in the Labs 4–7.