McIDAS-V Tutorial

An Introduction to Jython Scripting

updated June 2012 (software version 1.2)

McIDAS-V is a free, open source, visualization and data analysis software package that is the next generation in SSEC's 35-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

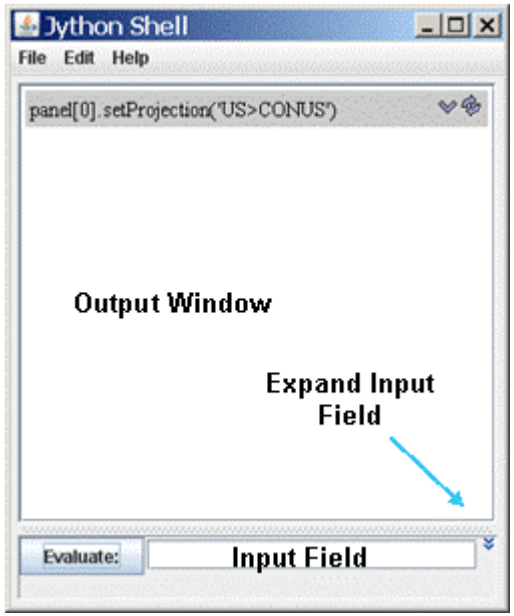
There are two windows displayed when McIDAS-V first starts, the **McIDAS-V Main Display** (hereafter **Main Display**) and the **McIDAS-V Data Explorer** (hereafter **Data Explorer**).

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.



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 *input field*, type:
   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 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:
   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 wire frame box on the Map Display and then rotate the Globe Display.

   a. In *input field*, type:
   panel[0].setWireframe(False)
   Click **Evaluate.**

   b. In *input field*, type:
   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

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/

Using the Jython Shell (continued)

4. For the rest of the examples we will use the Map Display, so at this time, close out the Globe Display.

5. Change the projection and center point of the display.

    a. In *input field*, type:
    panel[0].setProjection('US>States>Midwest>Wisconsin')
    Click **Evaluate**.

    b. In *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 *input field*, type (the 4 spaces before **print** are necessary):
   for fontname in allFontNames():
   ** print fontname**

   c. Click **Evaluate** and from the results in the Output Window, choose a font for the next commands. In these examples, SansSerif is used.

   d. In *input field*, type:
   panel[0].annotate('You Are Here', size=20, font='SansSerif', lat=43.0, lon=-89.0, color='Red')

   The bottom left corner of the text is located at the specified coordinates. In some cases you might want to use a line and element coordinate system. In this example we also specify a color using R,G and B values. Note that we used html tags to make the font bold.

   e. In *input field*, type:
   panel[0].annotate('+', size=20, font='SansSerif', line=225, element=295,color=[1.0,0.0,1.0])

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

Add A Function to the Jython Library

It is also possible to add user functions to the Jython Library that are accessible from the Jython Shell. In this example, you will add a **getDescriptor** function that will be used in all of the following examples of Jython scripting.

7. Using the **Jython Library**, add a getDescriptor function to the User's Local Jython Library.

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

   b. In the left panel, select ***Local Jython->User's library***.

   c. Cut and paste the lines from the <*local path*>**/Scripting/getDescriptor.txt** file into the right panel under **User's library**.

   d. Click **Save**.

Creating A Simple ADDE Request

Up until now we have been concentrating on customizing panel attributes. Next we will request data using ADDE and create data layers. For this part of the tutorial, we will be using data from the 'Storm of the Century' from 1993.

> **Note**: If you have not already done so, create two datasets in the local ADDE Data Manager with BLIZZARD/G7-IR-4K and BLIZZARD/G7-VIS-4K using <*local path*>/**Scripting/blizzard-areas/IR** and <*local path*>/**Scripting/blizzard-areas/VIS**.

getADDEImage is the function used to request imagery from an ADDE server. The inputs to **getADDEImage** are in the form of keyword= parameter. **getADDEImage** returns two objects, a list of metadata and an array of data. The following example makes a simple ADDE request.

8. Build a new window using **buildWindow**. In the *input field*, type: **panel=buildWindow(height=600,width=900,panelTypes=MAP)**

9. Request data using **getADDEImage.** In *input field*, type:
   desc = getDescriptor('BLIZZARD', 'G7-IR-4K')
   myMetaData, myData = getADDEImage(server='localhost:8112', dataset='BLIZZARD', descriptor=desc, unit='BRIT')

10. Display the image by creating a layer. The method **createLayer** uses the object **myData** returned from **getADDEImage**. In *input field*, type:
    dataLayer = panel[0].createLayer('Image Display', myData)

11. Use the method **captureImage** to save the display to a file - replace {user} with your user name. In *input field*, type:
    panel[0].captureImage('/Users/*{user}***/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.

12. After viewing **IR-Image.jpg** in a browser, close the image window.

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 keyword=parameter 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*>**/Scripting/ADDE-dictionary.txt** file into the Jython shell and then skip to step 16. All of the files used in this tutorial are also printed at the end of the document.

13. Create a dictionary to be use local data with getADDEImage. To use data from a remote server, skip to step 15. In *input field*, type (the 4 space indentation is required):

```
addeParms = dict(
   debug=True,
   server='localhost:8112',
   dataset='BLIZZARD',
   size='ALL',
   mag=(1, 1),
   time=('18:01:00', '18:01:00'),
   day=('1993072'),
   unit='BRIT',
)
```

14. Make an ADDE request for infrared and visible data using keyword=parameter pairs and a dictionary. The ** before the dictionary tells Python to evaluate the dictionary's contents and include the keyword=parameter pairs in **getADDEImage**. Note, the dictionary must be last in the list.

    a. In *input field*, type:

```
desc = getDescriptor('BLIZZARD', 'G7-IR-4K')
irMetadata, irData = getADDEImage(descriptor=desc, band=8, **addeParms)
```

    b. In *input field*, type:

```
desc = getDescriptor('BLIZZARD', 'G7-VIS-4K')
visMetadata, visData = getADDEImage(descriptor=desc, band=1, **addeParms)
```

15. To load data from a remote server, follow these steps. If you have already loaded the local data, proceed to the next step. Create a dictionary to be used with getADDEImage. In *input field*, type (the 4 space indentation is required):

```
addeParms = dict(
    debug=True,
    server='pappy.ssec.wisc.edu',
    dataset='BLIZZARD',
    size='ALL',
    mag=(1, 1),
    time=('18:01:00', '18:01:00'),
    day=('1993072'),
    unit='BRIT',
)
```

Make an ADDE request for infrared and visible data using keyword=parameter pairs and the dictionary.

a. In *input field*, type:
irMetadata, irData = getADDEImage(descriptor='G7-IR-4K', band=8, **addeParms)

b. In *input field*, type:
visMetadata, visData = getADDEImage(descriptor='G7-IR-4K', band=1, **addeParms)

16. Earlier in this tutorial we mentioned the importance of knowing the attributes of an object. The metadata object returned from **getADDEImage** contains useful information that we can use with other functions and methods. List out the keyword names and their values. In *input field*, type:

```
for key,value in irMetadata.items():
    print key,' = ',value
```

17. The previous ADDE 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 for half the number of lines and elements. In *input field*, type:

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

18. Now create layer objects for both the visible and infrared data. Using **createLayer** with the objects **irData** and **visData**.

    a. In *input field*, type:
    irLayer = panel[0].createLayer('Image Display', irData)

    b. In *input field*, type:
    visLayer = panel[0].createLayer('Image Display', visData)

19. Previously, we used the method **setProjection** to operate on a panel object. Now we will look at some methods to operate on a layer object. For example, turn off the visible layer. In *input field*, type:
    visLayer.setLayerVisible(False)

20. 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 *input field*, type:
    irLayer.setEnhancement('Longwave Infrared Deep Convection')

21. Using the values from the keywords '**sensor-type**' and '**nominal time**' from the metadata object **irMetadata**, create a string to use with **setLayerLayer** (remember that the 4 spaces of indentation are mandatory).

    a. In *input field*, type:
    ```
    irLabel = '%s %s' % (
       irMetadata['sensor-type'],
       irMetadata['nominal-time']
    )
    ```

    b. In *input field*, type:
    irLayer.setLayerLabel(label=irLabel, size=16, color='White', font='SansSerif.bold')

22. After checking the new layer label in the **Main Display**, exit your McIDAS-V session. The next steps in the tutorial will concentrate on creating and running scripts from a command prompt.

Running Scripts from a Command Prompt

23. Navigate to the file <*local path*>**/Scripting/example.py**.

24. 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 appropriate OS, uncomment the line and update if necessary.

    c. View the **example.py** file and see that it contains the exact commands that were run from the Jython Shell in the previous example.

25. 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 **example.py** script.

       For Unix, type:
       ./runMcV –script <*local path*>**/Scripting/example.py**

       For Windows type:
       runMcV.bat –script <*local path*>**/Scripting/example.py**

    c. From your browser, view the file <*local path*>**/Scripting/IR-Blizzard.jpg** that was created from **example.py**.

Applying Formulas in a McIDAS-V Script

Scripts are also useful for applying formulas to data. The file <*local path*>**/Scripting/formula.py** is an example script showing the use of formulas in McIDAS-V scripting. For this part of the tutorial, we will be using data from the Joplin, Missouri tornado case from 2011.

> **Note**: If you have not already done so, create the Joplin GOES 13 dataset in the local ADDE Data Manager with JOPLIN/GOES13 using <*local path*>/**Scripting/joplin-areas**.

26. 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 appropriate OS, uncomment the line and update if necessary.

    c. View the **formula.py** file. There are five lines that load the data, subtract the IR data from the Water Vapor data, and create the product

    irMetadata, irData= getADDEImage(band=4, **parms)
    - gets the IR meta data and data

    wvMetadata, wvData= getADDEImage(band=3, **parms)
    - gets the Water Vapor meta data and data

    productData= sub(wvData,irData)
    - subtracts the IR data from the Water Vapor data

    import visad.meteorology.NavigatedImage as NavigatedImage
    navigatedProduct= NavigatedImage(productData, wvData.getStartTime(), "wvData minus irData")
    - creates a new product to include the start time and a title

27. 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 **formula.py** script.

    For Unix, type:
    ./runMcV –script <*local path*>**/Scripting/formula.py**

    For Windows type:
    runMcV.bat –script <*local path*>**/Scripting/formula.py**

    c. From your browser, view the file <*local path*>**/Scripting/product-image.gif** that was created from **formula.py**.

Creating Movies in a McIDAS-V Script

28. In previous examples, you have created a single image. You can also create movies that contain loops of images. The file <*local path*>**/Scripting/movie.py** is an example script showing the creation of movie loops in McIDAS-V scripting.

29. 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 appropriate OS, uncomment the line and update if necessary.

    c. View the **movie.py** file. To create a loop of images, load them one at a time with separate getADDEImage calls.

    ```
    myLoop=[ ]
        - initializes the python list, allowing the list to grow with added images

    for pos in range(-4,1):
      irMetadata, myImages=getADDEImage(position=(pos),band=4,unit='BRIT',**parms)
      myLoop.append(myImages)
        - loops through the five dataset positions from -4 to 0.  When the incrementing gets to 1, it exits the loop.

    irLayer=panel[0].createLayer('Image Sequence Display', myLoop)
        - creates a layer as an Image Sequence Display, which creates a loop of images instead of a single image as before with Image Display
    ```

30. Run the McIDAS-V script using the –script flag.

    a. Open a terminal and change directory to the directory where McIDAS-V is installed. Run the **movie.py** script.

    For Unix, type:
    ./runMcV –script <*local path*>**/Scripting/movie.py**

    For Windows type:
    runMcV.bat –script <*local path*>**/Scripting/movie.py**

    b. From your browser, view the file <*local path*>**/Scripting/ir-loop.gif** that was created from **movie.py**.

Calculating Statistics in a McIDAS-V Script

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

32. 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 appropriate OS, 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.

```
outputFile = open(fileDir+"stats.txt", "w")
csvFile = open(fileDir+"stats.csv", "w")
    - open a text file and a csv file

csvFile.write("Time,latitude,longitude,geometricMean,min,median,max,kurtosis,skewness,stdDev,variance\n")
    - writes a header line to the csv file

csvData = csv.writer(csvFile, delimiter=",")
    - defines how to delimit the data going to the csv file

stats=Statistics(irData)
    - passes the data to the statistics package

outputFile.write("   std dev: %s \n" % (stats.standardDeviation()))
    - writes the statistic to the output text 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()])
    - writes the statistics to the csv file
```

33. 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 **stats.py** script.

       For Unix, type:
       ./runMcV –script <*local path*>**/Scripting/stats.py**

       For Windows type:
       runMcV.bat –script <*local path*>**/Scripting/stats.py**

34. You can use the statistics created by McIDAS-V in other software packages, or you can plot the statistics values on your McIDAS-V images.

    a. Using Excel, open the csv file <*local path*>**/Scripting/stats.csv**, and do something like create a line graph of your statistics.

    b. Using your text editor, open the text file <*local path*>**/Scripting/stats.txt**, and view the file.

Creating Your Own McIDAS-V Script

35. You now have all the tools necessary to write a script that creates a movie of product images. For this exercise, create a script that does the tasks listed below:
    a. uses the local ADDE JOPLIN GOES13 data
    b. uses the exact size of the image
    c. creates a movie that spans position numbers -4 to 0
    d. subtracts band 4 temperature from band 3 temperatures
    e. applies the color enhancement '**Longwave Infrared Deep Convection**'
    f. stretches the data range to span from -2 to 2
    g. sets the projection to the US state Missouri
    h. changes the center point to 37N 94.5W
    i. adds a layer label that includes the sensor type and time stamp
    j. saves the movie

An example solution is available at <*local path*>**/Scripting/formula-movie.py**. However, before using it we recommend that you try to complete the tasks on your own.

Files Used In This Tutorial

ADDE-dictionary.txt

```
#
#     Create a dictionary to be used with getADDEImage.
#     (remember the 4 space indentation is required)
#
addeParms = dict(
    debug=True,
    server='localhost:8112',
    dataset='BLIZZARD',
    size='ALL',
    mag=(1, 1),
    time=('18:01:00', '18:01:00'),
    day=('1993072'),
    unit='BRIT',
)
#
#     Make an ADDE request for infrared and visible data using keyword=parameter
#     pairs and the dictionary.
#
#     This assumes that BLIZZARD/G7-IR-4K and BLIZZARD/G7-VIS-4K are already
#      defined on your workstation in the local ADDE Data Manager
#     <local path>/Scripting/blizzard-areas/IR
#        and
#     <local path>/Scripting/blizzard-areas/VIS
#
desc = getDescriptor('BLIZZARD', 'G7-IR-4K')
irMetadata, irData = getADDEImage(descriptor=desc, band=8, **addeParms)
desc = getDescriptor('BLIZZARD', 'G7-VIS-4K')
visMetadata, visData = getADDEImage(descriptor=desc, band=1, **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.
#
```

example.py

```
#    Here is an example of a McIDAS-V script that does the following:
#          sets up parameters for an ADDE request
#          makes an ADDE request
#          creates a window with one panel
#          displays the data
#          changes the projection
#          applies an enhancement table
#          changes the center point
#          adds a layer label
#          annotates the image with an "L" for a Low pressure symbol
#          saves an output file
#     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.
#
myuser='username'
#
#     Windows XP example
#
imageDir=('C:\\Documents and Settings\\'+myuser+'\\McIDAS-V\\')
#
#     Windows 7 example
#
#imageDir=('C:\\Users\\'+myuser+'\\McIDAS-V\\')
#
#     UNIX example
#
#imageDir=('/home/'+myuser+'/McIDAS-V/')
#
#     OS X example
#
#imageDir=('/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:
adde_parms = dict(
    debug=True,
    server='localhost:8112',
    dataset='BLIZZARD',
    size='ALL',
    mag=(1, 1),
    time=('18:01:00', '18:01:00'),
    day=('1993072'),
    unit='BRIT',
)
```

```
#
#     Now make the request using the function getADDEImage.
#     This returns metadata and data objects.
#
desc = getDescriptor('BLIZZARD', 'G7-IR-4K')
ir_metadata,ir_data = getADDEImage(descriptor=desc,
    band=8,
    **adde_parms)
#
#     Create some strings from the metadata object to make it
#     easier to build our window and label the image.
#
bw_lines = ir_metadata['lines']/2
bw_eles = ir_metadata['elements']/2
ir_label = '%s %s' % (
    ir_metadata['sensor-type'],
    ir_metadata['nominal-time']
    )
#
#     Build a window with a single panel
#
panel = buildWindow(height=bw_lines,width=bw_eles)
#
#     Create a layer from the infrared data object
#
ir_layer = panel[0].createLayer('Image Display', ir_data)
#
#     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 an L to pinpoint the Low (panel)
#     Add a layer label (layer)
#     Apply an enhancement (layer)
#     Save the output file (panel)
#
panel[0].setProjection('US>CONUS')
panel[0].setWireframe(False)
panel[0].setCenter(35.5,-75.5, scale=1.0)
panel[0].annotate('<b>L</b>', line=353,element=398, size=24, color='Black')
panel[0].annotate('<b>L</b>', line=351,element=396, size=24, color='Red')
ir_layer.setLayerLabel(label=ir_label, size=14)
ir_layer.setEnhancement('Longwave Infrared Deep Convection')
panel[0].captureImage(imageDir+'IR-Blizzard.jpg')
```

formula.py

```
import visad.meteorology.NavigatedImage as NavigatedImage
#     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
myuser='username'
#
#     Windows XP example
imageDir=('C:\\Documents and Settings\\'+myuser+'\\McIDAS-V\\')
#
#     Windows 7 example
#imageDir=('C:\\Users\\'+myuser+'\\McIDAS-V\\')
#
#     UNIX example
#imageDir=('/home/'+myuser+'/McIDAS-V/')
#
#     OS X example
#imageDir=('/Users/'+myuser+'/Documents/McIDAS-V/')
#
#     Create a dictionary for requesting images
desc = getDescriptor('JOPLIN', 'GOES13')
parms = dict(
    debug=True,
    server='localhost:8112',
    dataset='JOPLIN',
    descriptor=desc,
    coordinateSystem=CoordinateSystems.LATLON,
    location=(37.15,-94.5),
    time=('23:45','23:45'),
    place=Places.CENTER,
    size=(1000, 2000),
    unit='TEMP',
)
irMetadata,irData=getADDEImage(band=4,**parms)
wvMetadata,wvData=getADDEImage(band=3,**parms)
productData=sub(wvData,irData)
navigatedProduct=NavigatedImage(productData, wvData.getStartTime(), "wvData minus irData")
#
#    Build a window
productPanel = buildWindow(height=600,width=900)
productLayer = productPanel[0].createLayer('Image Display', navigatedProduct)
productLayer.setEnhancement('Longwave Infrared Deep Convection', range=(-2,2))
productPanel[0].setProjection('US>States>Midwest>Missouri')
productPanel[0].setCenter(37,-94.5,scale=1.0)
productPanel[0].captureImage(imageDir+'product-image.gif')
```

movie.py

```
#
#     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
myuser='username'
#
#     Windows XP example
imageDir=('C:\\Documents and Settings\\'+myuser+'\\McIDAS-V\\')
#
#     Windows 7 example
#imageDir=('C:\\Users\\'+myuser+'\\McIDAS-V\\')
#
#     UNIX example
#imageDir=('/home/'+myuser+'/McIDAS-V/')
#
#     OS X example
#imageDir=('/Users/'+myuser+'/Documents/McIDAS-V/')
#
#     Create a dictionary for requesting images
desc = getDescriptor('JOPLIN', 'GOES13')
parms = dict(
    debug=True,
    server='localhost:8112',
    dataset='JOPLIN',
    descriptor=desc,
    coordinateSystem=CoordinateSystems.LATLON,
    location=(37.15,-94.5),
    place=Places.CENTER,
    size=(1000, 2000),
)
#
#    Initialize a python list
myLoop=[]
#
#    Loop through 5 images in the Joplin dataset
for pos in range(-4,1):
   irMetadata, myImages=getADDEImage(position=(pos),band=4,unit='BRIT',**parms)
   myLoop.append(myImages)
#
#    Build a window
panel = buildWindow(height=600,width=900)
irLayer=panel[0].createLayer('Image Sequence Display', myLoop)
irLayer.setLayerLabel(label='%displayname% %timestamp%')
writeMovie(imageDir+'ir-loop.gif')
```

stats.py

```
#
#     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="user"

#
#     Windows XP
#
fileDir=("C:\\Documents and Settings\\"+myuser+"\\McIDAS-V\\")
#
#     Unix
#
#fileDir=("/home/"+myuser+"/McIDAS-V/")
#
#     OS X example
#
#imageDir=('/Users/'+myuser+'/Documents/McIDAS-V/')
#
#     The easiest way to make an ADDE request is to create a dictionary
#     That defines your parameters.  Here we a  generic request
#
desc = getDescriptor('JOPLIN', 'GOES13')
adde_parms = dict(
    debug=True,
    server="localhost:8112",
    dataset="JOPLIN",
    descriptor=desc,
    place=Places.CENTER,
    size=(100,200),
    coordinateSystem=CoordinateSystems.LATLON,
    location=(37.0,-94.5),
    mag=(1, 1),
    unit="TEMP",
)

outputFile = open(fileDir+"stats.txt", "w")
csvFile = open(fileDir+"stats.csv", "w")
csvFile.write("Time,latitude,longitude,geometricMean,min,median,max,kurtosis,numPoints,skewness,stdDev,variance\n")
csvData = csv.writer(csvFile, delimiter=",")
```

```
#
#     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" % irMetadata["nominal-time"])
    outputFile.write("   std dev: %s \n" % (stats.standardDeviation()))
    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 cvs library for writing out the
#     statistics values
#
    theTime = str(irMetadata["nominal-time"])[11:16]
    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()])

csvFile.close()
outputFile.close()
```