A Graphical User Interface and Database Management System for Documenting Glacial Landmarks

Whisler, Abbey, Paden, John,
CReSIS, University of Kansas
awhisler08@gmail.com

Abstract

The Landmarks Tool is a new feature in the CReSIS Data Picking Graphical User Interface that allows users to mark landmarks in the echogram window with a rectangle and save the segment ID, GPS times, two way travel times, and description of each feature directly to the CReSIS database. It also allows users to query landmarks from the database using geospatial information. Before this tool was created, recording a landmark required the use of an external piece of software, like a shared spreadsheet.

The first step in building this tool was to add new Django scripts to the CReSIS Open Polar Server that can accept the landmark data and store it in a PostgreSQL table. The second part of the project was to construct a graphical user interface in Matlab that can accept user input and call the Django scripts [4].

Introduction

The Data Picker is an important data processing tool that allows users to explore CReSIS' echogram images in an image browser window and trace the ice surface and ice bottom on the image. The surface and bottom information is added to the PostgreSQL database and becomes accessible across the CReSIS network. The landmarks project enables users to mark glacial features they find in the echogram in a similar fashion and save them to the database.

Open Polar Server, or OPS, is a spatial data infrastructure that allows users to interact with the CReSIS database from custom Matlab data processing tools or the web. The OPS system employs multiple coding languages to accept commands from a user, access and alter the database, and return a result [3].

Python and Django

Django is a Python library built to streamline web development and database management in Python [5]. The first step in the process of building the landmark tool was to create new Django tables to hold the landmarks and landmark classes and add new Django functions to the OPS Django library that the Matlab tool will call to create, delete, and update the landmarks and landmark classes. Django serves as the middleman between Matlab and the PostgreSQL database [3].

I. Setting Up the Virtual Machine

All of CReSIS' Django code is developed and tested on an Oracle Virtual Machine build with a Linux Red Hat 64 bit operating system. CReSIS has a preconfigured virtual machine stored on the server named ops.build.2014.08.19, which comes with some software already downloaded on it, including Django and Python. To access these software, developers start the Python virtual environment from the Linux terminal using the command `source /usr/bin/venv/bin/activate` [2].

The next step is to download a copy of the CReSIS Django Project from the OPS repository using the following commands in the Linux terminal:

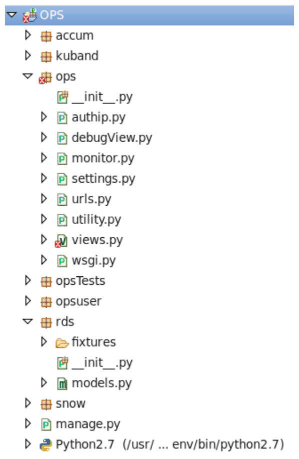
```
cd /vagrant
git pull
sh conf/tools/updateDjangoProject.sh
```

This code adds the OPS Django scripts to the user's personal virtual machine, and the user can manipulate and add to the scripts in this environment without committing any changes to CReSIS' actual database.

Once the Django code is added to the virtual machine, it can be edited directly from the Linux terminal, or developers can use an integrated development software like Eclipse for writing and debugging [3].

II. Editing URLs and Models

The files that were modified for the landmarks project are organized in the OPS Django Project as shown below:



```
OPS
  accum
  kuband
  ops
    __init__.py
    authip.py
    debugView.py
    monitor.py
    settings.py
    urls.py
    utility.py
    views.py
    wsgi.py
  opsTests
  opsuser
  rds
    fixtures
    __init__.py
    models.py
  snow
  manage.py
  Python2.7  (/usr/ ... env/bin/python2.7)
```

Urls.py defines a variable named urlpatterns, which contains the URLs for each view that will be called by the web server [5]. The lines of code that define the new URLs all look something like this one:

```
url(r'^create/landmark$','ops.views.createLandmark')
```

Models.py defines the structure for each table in the PostgreSQL database [5]. In order to add landmarks and landmark classes to the database, the models.py file for the RDS module needed a new table for landmark classes and a new table for landmarks. The following was added to the models file in the RDS module:

```
class landmarks(models.Model):
      segment = models.ForeignKey('segments')
      start_GPS_time =
            models.DecimalField(max_digits=12,decimal_places=11,db_inde
      x
            =True)
      stop_GPS_time =
            models.DecimalField(max_digits=12,decimal_places=11,db_inde
      x
            =True)
      start_twtt =
            models.DecimalField(max_digits=12,decimal_places=11,db_inde
      x
            =True)
      stop_twtt =
            models.DecimalField(max_digits=12,decimal_places=11,db_inde
      x
            =True)
      description =
            models.CharField(max_length=200,blank=True,null=True)
class landmark_classes(models.Model):
      name = models.CharField(max_length = 32)
      description = models.CharField(max_length =
            200,blank=True,null=True
```

The `syncdb` command can be used in Python to save new tables to the database after they have been added to models.py. To modify a table that already exists with Django 1.6.5 or older, developers should use the PostgreSQL command `DROP TABLE` to remove the existing table before using `syncdb` to create the new tables [2].

III. Creating New Views

Views.py contains the Django functions that are used to access, modify, and return data from the PostgreSQL database [5].

The `createLandmark` view accepts the segment ID, GPS times, two way travel times, class, and description for a new landmark as input and stores that information in a temporary CSV file. Then it creates a cursor to interact with the database and uses the cursor to execute a line of SQL code to copy the landmarks from the temporary file to the database. If the function was successful it will return a response back to Matlab with a message saying that the landmark insertion was successful.

The `deleteLandmark` view accepts the landmark ID number as input and deletes that landmark from the database. Just like in `createLandmark`, a cursor executes an SQL statement telling PostgreSQL to delete all landmarks where the landmark_id variable is equal to the landmark ID of the landmark that the user selected. Like `createLandmark`, it then returns a response back to Matlab to indicate if the delete was successful.

The `editLandmark` view accepts a landmark ID number, GPS times, two way travel times, class, and description, then changes the fields for the landmark with the given landmark ID number using a cursor and an SQL statement.

The `getLandmarks` view accepts a segment ID number as input and uses an SQL search for landmarks that fall within this segment in the database. If it finds any, it returns their segment number, GPS times, two way travel times, and description as output and returns a statement back to Matlab indicate the success or failure of the search.

The landmark classes table stores the name and description of each landmark type that users can select when they create a new landmark. The landmark classes views are similar to the landmarks views listed above except they use the fields `landmark_class_id`, `name`, and `description`.

IV. Debugging Views

Although the views are normally run from Matlab, the debugging process takes place in Eclipse. There is a file within the OPS Django Project called debugView.py where developers select the application name and set a JSON string [3]. The JSON string contains the data that Matlab would normally pass into the Django function as `param.properties` if the function were being run from Matlab.Then the variable `viewName` is set equal to the name of the view to be tested and run debugView.py. This allows developers to run through the code line by line and set breakpoints. Most of the code for the views is contained within try and except statements, so if the program encounters an error it will return the error as the variable `e`.

Matlab Picking Tool Graphical User Interface

The next step in building the landmark tool was to create the graphical user interface that will allow users to manipulate the landmarks and landmarks class data easily from the Data Picker. The code for this portion of the project is split up into four different folders in the CReSIS Toolbox.

I. OPS Functions

The following landmark functions are contained in the OPS folder:

```
opsCreateLandmark
opsDeleteLandmark
opsEditLandmark
opsGetLandmarks
opsCreateLandmarkClass
opsDeleteLandmarkClass
opsEditLandmarkClass
opsGetLandmarkClasses
```

Each of these Matlab functions calls its respective landmarks or landmark classes Django view. They all have the same basic layout. First they accept the system name and other parameters as input, then they turn the parameters into a JSON string. This information is sent to the appropriate Django view as a command, then Django accesses the database and returns the response to Matlab, and finally the Matlab function returns this response as output. The OPS functions are called by the echogram window, landmark pick tool, and landmark classes window.

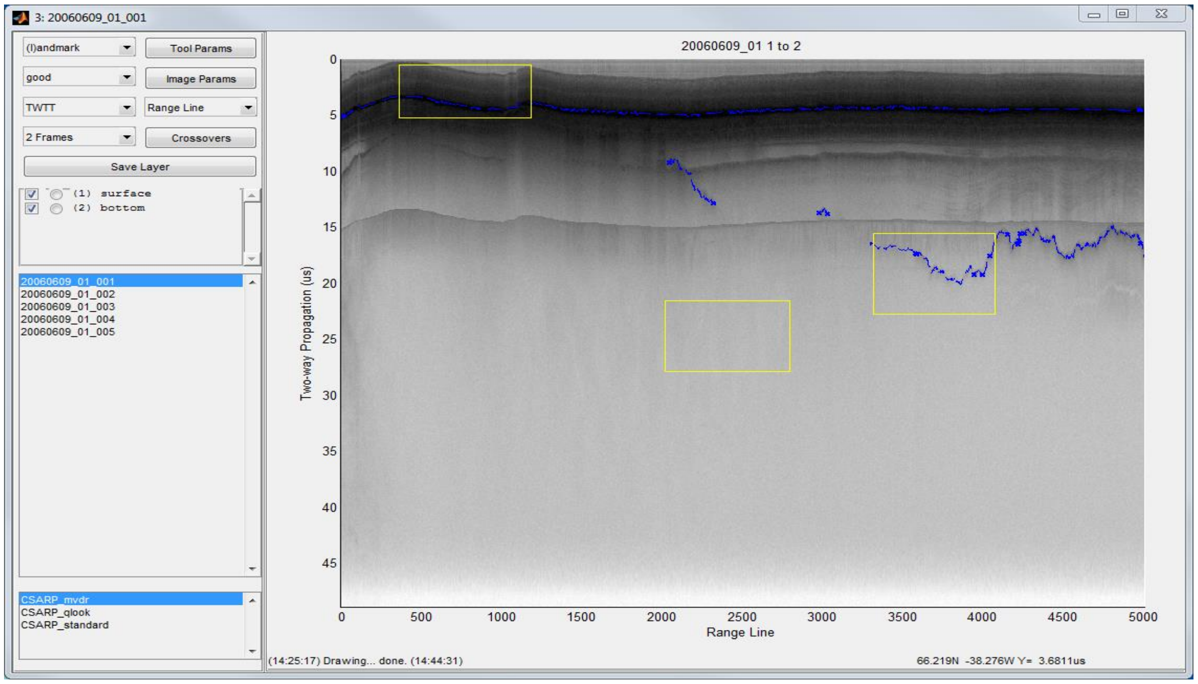
II. Echogram Window

The echowin class folder within the CReSIS Toolbox contains the scripts that run when that window is loaded. One of these is a class definition file named echowin.m that defines the properties, methods, and events used by echowin. The echowin file runs when the user presses the load echogram button. When the echowin file runs it initializes the fields contained in the echowin object, then it calls the echogram create_ui function.

`create_ui` creates the echogram figure window and sets its properties, appearance, callbacks, and listeners. This is where the developer designs the layout of the window, including all of its text, buttons, and pulldown menus [1]. When it is called, the GUI window opens. In the GUI, users can select (l)andmark from the pulldown menu or press the 'l' key on their keyboard. If (l)andmark is selected, and the user clicks the Tool Params button, a window will appear with the parameters for the Landmark Tool.

Within the echogram create_ui function, instances of the picktool_landmark and landmark_classes are created as well. While Matlab makes the echogram GUI it will also make the landmarks GUI and the landmark classes GUI, but they are not visible until the user opens them again by pressing the Tool Params button in the echogram window or the Create Class button in the landmarks window respectively.

Next, the draw function is called, and it calls the load functions, which load data to display in the echogram window. `load_landmarks` uses `opsGetLandmarks` and `opsGetLandmarkClasses` to load in the landmarks, then it calls `set_landmarks` from the picktool_landmark folder, which passes in the landmarks in classes to the picktool_landmark object and prints them in the landmarks window so that the user can select classes and landmarks. Then `load_landmarks` calls `plot_landmarks`, which draws the landmarks from the database on the echogram window as yellow rectangles. Finally, then echogram window will finish loading and the user can draw new landmarks and modify the landmarks stored in the database from the landmarks Tool Params window.

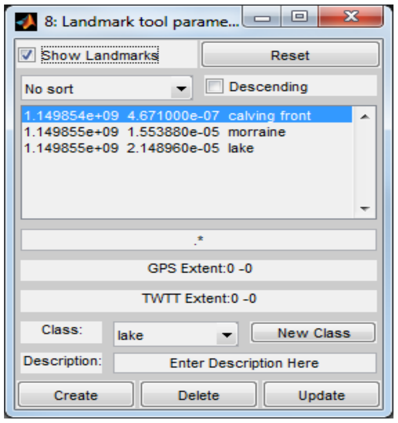


III. Landmark Pick Tool Window

Like the echogram window, the landmark pick tool window folder contains a class definition file, a create UI function, and a load function. It also contains a function called `left_click_and_drag` that defines what will happen when the user presses the alt key and drags the mouse.

When echowin calls the picktool_landmark window, the class definition file initializes the properties, events, and functions for the landmarks tool, and the create UI function for picktool_landmark runs to create the new GUI. Inside the picktool_landmark class definition file all of the functions to create, delete, update, sort, show, and hide landmarks are defined. These functions are called when the user interacts with the corresponding GUI objects.

`left_click_and_drag` tells the program to draw and store the coordinates of a new landmark when a user uses alt-click and drag to draw a rectangle.

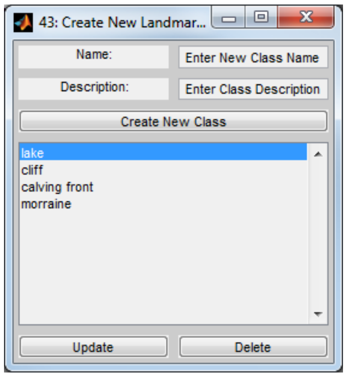


```
1.149854e+09  4.671000e-07  calving front
1.149855e+09  1.553880e-05  morraine
1.149855e+09  2.148960e-05  lake
```

IV. Landmark Classes Window

The landmark class window operates in very much the same way as the landmark pick tool. It also contains a class definition file, a create UI function, and a load function.

The echogram window calls the landmark_class window, then the landmark_class class definition file initializes the properties, events, and functions for the create landmark class window and calls the landmark_class' create UI function to create its GUI. Inside this class definition file, functions to create, delete, and update landmark classes are defined. If the user tries to delete or modify a class that is already in use by existing landmarks in the database the program will throw an error.



Results

There are some logical errors in the Landmarks Tool that have not yet been resolved, but the tool is expected to be completed and committed to the CReSIS server sometime in August of 2015.

When the user loads the echogram window and presses the 'l' key or selects '(l)andmarks' from the pulldown menu they can draw landmarks in the echogram window with a yellow rectangle alt-click and drag. Then they can open the Tool Params window and the GPS times and two way travel times for the new rectangle will be displayed in the window. Then, to save the landmark to the database they must select a class from the pulldown menu and enter a description in the description edit box. Once the class, description, GPS times and two way travel times have been selected, they click the create button to add the new landmark to the database. If they don't enter a class and description, `opsCreateLandmark` will throw an error. To delete a landmark, they click on a landmark in the list of landmarks in the Landmarks Tool Params window and click the delete button. To edit the landmark, the user selects a landmark from the list in the window and then selects whichever class they would like to change it to and types a new description, then clicks 'Update'. Like with 'Create', 'Update' will throw an error if there is not a class and description. When the user runs create, delete, or update, the list of landmarks in the window will refresh to show the landmarks that are currently in the database. When the user presses the New Class button, the Create Classes window appears. In this window there are editable fields for class name and class description. After the user has set a name and description they can click the create button to add the class. To use the delete button they select a class from the list of classes and click 'Delete', and to edit a class they select a class and fill the name and description fields then click 'Update'. Just like in the Pick Tool Landmark window, if the user fails to fill the name or description field when they create or edit a class the program will throw an error. After the user clicks 'Create', 'Delete', or 'Update', the list of landmark classes in the Landmark Class window will be updated.

Conclusions and Future Applications

The Landmark Tool is functional, but it will still need to go through additional revisions before it is be ready for implementation in the OPS system.

This project serves as a good example of how the OPS system as a whole works. Almost forty different Matlab scripts were created or modified in the construction of the GUI, and eight new views were added to the Django Project. The connection between user, user interface, and database is complex. While this project only scratches the surface of that subject, the basic methodology of this project is applicable to other CReSIS GUI projects. Django in particular is difficult to conceptualize, so hopefully this project is useful to programmers working with Django.

Acknowledgement

Many thanks to the Center for the Remote Sensing of Ice Sheets for giving me the opportunity to work in their Research Experience for Undergraduates program, where I had the opportunity to work in the midst of prominent scientists from many different disciplines. Thanks also to the National Science Foundation for providing the funding to support this summer program.

Much of my work was loosely modeled after similar code written by Weibo Liu, Trey Stafford, and Kyle Purdon. Without their previous work and their mentorship, my work this summer would have been infinitely more difficult.

Particular thanks to John Paden, my mentor, for taking me on as his student research assistant and challenging me to meet my full potential in academia. The opportunity he has provided me here at CReSIS to improve my programming and problem solving skills will serve me well in both my academic and professional life in the years to come.

References

[1] GUI Building. (n.d.). Retrieved July 20, 2015, from http://www.mathworks.com/help/matlab/gui-development.html

[2] PostgreSQL Documentation. (n.d.). Retrieved July 22, 2015, from http://www.postgresql.org/docs/

[3] Stafford, T. (n.d.). OPS. Retrieved July 20, 2015, from https://github.com/CReSIS/OPS

[4] Weibo Liu, Kyle Purdon, Trey Stafford and John Paden. (2015, April 21). Development of a Web GIS Application for Cryosphere Community based on Open Source Software Tools. *The Association of American Geographers 111th Annual Meeting*. Lecture conducted from Chicago, Illinois.

[5] Writing Your First Django App, Part 1. (n.d.). Retrieved July 20, 2015, from https://docs.djangoproject.com/en/1.8/intro/tutorial01/