VisTrails in UV-CDAT

Emanuele Santos
Agenda

• Provenance features available in UV-CDAT
• How to include plot types in UV-CDAT
• Scripting support
Preliminaries: Naming conventions

- project
- plot
- visualization
- sheet
- variable
- spreadsheet
- cells
Provenance features in UV-CDAT
Projects

• Files that contain the provenance of the visualizations
  ‣ workflows
  ‣ where in the spreadsheet the visualizations are displayed
  ‣ execution logs
Projects toolbar

- create new project
- open a project
- save the selected project
- close the selected project
Project organization: sheets and visualizations

- A project contains sheets
- A sheet contains visualizations
- When creating a visualization it tells where in the spreadsheet it is located
Project organization: sheets and visualizations

• A project contains sheets
• A sheet contains visualizations
• When creating a visualization it tells where in the spreadsheet it is located
Project organization: sheets and visualizations

• Sheets can be created using the **Create a new sheet** button in the Spreadsheet toolbar.
Project organization: sheets and visualizations

• Sheets can be removed using the **Close Tab** button on the sheet tab
Project organization: sheets and visualizations

- The number of rows and columns can be changed using the spin buttons.
Project organization: sheets and visualizations

• Sheets and visualizations can be named
• To name a sheet, double-click the title of the sheet tab at the top of the spreadsheet
Project organization: sheets and visualizations

• The sheet name will be updated in the projects panel
Project organization: sheets and visualizations

• To name a visualization, double-click the visualization name in the Projects panel.
Project organization: sheets and visualizations

• A new category named visualizations is added to the Projects and will list all named visualizations
Project organization: sheets and visualizations

- You can copy visualizations by dragging them from the Projects panel to the sheet location where they should be displayed.
Project organization: sheets and visualizations

• The Projects panel will be also updated to indicate that the same visualization is also displayed in cell A2.
Editing a visualization: creating overlays in VCS visualizations

• Just drag other plot type into the cell
Editing a visualization: creating overlays in VCS visualizations

• Just drag other plot type into the cell
Editing a visualization: changing visualization parameters

Configure visualization button
Editing a visualization

- Workflow is updated based on the changes
Accessing the provenance of a visualization

view provenance button
Accessing the provenance of a visualization

The VisTrails Builder shows the workflow of the selected visualization
Accessing the provenance of a visualization

The VisTrails Builder shows the workflow of the selected visualization
Accessing the python script that generates a visualization

view source button
Accessing the python script that generates a visualization

• A self-contained script is generated based on the workflow of the visualization
• To execute just call UV-CDAT’s python with the script filename
How to include plot types in UV-CDAT
Workflows, Variables and Plots

• Users are able to create plots in UV-CDAT by dragging variables and plot types to a spreadsheet cell

• Behind the scenes, two separate subworkflows (one for the variables and one for the plot types) are being created and connected to form the workflow for the visualization

• That workflow is then added to the provenance, executed and displayed in the spreadsheet cell
A complete workflow

Variables
subworkflow
Variables and Operations

- Variable
  - PWVariable
  - CDMSVariable
  - CDMSVariableOperation
    - CDMSUnaryVariableOperation
    - CDMSBinaryVariableOperation
    - CDMSNaryVariableOperation
Adding a new plot: Overview

• Create a VisTrails Package
• Create subworkflows for the plot types
• Expose the new plots in the Plots panel
Building a VisTrails package

• The first step is to build a VisTrails Package for the library you want to integrate

• Instructions on how to build a package are available on the VisTrails user's guide (http://www.vistrails.org/usersguide/dev/html/packages.html)

• If you want to seamlessly support the variables loaded in the Variables panel, you need to make your package support the CDMSVariable module from the uvcdat_cdms package

• This will make possible for users to use your package with the already loaded variables

• Add this package to application.py required packages list to make sure it will always be enabled
Subworkflows for the plots

• Every plot type has a corresponding workflow

• There are two ways of doing this:
  ‣ Store the subworkflow (it will be the workflow minus the variables subworkflow) in a vistrail file and load it
    - DV3D follows a strategy similar to this
  ‣ Build the subworkflow dynamically when necessary
    - VCS follows this strategy

• If the subworkflows are simple (3 to 4 modules) we recommend to create them dynamically
Subworkflows for the plots

• At the moment of the workflow creation, the plot type will know in which row and column in the spreadsheet the visualization should be displayed

• The plot type will be also given a subworkflow of Variables

• A complete workflow must be created, including the CellLocation with the position sent and also added to the provenance using the provided API
  
  ‣ All this is done by using a **PipelineHelper** class.
Exposing the Package in the Plots Panel

• UV-CDAT keeps a global registry of plot types that is loaded at startup

• The plot registry is used to populate the Plots panel in the Main Window

• Create a folder in core/uvcdat/plots for your plot type package and add a section to core/uvcdat/plots/registry.cfg file for your package

```plaintext
[PackageName]
codepath = <folder_name>
config_file = <name_of_config_file in folder_name> #usually registry.cfg
helper = <codepath to PipelineHelper class> #example:
          # packages.mypackage.pipeline_helper.MyPipelineHelper
```
Exposing the Package in the Plots Panel

• Inside the folder created for the plot type package, create another registry.cfg file listing all the plot types that should be loaded in the panel

• Write a section for each plot type. Each plot type can have a .vt file (if you decide to store the workflows in a .vt file) and a configuration file
Exposing the Package in the Plots Panel

[global]
cellnum = 1
filenum = 1
varnum = 1
workflow_tag = Simple Plot
dependencies = edu.utah.sci.vistrails.vtk, edu.utah.sci.eranders.ParaView
filename_alias1 = filename
varname_alias1 = varname

[cell1]
celltype = PVCell
row_alias = row
col_alias = col
ProjectController class

• UV-CDAT is based in the concept of Projects

• Every project has its own provenance and its own project controller.

• At any moment in time there is only one active project, and consequently, only one active controller.

• The ProjectController is responsible for the interface between the GUI Actions and the provenance and the plot packages

• It will tell the plot types when and where to build the workflows
  ‣ It will use the pipeline helper of the plot type package the user selected to use
ProjectController Interaction

- DV3D
- VCS
- VisIt
- PipelineHelper
- Project
- ProjectController
- GUI Actions
PipelineHelper class

- Responsible for manipulating the workflows for your plots and to update the provenance information accordingly.

- Base class
  
  `packages.uvcdat_cdms.pipeline_helper.CDMSPipelineHelper`

- In this class, you should reimplement the following static methods:
  - `build_plot_pipeline_action()`
  - `load_pipeline_in_location()`
  - `build_python_script_from_pipeline()`
  - `copy_pipeline_to_other_location()`
  - `show_configuration_widget()`
Changing plot parameters

- Package developers can implement their own widgets to change parameters
  - VCS implements its own widget
  - DV3D also implements its own widgets but they are available in the spreadsheet cell
  - PVClimate uses the default mechanism which is based on aliases
    - Create an alias for each configurable parameter and UV-CDAT will display a widget with all configurable parameters
    - Provenance is also captured automatically by generating changing parameter events
Scripting support
Two types of scripting

GUI Actions

Load variable ts
Load variable TS

Regrid TS according to ts
Subtract ts from the regridded variable
Drag the variable resulted from the subtraction to the spreadsheet cell
Drag Isofill:ASD plot type to the spreadsheet cell

produced workflow

Python script

```python
from api import load_workflow_as_function
proj_file = '/projects/hovmoller.vt'
vis_id = 3
vis = load_workflow_as_function(proj_file, vis_id)
```

```python
cdmsfile = cdms2.open('/data/p101-512.nc')
ts = cdmsfile('ts')
ts = ts(lat=(-89.0, 89.0), squeeze=1, lon=(1.25, 358.75), time=('1-1-16 12:0:0.0', '1-1-16 12:0:0.0'),)

regrid_TS_ts = TS.regrid(ts.getGrid())
sub_regrid_TS_ts_ts = regrid_TS_ts-ts

canvas = vcs.init()
gmIsofill = canvas.getisofill('ASD')
```

fine-grained

cdmsfile = cdms2.open('/data/h0.301-02.nc')
TS = cdmsfile('TS')
TS = TS(lat=(-88.927735352295898, 88.927735352295898), squeeze=1, lon=(0.0, 358.59375), time=('301-3-1 0:0:0.0', '301-3-1 0:0:0.0'),)
regrid_TS_ts = TS.regrid(ts.getGrid())
sub_regrid_TS_ts_ts = regrid_TS_ts-ts
canvas = vcs.init()
gmIsofill = canvas.getisofill('ASD')

coarse-grained
```
Scripting support

• Coarse-grained script is supported by default

• Fine-grained script support
  ‣ Each module needs to implement a `to_python_script()` method
  ‣ `build_python_script_from_pipeline()` needs to be implemented in the PipelineHelper class
    - the default implementation will do a topological sort on the workflow graph and call the `to_python_script()` method of each module