McIDAS-V Tutorial
An Introduction to Jython Scripting
updated September 2013 (software version 1.4)

McIDAS-V is a free, open source, visualization and data analysis software package that is the next generation in SSEC's 40-year history of sophisticated McIDAS software packages. McIDAS-V displays weather satellite (including hyperspectral) and other geophysical data in 2- and 3-dimensions. McIDAS-V can also analyze and manipulate the data with its powerful mathematical functions. McIDAS-V is built on SSEC's VisAD and Unidata's IDV libraries, and contains "Bridge" software that enables McIDAS-X users to run their commands and tasks in the McIDAS-V environment. The functionality of SSEC's HYDRA software package is also being integrated into McIDAS-V for viewing and analyzing hyperspectral satellite data.

McIDAS-V version 1.2 includes the first release of a suite of fully supported scripting tools. Running scripts with McIDAS-V allows the user to automatically process data and generate displays for web pages and other environments. Scripting in McIDAS-V is provided in Jython. Jython was chosen because it is a common coding language that follows Python syntax and can access Java. The system library of Jython tools is still under development and new tools will be added with future releases of McIDAS-V. You will be notified at the start-up of McIDAS-V when new versions are available on the McIDAS-V webpage - http://www.ssec.wisc.edu/mcidas/software/v/.

If you encounter any errors or would like to request an enhancement, please post questions to the McIDAS-V Support Forums - http://www.ssec.wisc.edu/mcidas/forums/. The forums also provide the opportunity to share information with other users.

This tutorial assumes that you have McIDAS-V installed on your machine, and that you know how to start McIDAS-V. If you cannot start McIDAS-V on your machine, you should follow the instructions in the document entitled McIDAS-V Tutorial – Installation and Introduction. More training materials are available on the McIDAS-V webpage and in the “Getting Started” chapter of the McIDAS-V User's Guide, which is available from the Help menu within McIDAS-V.

Terminology

There are two windows displayed when McIDAS-V first starts, the McIDAS-V Main Display (hereafter Main Display) and the McIDAS-V Data Explorer (hereafter Data Explorer).

The Data Explorer contains three tabs that appear in bold italics throughout this document: Data Sources, Field Selector, and Layer Controls. Data is selected in the Data Sources tab, loaded into the Field Selector, displayed in the Main Display, and output is formatted in the Layer Controls.

Menu trees will be listed as a series (e.g., Edit -> Remove -> All Layers and Data Sources). Mouse clicks will be listed as combinations (e.g., Shift+Left Click+Drag).
Using the Jython Shell

The Jython Shell consists of an output window on top and an input field on the bottom. The user enters Jython into the input field. When the Enter key or "Evaluate" is pressed, the Jython input is evaluated and output is shown in the output window. The Jython Shell is a great tool to begin writing scripts that can be run from the background. When inputting commands, the Jython Shell runs in single or multi-line mode. You can switch modes by using the double down arrows or with the shortcut Ctrl+/. The Evaluate button also has a shortcut Shift+Enter.

1. Using the Jython Shell, create a window with a single panel Map Display.
   
   a. In Main Display, select Tools -> Formulas -> Jython Shell to open the Jython Shell.

   b. In the input field, type:
      
      ```python
      panel = buildWindow()
      Click Evaluate.
      ```

   buildWindow is the function used to create an object that contains an array of panels. This creates a window as you would using the GUI with File -> New Display Window....

2. Now create another window, this time with a Globe Display. Using the same Jython Shell, in the input field, type:

   ```python
   globePanel = buildWindow(height=600, width=600, panelTypes=GLOBE)
   Click Evaluate.
   ```

   You now have two single paneled displays, each of which can be modified.

3. Turn off the wireframe box on the Map Display and then rotate the Globe Display.

   In the input field, type:
   
   ```python
   panel[0].setWireframe(False)
   Click Evaluate.
   ```

   ```python
   In the input field, type:
   
   ```python
   globePanel[0].setAutoRotate(True)
   Click Evaluate.
   ```

setWireframe and setAutoRotate are methods which operate on an object. In these examples, the objects are panel and globePanel.
Basic Jython Terminology

The terminology used by Jython programmers can sometimes be confusing. In the above examples we introduced the terms *function*, *method* and *object*. In most general terms, an object is returned from a function and a method operates on an object and may return a new object.

In steps 1 and 2, the `buildWindow` function is used to create an object, in this case an array of panels. Objects can have one or more attributes and these attributes are defined by a class. In later examples of this tutorial, you will see the importance of knowing these attributes. Methods are used to operate on an object. In step 3, `setWireframe` operates on the panel object by turning off the wireframe box.

It is important to know the input parameters for each of the functions and methods. All of the McIDAS-V Jython functions and methods are documented in the scripting section of the *McIDAS-V User's Guide* - [http://www.ssec.wisc.edu/mcidas/doc/mcv_guide/current/index.php?page=misc/Scripting.html](http://www.ssec.wisc.edu/mcidas/doc/mcv_guide/current/index.php?page=misc/Scripting.html)

Note that when you are scripting in Jython, you are using the Python syntax. The syntax is case sensitive and adheres to strict indentation practices. A good source of information on Python scripting is “Learn Python the Hard Way” - [http://learnpythonthehardway.org/book/](http://learnpythonthehardway.org/book/)

Using the Jython Shell (continued)

4. The Map Display will be used in the remaining examples, so at this time, close out the Globe Display.

5. Change the projection and center point of the display.
   a. In the *input field*, type:
      ```
      panel[0].setProjection('US>States>Midwest>Wisconsin')
      ```
      Click Evaluate.
   b. In the *input field*, type:
      ```
      panel[0].setCenter(43.0,-89.0)
      ```
      Click Evaluate.

   *setProjection* changes the projection of a panel. The syntax for input projection is similar to what you see when you change the projection using the GUI. Note, Jython is a case sensitive language, and you must type things exactly as documented here.
6. Add some annotations to the display.

   a. Click the Expand Input Field icon to the right of the input field, so that you can type multiple lines into the Jython Shell.

   b. Determine the available fonts for your OS. In the input field, type (the 4 spaces before print are necessary):

   ```python
   for fontname in allFontNames()
     print fontname
   ```

   c. Click Evaluate and from the results, pick a font for the next commands. In these examples, SansSerif.bold is used.

   d. In the input field, type:

   ```python
   panel[0].annotate('<b>You Are Here</b>', size=20, font='SansSerif.bold', lat=43.5, lon=-89.2, color='Red')
   ```

   Click Evaluate.

   The bottom left corner of the text is located at the specified latitude/longitude coordinates. Line and element coordinates are also available in annotate. Color can be specified using RGB values, or the color name. html tags can also be used to do things like make the font bold.

   e. In the input field, type:

   ```python
   panel[0].annotate('<b>+</b>', size=20, font='SansSerif.bold', line=200, element=295, color=[1.0,0.0,1.0])
   ```

   Click Evaluate.

   f. When you are through adding annotations to the display, close the window created with buildWindow.

   Creating a Simple Local ADDE Request

   Up until now all of the functions have been customizing panel attributes. McIDAS-V scripting can also make ADDE requests to list and transfer image data. Once data has been transferred, it can be used to create data layers. The next part of this tutorial will access data from the 'Storm of the Century' from 1993.

7. Create local datasets to access the 'Storm of the Century' infrared imagery files on your local machine.

   a. In the input field of the Jython Shell, type:

   ```python
   dataDir = '<local path>/Data/Scripting/blizzard-areas'
   irDataSet = makeLocalADDEEntry(dataset='BLIZZARD', imageType='Meteosat-3', mask=dataDir, format='McIDAS Area', save=True)
   ```
8. **listADDEImages** is a function that creates a list of dictionaries containing information about each available image. Dictionaries will be described in more detail later in this tutorial. Request a listing of all images from the dataset BLIZZARD. In the *input field* of the **Jython Shell**, type:

```python
dirList = listADDEImages(server='localhost', position='ALL', localEntry=irDataSet)
```

9. In step 6, we listed all the available fonts found on your machine. Using the same techniques, list out the directory information for each image. In the *input field* of the **Jython Shell**, type:

```python
for imageDir in dirList:
    print '  
    print 'New image directory %s %s' % (imageDir['sensor-type'], imageDir['nominal-time'])
    print '--------------------------------------------------------------------------------------'
    for key,value in imageDir.iteritems():
        print key,value
```

10. **getADDEImage** is the function used to request imagery from an ADDE server. The inputs to **getADDEImage** are in the form of keyword, value pairs. The dictionaries returned from **listADDEImages** are in this same format and can be used as inputs to **getADDEImage**. Make an ADDE request to get the imagery data from the first keyword parameter pairing returned from **listADDEImages**. In the *input field*, type:

```python
metaData, imageData = getADDEImage(size='ALL', **dirList[1])
```

11. **getADDEImage** returns two objects, a list of metadata and an array of data. Build a new window using **buildWindow** and display the data using **createLayer**. In the *input field*, type:

```python
panel = buildWindow(height=600, width=900, panelTypes=MAP)
dataLayer = panel[0].createLayer('Image Display', imageData)
```

12. Use the method **captureImage** to save the display to a file in the <local path>/McIDAS-V directory. In the *input field*, type:

```python
panel[0].captureImage('<local path>/McIDAS-V/IR-Image.jpg')
```

Because McIDAS-V does a screen capture on some platforms, be sure that the entire window is showing and is not blocked by other windows, or your resulting image will not be complete. After viewing **IR-Image.jpg** in a browser, close the image window.
Creating a Simple Remote ADDE Request

If you do not have internet access to remote servers, continue with next section. The data from the 'Storm of the Century' from 1993 can also be found on the remote server pappy.ssec.wisc.edu.

13. Request a listing of all images from the dataset BLIZZARD found on the server pappy.ssec.wisc.edu. In the input field of the Jython Shell, type:

\[
\text{dirList} = \text{listADDEImages}(\text{server='pappy.ssec.wisc.edu', dataset='BLIZZARD', descriptor='M3-IR', position='ALL'})
\]

14. As was done with the local dataset, directory information for each image can be listed. In the input field, type: (the 4 space indentations are necessary)

\[
\text{for imageDir in dirList:}
\]

\[
\text{print} \ ' \ \ '
\]

\[
\text{print} \ '\text{New image directory %s %s' % (imageDir['sensor-type'], imageDir['nominal-time'])}
\]

\[
\text{print} \ '\text{-----------------------------------------------------------------------------------------------------}'
\]

\[
\text{for key,value in imageDir.iteritems():}
\]

\[
\text{print} \ 	ext{key},\text{value}
\]

The directories returned from a remote listADDEImages request are identical to those of a local ADDE request and can be used as inputs to getADDEImage.

Using Dictionaries and Metadata to Formulate an ADDE Request

Most ADDE requests need many more parameters than the previous example. Specifying long lists of keyword parameters can be cumbersome and create code that is difficult to read. To avoid these problems, you can take advantage of a Python dictionary. Using a Python dictionary, you can specify all of the key:value pairs, or include just a few, and add the extra ones directly to the getADDEImage function call.

The next few steps require a lot of typing. If you'd like, you can cut and paste the lines from the `<local path>/Data/Scripting/ADDE-dictionary.txt` file into the Jython Shell and then skip to step 17. All of the files used in this tutorial are also printed at the end of the document.

15. Earlier in the tutorial, you created a local ADDE dataset for Meteosat-3 dataset for the BLIZZARD case. Use `getLocalADDEEntry` to get the value for `localEntry` and use it to create a dictionary to be use local data with `getADDEImage`.

a. In the input field, type

\[
\text{desc} = \text{getLocalADDEEntry('BLIZZARD','Meteosat-3')}
\]
b. In the input field, type (the 4 space indentation is required):

```python
addeParms = dict(
    server='localhost',
    localEntry=desc,
    size='ALL',
    mag=(1, 1),
    time=('18:00:00', '18:00:00'),
    day=('1993072'),
    unit='BRIT',
)
```

16. Make an ADDE request for infrared data using key:value pairs and a dictionary. The ** before the dictionary tells Python to evaluate the dictionary's contents and include the key:value pairs in getADDEImage. The dictionary must be last in the list. In the input field, type:

```python
irMetadata, irData = getADDEImage(band=8, **addeParms)
```

17. getADDEImage returns two object, the first is a dictionary of keyword=parameter pairs describing the data and the second object is the actual data. The above request was for all the lines and elements (size='ALL'). Creating a window to show the entire image would probably go beyond the extents of your desktop. To avoid this problem, use the metadata to create a window with dimensions of half the number of lines and elements. In the input field, type:

```python
bwLines = irMetadata['lines'] / 2
bwEles = irMetadata['elements'] / 2
panel = buildWindow(height=bwLines, width=bwEles)
```

18. Now create layer objects for the infrared data. Use createLayer with the objects irData. In the input field, type:

```python
irLayer = panel[0].createLayer('Image Display', irData)
```

19. Apply the 'Longwave Infrared Deep Convection' color table to the infrared layer. Since there is a unique name for each color table, the syntax is a little different than used with setProjection, and the entire naming structure is not necessary here. In the input field, type:

```python
irLayer.setEnhancement('Longwave Infrared Deep Convection')
```
20. Using the values from the keywords 'sensor-type' and 'nominal-time' from the metadata object irMetadata, create a string to use with setLayerLabel (remember that the 4 spaces of indentation are mandatory).

   a. In the input field, type:

```
irLabel = '%s %s' % (irMetadata['sensor-type'], irMetadata['nominal-time'])
```

   b. In the input field, type:

```
irLayer.setLayerLabel(label=irLabel, size=16, color='White', font='SansSerif.bold')
```

After checking the new layer label in the Build Window Display, close the window.

### Creating Movies in a McIDAS-V Script

In previous examples, you have created a single image. You can also create movies that contain loops of images. To do this, multiple data requests must be made. The `<local path>/Data/Scripting/movie.py` file is an example script showing the creation of movie loops in McIDAS-V scripting.

In this example, the loop is created by making a call to listADDEImageTimes and multiple calls to getADDEImage. listADDEImageTimes is similar to listADDEImages, but returns a list of dictionaries containing only image days and times. Below is part of the script with some comments (these are not be entered into the Jython Shell)

A python list is used to store data objects and is initialized using the syntax below. As the script loops through getADDEImage calls, the data objects returned are appended to the list. In this script, myLoop is the python list:

```
myLoop= []
```

listADDEImageTimes uses the dictionary parms as its input parameters. The dictionary object dateTimeList is returned and contains keyword/value pair for each day and time.

```
datetimeList=listADDEImageTimes(**parms)
```
The script then loops through all the dictionaries returned from the call to `listADDEImageTimes`. Using a for loop, individual directories, `dateTime`, are extracted from the list dictionaries, `dateTimeList`, which was returned from `listADDEImageTimes`. The loop takes the `time` value out of the `dateTime` dictionary which is used to create a new dictionary that is passed into `getADDEImage`.

```python
for dateTime in dateTimeList:
    imageTime = dateTime['time']

    ADDE_IR_getRequest = dict(
        localEntry=localDataSet,
        day=dateTime['day'],
        time=(imageTime,imageTime),
        band=8,
        unit='BRIT',
        location=(28.5,-75),
        coordinateSystem=LATLON,
        size=(1000,1000)
    )

    IRMetaData,IRData=getADDEImage(**ADDE_IR_getRequest)
```

The data objects returned from `listADDEImageTimes` are added to `myLoop` using the `append` method.

```python
myLoop.append(IRData)
```

Once the loop is completed, a window is built and `myLoop` is used to create an Image Sequence Layer which is then saved as an animated gif.

```python
panel = buildWindow(height=600,width=900)
irLayer=panel[0].createLayer('Image Sequence Display',myLoop)
writeMovie(imageDir+'ir-loop.gif')
```

21. Open a text editor (e.g., gedit, vi, WordPad), and edit the `<local path>/Data/Scripting/movie.py` file to run in your environment.

   a. Find the following line: `myUser='username'`, and change `username` to the name of your user.

   b. Find the line for your current Operating System, uncomment the line and update if necessary.
22. From the **Jython Shell** run `movie.py`. In the **input field**, type:

```python
editFile('<local path>/Data/Scripting/movie.py')
Click Evaluate.
```

23. Evaluate the `movie.py` file by clicking **Evaluate**.

24. Open a browser and view the file '<local path>/McIDAS-V/ir-loop.gif'.

### Applying McIDAS-V Formulas in a Script

Scripts are also useful for applying predefined system or user formulas to data. The `<local path>/Data/Scripting/formula.py` file is an example showing the use of formulas in McIDAS-V scripting. For this part of the tutorial, a dataset has been chosen that with regions of snow, snow-free land, water, water clouds and ice clouds. Using a multi-spectral approach and system formulas, we can see how to classify pixels into categories.

25. Open a text editor (e.g., gedit, vi, WordPad), and edit the file `<local path>/Data/Scripting/formula.py` to run in your environment.

   a. Find the following line: `myUser='username'`, and change `username` to the name of your user.

   b. Find the line for your appropriate OS, uncomment the line and update if necessary.

There are lines that request albedo data from band 1 Visible (0.67 µm) and temperature data from band 2 Near IR (3.9 µm), band 4 IR (11 µm) and band 6 CO₂ (13 µm). Additionally there is a line that subtracts the band 4 IR (11 µm) data from band 2 Near IR (3.9 µm) data and final line to subtracts band 6 CO₂ (13 µm) data from band 4 IR (11.0 µm).

These lines get the meta data and data for the most recent image:

```python
albedoMetaData, albedoData = getADDEImage(band=1, unit='ALB', **parms)
NearIRMetaData, NearIRData = getADDEImage(band=2, unit='TEMP', **parms)
IRMetaData, IRData = getADDEImage(band=4, unit='TEMP', **parms)
CO2MetaData, CO2Data = getADDEImage(band=6, unit='TEMP', **parms)
```

These lines subtract data from two different bands:

```python
NIRsubIR = sub(NIRD ata, IRData)
IRsubCO2IR = sub(IRData, CO2Data)
```

27. Enter the file `formula.py` into the **Jython Shell**.
   
   a. In the *input field*, type:
      ```python
      editFile('<local path>/Data/Scripting/formula.py')
      ```
   
   b. Click *Evaluate*

28. View the different images by using the layer animation tool. From the **Main Display**, select *View -> Displays -> Visibility Animation -> On*.

29. Close the **Build Window** Display.

**Adding a User Function to the Jython Library**

In the previous example, the system function sub() was used to subtract the temperatures of two bands. For this part of the tutorial, we will add our own function to the Jython Library to further classify each pixel. Prior to that, you will import a color enhancement table that assigns a color to each of the classifications.

30. Import the color enhancement file.
   
   a. From the **Main Display**, select *Tools -> Color Tables*.
   
   b. From the **Color Table Editor**, select *File -> Import*.
   
   c. Browse through your directory structure, and select `<local path>/Data/Scripting/classify-pixels.xml`.
   
   d. In the **Category** text box, enter the text **Scripting** and then hit *Enter* (make sure you hit the *Enter key* after entering the text).
   
   e. From the **Color Table Editor**, select *File -> Save*.
   
   f. Close the **Color Table Editor**.
31. Open the Jython Library and add a new function.

   a. In Main Display, select **Tools -> Formulas -> Jython Library**.

   b. Open the **Local Jython** tab and select **User's library**.

   c. Open a text editor (e.g., gedit, vi, WordPad), and edit the file `<local path>/Data/Scripting/classify-pixels-userlib.txt`

   d. Copy the entire contents of this file and paste it into the **User's Library**.

   e. Select **Save**.

32. In the previous part of this tutorial, objects for several bands of data were created and then the sub() function was used to create two new objects. These objects will now be used to create a new object that classifies each pixel and assigns a color to each.

   Make sure your **Jython Shell** is expanded and copy and paste the following lines into the **Jython Shell**.

   ```python
   panel = buildWindow(height=800, width=800, panelTypes=MAP)
   pixelType = pixelClassification(albedoData, NIRsubIR, IRsubCO2)
   productLayer = panel[0].createLayer('Image Display', pixelType)
   productLayer.setLayerLabel(label=' ')
   productLayer.setEnhancement('Classification', range=(0, 60))
   panel[0].setProjection('US>CONUS')
   panel[0].setCenter(43, -95.5, 2.75)
   panel[0].setWireframe(False)
   panel[0].annotate('<b>Snow', lat=43.3, lon=-95.0, size=18, alignment=('right', 'center'), color='Red')
   panel[0].annotate('Ice Cloud <b>', lat=32, lon=-83.5, size=18, alignment=('left', 'center'), color='Red')
   panel[0].annotate('Water Cloud', lat=37.5, lon=-93.0, size=18, alignment=('center', 'center'), color='Red')
   panel[0].annotate('Land', lat=31.5, lon=-100.5, size=18, alignment=('center', 'center'), color='Red')
   panel[0].annotate('Water <b>', lat=28, lon=-95.0, size=18, alignment=('left', 'center'), color='Red')
   ```

   The image shows the results from the function entered into the Jython Library and classification of each pixel using the following color scheme:

   - **Pink**  Snow
   - **Black**  Bare Land
   - **Blue**  Water
   - **Gray**  Water Clouds
   - **Teal**  Ice Clouds
Running Scripts from a Command Prompt

So far in this tutorial, you have been running commands and scripts using the Jython Shell. Scripts can also be run from the command line by adding the flag `script` to the startup script.

33. Open a text editor (e.g., gedit, vi, WordPad), and edit the `<local path>/Data/Scripting/classify-pixels.py` file to run in your environment.
   a. Find the following line: `myUser='username'`, and change `username` to the name of your user.
   b. Find the line for your current Operating System, uncomment the line and update if necessary.
   c. View the `classify-pixels.py` file and see that it contains the exact commands that were run from the Jython Shell in the previous example.

34. Run the McIDAS-V script using the `script` flag.
   a. Open a terminal and change directory to the directory where McIDAS-V is installed.
   b. Run the `classify-pixels.py` script.
      
      For Unix, type:
      
      ```
      ./runMcV script <local path>/Data/Scripting/classify-pixels.py
      ```
      
      For Windows type:
      
      ```
      runMcV.bat script <local path>/Data/Scripting/classify-pixels.py
      ```
      
   c. The progress of the script can be monitored by watching the `mcidasv.log` file in your McIDAS-V directory with the `tail` command.
      
      For Unix, type:
      
      ```
      tail -f <local path>/McIDAS-V/mcidasv.log
      ```
      
      For Windows type:
      
      ```
      tail -f <local path>/McIDAS-V/mcidasv.log
      ```
      
   d. From your browser, view the file `<local path>/McIDAS-V/classify-pixels.jpg` that was created from `classify-pixels.py`. 
Calculating Statistics in a McIDAS-V Script

35. Calculating statistics for data is also important. McIDAS-V uses the VisAD statistics package to calculate statistics. The file `<local path>/Data/Scripting/stats.py` is an example script showing statistics calculations in McIDAS-V scripting.

36. Open a text editor (e.g., gedit, vi, WordPad), and edit the file to run in your environment.

   a. Find the following line: `myUser='username'`, and change `'username'` to the name of your user.

   b. Find the line for your current Operating System, uncomment the line and update if necessary.

   c. View the `stats.py` file.

To calculate statistics on your data, you’ll need to pass the data into the statistics package. To do this, set the output files, loop through the images, pass the data into the statistics package, and output the statistics to a file.

These lines open the files for writing statistics:
```
outputFile = open(fileDir+"stats.txt", "w")
csvFile = open(fileDir+"stats.csv", "w")
```

This line writes a header text file:
```
csvFile.write("Time,latitude,longitude,geometricMean,min,median,max,kurtosis,skewness,stdDev,variance\n")
```

This line defines how to delimit the data going to the csv file:
```
csvData = csv.writer(csvFile, delimiter="",)  
```

This line passes the data from getADDEImage to the statistics package:
```
stats=Statistics(irData)
```

This line writes the statistic to the output text file:
```
outputFile.write(" std dev: %s \n" % (stats.standardDeviation()))
```

This line writes the statistics to the csv file:
```
csvData.writerow([theTime, "43.0", "-89.0", stats.geometricMean(), stats.min(), stats.median(), stats.max(), stats.kurtosis(), stats.numPoints(), stats.skewness(), stats.standardDeviation(), stats.variance()])
```
38. From a terminal in the directory where McIDAS-V is installed, run the `stats.py` script using the `--script` flag.

    For Unix, type: `.runMcV --script <local path>/Data/Scripting/stats.py`
    For Windows, type: `runMcV.bat --script <local path>/Data/Scripting/stats.py`

39. You can use the statistics created by McIDAS-V in other software packages, and you can plot the statistics values on your McIDAS-V images. Using Excel, open the csv file `<local path>/McIDAS-V/stats.csv`, and do something like create a line graph of your statistics. Using your text editor, open the text file `<local path>/McIDAS-V/stats.txt`, and view the file. From your browser, view the file `<local path>/McIDAS-V/stats-image.jpg` that was created from `stats.py`.

### Creating Your Own McIDAS-V Script

40. You now have all the tools necessary to write a script that creates a movie of product images. For this exercise,

   a. import the enhancement table from `<local-path>/Data/Scripting/transparent-albedo.xml`
   b. write a script that does the tasks listed below:
      1. uses the local data files from the GOES-13 'Pixel Classification Dataset' dataset
      2. uses the entire size of the image
      3. creates two lists of data that span 17:45 and 20:45 on day 2011038
         a. first list is images of albedo
            i. data range between 0 and 80
            ii. applies the color enhancement 'Transparent Albedo'
         b. second list is from images created using pixelClassification formula
            i. data range between 10 and 60
            ii. applies the color enhancement 'Classification'
      4. builds a 900x900 size window
      5. creates an Image Sequence Display layer from the albedo data list (do not include a layer label)
      6. overlays an Image Sequence Display layer from the pixelClassification data list
      7. sets the projection to CONUS with a scale factor of 2.75
      8. changes the center point to 43N -95.5W
      9. turns off the wireframe box
     10. adds the annotation '<—Melting Snow'; text is right and center justified at 34.8N and 101W
     11. saves the movie

An example solution is available at `<local path>/Data/Scripting/classify-movie.py`. However, before checking the solution, it is recommended that you try to complete the tasks on your own.
Files Used In This Tutorial

ADDE-dictionary.txt

# This example assumes that the BLIZZARD dataset has been
# defined on your workstation in the local ADDE Data Manager
# <local path>/Scripting/blizzard-areas
#
# Create a dictionary to be used with getADDEImage.
# (remember the 4 space indentation is required)
#
desc=getLocalADDEEntry(dataset='BLIZZARD',imageType='Meteosat-3')

addeParms = dict(
    debug=True,
    server='localhost',
    localEntry=desc,
    size='ALL',
    mag=(1,1),
    time=('18:00:00','18:00:00'),
    day='1993072',
    unit='BRIT',
)

# Make an ADDE request for infrared data using keyword=parameter
# pairs and the dictionary.
#
irMetadata, irData = getADDEImage(band=8,**addeParms)

# The ** before the dictionary tells python to evaluate the contents of the
# dictionary and include the keyword=parameter pairs with the request to
# getADDEImage. Note, the dictionary must be the last parameter specified.
**classify-movie.py**

```python
myUser='username'

# Windows XP
fileDir=('C:\Documents and Settings\'+myUser+'\McIDAS-V\')
dataPath=('C:\Documents and Settings\'+myUser+'\Data\Scripting\classify-areas')

# Windows 7
fileDir=('C:\Users\'+myUser+'\McIDAS-V\')
dataPath=('C:\Users\'+myUser+'\Data\Scripting\classify-areas')

# Unix
fileDir=('~/McIDAS-V/)
dataPath=('~/Data/Scripting/classify-areas')

# OSX
fileDir=('~/Documents/McIDAS-V/)
dataPath=('~/Documents/Data/Scripting/classify-areas')

# This example uses listADDEImageTimes to create a list images.
# The dates and times returned are sent into getADDEImage.
# The bands returned from getADDEImage are then sent to an algorithm for
# processing.

setLogLevel('TRACE')

# Define a local ADDE dataset
localData=makeLocalADDEEntry(dataset='GOES-13', imageType='Pixel Classification Dataset', mask=dataPath, format='McIDAS Area', save=True)
```
# --- Initialize a python list for storing the data for our
# --- image sequence
#
classificationLoop=[]
albedoLoop=[]

day='2011038'
beginTime='17:45'
endTime='20:45'

# --- Define an ADDE request to create a list of images for a range of days
#
ADDE_listRequest = dict(
    localEntry=localData,
    day=day,
    time=(beginTime,endTime),
    position='ALL'
)

# --- Try to get all the directories for the particular day, if it fails,
# --- send a nice message and continue with the next day
#
dateTimeList = listADDEImageTimes(**ADDE_listRequest)

# --- listADDEImages was successful, so now try getADDEImage for each of the
# --- directories returned. There may be occasions when the getADDEImage fails
# --- but we want to continue
#
for dateTime in dateTimeList:
    
    imageTime = dateTime['time']

    ADDE_albedo_getRequest = dict(
        LocalEntry=localData,
        day=day,
        time=(imageTime,imageTime),
        band=1,
        unit='ALB',
        size='ALL',
    )

    albedoMetaData,albedoData=getADDEImage(**ADDE_albedo_getRequest)
ADDE_NearIR_getRequest = dict(
    localEntry=localData,
    day=day,
    time=(imageTime,imageTime),
    band=2,
    unit='TEMP',
    size='ALL',
)
NearIRMetaData,NearIRData=getADDEImage(**ADDE_NearIR_getRequest)

ADDE_IR_getRequest = dict(
    localEntry=localData,
    day=day,
    time=(imageTime,imageTime),
    band=4,
    unit='TEMP',
    size='ALL',
)
IRMetaData,IRData=getADDEImage(**ADDE_IR_getRequest)

ADDE_CO2_getRequest = dict(
    localEntry=localData,
    day=day,
    time=(imageTime,imageTime),
    band=6,
    unit='TEMP',
    size='ALL',
)
CO2MetaData,CO2Data=getADDEImage(**ADDE_CO2_getRequest)

print 'finished get calls for time %s' % imageTime

# --- getADDEImage returned data, so apply algorithm and add it to the loop#
NIRsubIR=sub(NearIRData,IRData)
IRsubCO2=sub(IRData,CO2Data)
pixelType=pixelClassification(albedoData,NIRsubIR,IRsubCO2)
classificationLoop.append(pixelType)
albedoLoop.append(albedoData)

print 'finished data calls'
# --- Build a window
panel=buildWindow(height=800,width=800,panelTypes=MAP)

# --- Create a layer of albedos
albedoLayer = panel[0].createLayer('Image Sequence Display',albedoLoop)
albedoLayer.setLayerLabel(label=IRMetaData['sensor-type'] + ' Pixel Classification %timestamp%')
albedoLayer.setEnhancement('Transparent Albedo',range=(0,80))

# --- Create a layer of pixel types
productLayer = panel[0].createLayer('Image Sequence Display',classificationLoop)
productLayer.setLayerLabel(label=' ')
productLayer.setEnhancement('Classification',range=(0,60))

panel[0].setProjection('US>CONUS')
panel[0].setCenter(43,-95.5,2.75)
panel[0].setWireframe(False)
panel[0].annotate('<b>&lt;</b> - Melting Snow',lat=34.8,lon=-101,size=18,alignment=('right','center'),color='Red')
writeMovie(fileDir+'classify-pixels.gif')
classify-pixels.py

myUser='username'
#
#     Windows XP
# fileDir=('C:\Documents and Settings\'+myUser+'\McIDAS-V\')
dataPath=('C:\Documents and Settings\'+myUser+'\Data\Scripting\classify-areas')
#
#     Windows 7
# fileDir=('C:\Users\'+myUser+'\McIDAS-V\')
dataPath=('C:\Users\'+myUser+'\Data\Scripting\classify-areas')
#
#     Unix
# fileDir=('~/'+myUser+'/McIDAS-V/')
dataPath=('~/'+myUser+'/Data/Scripting/classify-areas')
#
#     OSX
# fileDir=('~/Users/'+myUser+'/Documents/McIDAS-V/')
dataPath=('~/Users/'+myUser+'/Documents/Data/Scripting/classify-areas')
#
# Define a local ADDE dataset
# localData=makeLocalADDEEntry(dataset='GOES13', imageType='Pixel Classification Dataset', mask=dataPath, format='McIDAS Area', save=True)
#
# --- Define date and time
# day='2011038'
time='18:15'
#
# --- Define an ADDE request
# ADDE_Request = dict(
#    localEntry=localData,
#    day=day,
#    time=(time,time),
#    size='ALL'
#)
# --- Request data for each band using getADDEImage
#
albedoMetaData,albedoData=getADDEImage(band=1, unit='ALB', **ADDE_Request)
NearIRMetaData,NearIRData=getADDEImage(band=2, unit='TEMP', **ADDE_Request)
IRMetaData,IRData=getADDEImage(band=4, unit='TEMP', **ADDE_Request)
CO2MetaData,CO2Data=getADDEImage(band=6, unit='TEMP', **ADDE_Request)

# --- Subtract band 4 from band 2 and subtract band 6 from band 4
#
NIRsubIR=sub(NearIRData,IRData)
IRsubCO2=sub(IRData,CO2Data)
classifyData=pixelClassification(albedoData,NIRsubIR,IRsubCO2)

# --- Build a window
#
panel=buildWindow(height=800,width=800,panelTypes=MAP)

# --- Create an image showing classification of each pixel
#
classifyLayer = panel[0].createLayer('Image Display',classifyData)
classifyLayer.setLayerLabel(label=IRMetaData['sensor-type'] + ' Pixel Classification %timestamp%')
classifyLayer.setEnhancement('Classification',range=(0,60))
panel[0].setProjection('US>CONUS')
panel[0].setCenter(43,-95.5,2.75)
panel[0].setWireframe(False)
panel[0].annotate('&lt;Snow',lat=43.3, lon=-95.0,size=18,alignment=('right','center'),color='Red')
panel[0].annotate('Ice Cloud - &lt;b&gt;&lt;/b&gt;',lat=32, lon=-83.5,size=18,alignment=('left','center'),color='Red')
panel[0].annotate('Water Cloud',lat=37.5, lon=-93.0,size=18,alignment=('center','center'),color='Red')
panel[0].annotate('Land',lat=31.5, lon=-100.5,size=18,alignment=('center','center'),color='Red')
panel[0].annotate('Water - &lt;b&gt;&lt;/b&gt;',lat=28, lon=-95.0,size=18,alignment=('left','center'),color='Red')
panel[0].captureImage(fileDir+'classify-pixels.jpg')
classify-pixels-userlib.txt
from decorators import transform_flatfields

@transform_flatfields
def pixelClassification(albedos, temp2sub4, temp4sub6):

    ""
    Input Parameters
    albedo -.63um albedo
    temp2sub4 - 3.9um(temp) - 11.0um(temp)
    temp4sub6 - 11um(temp) - 13.3um(temp)
    ""

    # Create an object for the output array by copying from the first source
    destinationDataset = albedos.clone()

    # This loop steps through times in the dataset - again times are defined
    # through the GUI. The variable Time is not actual value of time (i.e., 14:45)
    # rather and index into the list of times.
    for time in range(albedos.getDomainSet().getLength()):

        # Get the domain size (lines and elements) of albedos, these values will
        # be used when looping through the data
        albedoSample = albedos.getSample(time)
        temp2sub4Sample = temp2sub4.getSample(time)
        temp4sub6Sample = temp4sub6.getSample(time)

        domain = GridUtil.getSpatialDomain(albedoSample)
        [elementSize,lineSize] = domain.getLengths()

        # Define Output Array
        destinationSample= destinationDataset.getSample(time)
        destinationArray = destinationSample.getFloats(0)

        # Setup up arrays from the fieldimpl (VisAD object containing multiple flat fields)
        albedoArray = albedoSample.getFloats(0)
        temp2sub4Array = temp2sub4Sample.getFloats(0)
        temp4sub6Array = temp4sub6Sample.getFloats(0)
# This is the loop for reading/writing and displaying the data
for line in range(lineSize):
    for element in range(elementSize):
        # Set a variable to point to the location in the array to
        # start reading data
        arrayOffset = line*elementSize+element

        # albedoObject is a one dimensional array containing a list
        # of all lines of data
        albedo = albedoArray[0][arrayOffset]
temp24Diff = temp2sub4Array[0][arrayOffset]
temp46Diff = temp4sub6Array[0][arrayOffset]

        # Pixel classification ALGORITHM
        # Water
        if (temp24Diff <= 0.5):
            outputValue = 10.0
        elif ((temp24Diff > 0.5) and (albedo <= 4.0)):
            outputValue = 10.0
        # Snow
        elif (((temp24Diff > 0.5) and (temp24Diff <= 4.0)) and ((albedo > 18.0) and (albedo <= 40.0))):
            outputValue = 20.0
        # Land
        elif (temp24Diff > 0.5 and temp24Diff <= 12.0) and (albedo > 5.0 and albedo <= 18.0):
            outputValue = 30.0

        # The remainder of the pixels are classified as clouds and use the temperature
        # difference between the 11um and 13.3um to distinguish between ice and water
        # Clouds

        # Water Cloud
        elif (temp46Diff >= 11.0):
            outputValue = 40.0
        # Ice Cloud
        else:
            outputValue = 50.0

        # Write the value to the Output Object
        destinationArray[0][arrayOffset] = outputValue
        return destinationDataset
from decorators import transform_flatfields

@transform_flatfields
def pixelClassification(albedos, temp2sub4, temp4sub6):

    
    Input Parameters
        albedo - .63um albedo
        temp2sub4 - 3.9um(temp) - 11.0um(temp)
        temp4sub6 - 11um(temp) - 13.3um(temp)
    
    # Create an object for the output array by copying from the first source
destinationDataset = albedos.clone()

    # This loop steps through times in the dataset - again times are defined
    # through the GUI. The variable Time is not actual value of time (i.e., 14:45)
    # rather and index into the list of times.
    for time in range(albedos.getDomainSet().getLength()):

        # Get the domain size (lines and elements) of albedos, these values will
        # be used when looping through the data
        albedoSample = albedos.getSample(time)
temp2sub4Sample = temp2sub4.getSample(time)
temp4sub6Sample = temp4sub6.getSample(time)

domain = GridUtil.getSpatialDomain(albedoSample)
[elementSize,lineSize] = domain.getLengths()

    # Define Output Array
    destinationSample = destinationDataset.getSample(time)
destinationArray = destinationSample.getFloats(0)

    # Setup up arrays from the fieldimpl (VisAD object containing multiple flat fields)
albedoArray = albedoSample.getFloats(0)
temp2sub4Array = temp2sub4Sample.getFloats(0)
temp4sub6Array = temp4sub6Sample.getFloats(0)

    # This is the loop for reading/writing and displaying the data
    for line in range(lineSize):
        for element in range(elementSize):
# Set a variable to point to the location in the array to
# start reading data
arrayOffset = line*elementSize+element

# albedoObject is a one dimensional array containing a list
# of all lines of data
albedo = albedoArray[0][arrayOffset]
temp24Diff = temp2sub4Array[0][arrayOffset]
temp46Diff = temp4sub6Array[0][arrayOffset]

# Pixel classification ALGORITHM
# Water
if (temp24Diff <= 0.5):
    outputValue = 10.0
elif ((temp24Diff > 0.5) and (albedo <= 4.0)):
    outputValue = 10.0
# Snow
elif (((temp24Diff > 0.5) and (temp24Diff <= 4.0)) and ((albedo > 18.0) and (albedo <= 40.0))):
    outputValue = 20.0
# Land
elif (temp24Diff > 0.5 and temp24Diff <= 12.0) and (albedo > 5.0 and albedo <= 18.0):
    outputValue = 30.0
# The remainder of the pixels are classified as clouds and use the temperature
# difference between the 11um and 13.3um to distinguish between ice and water
# Clouds
# Water Cloud
elif (temp46Diff >= 11.0):
    outputValue = 40.0
# Ice Cloud
else:
    outputValue = 50.0

# Write the value to the Output Object

destinationArray[0][arrayOffset] = outputValue
return destinationArray[0][arrayOffset] = outputValue
Setting up a variable to specify the location of your final images makes your script easier to read and more portable when you share it with other users.

```python
myUser='username'

#     Windows XP example
imageDir=('C:\Documents and Settings\'+myUser+'\McIDAS-V\')
dataDir=('C:\Documents and Settings\'+myUser+'\Data\Scripting\blizzard-areas')

#     Windows 7 example
imageDir=('C:\Users\'+myUser+'\McIDAS-V\')
dataDir=('C:\Users\'+myUser+'\Data\Scripting\blizzard-areas')

#     UNIX example
imageDir=('~/home/'+myUser+'/McIDAS-V/')
dataDir=('~/home/'+myUser+'/Data/Scripting/blizzard-areas')

#     OS X example
imageDir=('~/Users/'/myUser+'/Documents/McIDAS-V/')
dataDir=('~/Users/'/myUser+'/Documents/Data/Scripting/blizzard-areas')

#     Create a dictionary for requesting images
localDataSet = makeLocalADDEEntry(dataset='BLIZZARD', imageType='Meteosat-3', mask=dataDir, format='McIDAS Area', save=True)
```

parms = dict(
    server='localhost',
    localEntry=localDataSet,
    position='ALL'
)
# Initialize a python list
myLoop=[]

# Create a list of all available Images using listADDEImageTimes
dateTimeList=listADDEImageTimes(**parms)

# ---- listADDEImages was successful, so now try getADDEImage for each of the
# ---- directories returned. There may be occasions when the getADDEImage fails
# ---- but we want to continue
for dateTime in dateTimeList:
    imageTime = dateTime['time']
    print dateTime['time']

    ADDE_IR_getRequest = dict(
        localEntry=localDataSet,
        day=dateTime['day'],
        time=(imageTime,imageTime),
        band=8,
        unit='BRIT',
        location=(28.5,-75),
        coordinateSystem=LATLON,
        size=(1000,1000),
    )

    IRMetaData,IRData=getADDEImage(**ADDE_IR_getRequest)
    myLoop.append(add(IRData,IRData))

# Build a window
panel = buildWindow(height=600,width=900)
irLayer=panel[0].createLayer('Image Sequence Display',myLoop)
writeMovie(imageDir+'ir-loop.gif')
```python
# Setting up a variable to specify the location of your final images
# makes your script easier to read and more portable when you share it
# with other users
import csv

myUser="username"

# Windows XP
# fileDir=("C:\\Documents and Settings\\"+myUser+"\\McIDAS-V\\")

# Windows 7 example
# fileDir=('C:\\Users\\'+myUser+'\\McIDAS-V\\')

# Unix
# fileDir=('~/home/"+myUser+'/McIDAS-V/')

# OS X example
# fileDir=('~/Users/"+myUser+'/Documents/McIDAS-V/')

# The easiest way to make an ADDE request is to create a dictionary
# That defines your parameters. Here we have a generic request
# desc = getLocalADDEEntry('GOES-13', 'Pixel Classification Dataset')
adde_parms = dict(
    server='localhost',
    localEntry=desc,
    place=Places.CENTER,
    size=(100,200),
    coordinateSystem=LATLON,
    location=(44.0,-100.0),
    mag=(1, 1),
    unit='TEMP',
)
```

---

McIDAS-V Tutorial – An Introduction to Jython Scripting

September 2013 – McIDAS-V version 1.4
outputFile = open(fileDir+"stats.txt", "w")
csvFile = open(fileDir+"stats.csv", "wb")
csvData = csv.writer(csvFile, delimiter=",")
csvData.writerow(["Time", "latitude", "longitude", "geometricMean", "min", "median", "max", "kurtosis", "numPoints", "skewness", "stdDev", "variance"])

#     Now make the request using the function getADDEImage
#     This returns a data and metadata object
#
for pos in range(-4,1):
    irMetadata,irData = getADDEImage(position=(pos),band=4, **adde_parms)
#
    pass the irData into the Statistics package
#
stats=Statistics(irData)
#
    open a file and write out the statistics data
#
outputFile.write("stat and value for: %s n %s " % irMetadata["nominal-time")
outputFile.write(" geometric mean: %s n " % stats.geometricMean())
outputFile.write(" kurtosis: %s n " % stats.kurtosis())
outputFile.write(" num points: %s n " % stats.numPoints())
outputFile.write(" skewness: %s n " % stats.skewness())
outputFile.write(" std dev: %s n " % stats.standardDeviation())
outputFile.write(" variance: %s n " % stats.variance())
outputFile.write("\n")

#     import the csv library for writing out the
#     statistics values
#
theTime = str(irMetadata["nominal-time"])[11:16]
csvData.writerow([theTime, "44.0", "-100.0", stats.geometricMean(), stats.min(), stats.median(), stats.max(),
stats.kurtosis(),
    stats.numPoints(), stats.skewness(), stats.standardDeviation(), stats.variance()])
csvFile.close()
outputFile.close()
The last section of the script will annotate an image with the information from the statistics package. Now make the request using the function `getADDEImage`. This returns metadata and data object.

```python
irMetadata, irData = getADDEImage(position=-1, band=4, **adde_parms)
```

pass the `irData` into the Statistics package for this image

```python
statsimage = Statistics(irData)
```

Create some strings from the metadata object to be able to annotate our window with the stats values.

```python
min = 'min: %s' % (statsimage.min())
max = 'max: %s' % (statsimage.max())
stddev = 'std dev: %s' % (statsimage.standardDeviation())
geomean = 'geometric mean: %s' % (statsimage.geometricMean())
numpoints = 'num points: %s' % (statsimage.numPoints())
```

Create a string from the metadata object to make it easier to label the image.

```python
irLabel = '%s %s' % (irMetadata['sensor-type'], irMetadata['nominal-time'])
```

Build a window with a single panel

```python
panel = buildWindow(height=600, width=900)
```
Create a layer from the infrared data object

```
irLayer = panel[0].createLayer('Image Display', irData)
```

When changing attributes, some are panel based and others are layer based. In the following steps, they are:

- Change the projection (panel)
- Turn off the wire frame box (panel)
- Change the center point (panel)
- Add the statistics values (panel)
- Add a layer label (layer)
- Save the output file (panel)

```
panel[0].setProjection('US>States>N-Z>South Dakota')
panel[0].setWireframe(False)
panel[0].setCenter(44.5,-100.0, scale=1.0)
panel[0].annotate(min, line=26,element=170, size=18, color='Blue')
panel[0].annotate(max, line=44,element=170, size=18, color='Blue')
panel[0].annotate(stddev, line=62,element=170, size=18, color='Blue')
panel[0].annotate(geomean, line=80,element=170, size=18, color='Blue')
panel[0].annotate(numpoints, line=98,element=170, size=18, color='Blue')
irLayer.setLayerLabel(label=irLabel, size=14)
panel[0].captureImage(fileDir+'stats-image.jpg')
```