2D Canopy View Architecture

Overview

The 2D Canopy View Plugin API allows programmers to define 2D data-driven visualizations requiring specific data and delivering custom image outputs.

The API

`init()`
Returns an instance of a plugin class.

The following methods should be defined by the plugin class:

`Info()`
Returns a dictionary containing basic information about the plugin. The following keys are recognized:

Key Name	Value Description
name (required)	Visualization name.
description (required)	Visualization description.
unit_label (required)	Describes what type of object is visualized by an image or one group of images. E.g., “stem”
views	The number of views (individual images) generated per unit (stem, site, etc.) Default value is *1*.

`RequiredType()`
Returns 'sql', 'flat', or None depending on required dataset type.

`RequiredTables()`
Returns a list of required tables, or None if expecting a table-less dataset.

`RequiredColumns()`
Returns a list of required columns in the form [(table, column), ...]

`ValidData(dataset)`
Returns True if the provided dataset contains the data necessary to generate this visualization. False otherwise. *See Dataset.py for more information on the Dataset class.*

`GetUnits(dataset)`

Returns a list of ids and labels for units (stems, sites, etc.) from the dataset that can be visualized given the available data. The returned list should be in the form [(id, label), ...] *See Dataset.py for more information on the Dataset class.*

`GetOptions(dataset)`
Returns a list of options to be provided to the end user at run time. The returned list should contain instances of Option classes defined in Plugin.py. The following classes are available. *See Dataset.py for more information on the Dataset class.*

Class / Constructor Arguments	Description
LabelOption(label, [parent])	Static label.
SpacerOption([parent])	Places a bit of visual space; useful for creating groups of options.
TextOption(name, label, [default, [parent]])	Text option.
NumberOption(name, label, [default, [parent]])	Like text option, but with automatic number validation and right-aligned.
SliderOption(name, label, [min, [max, [default, [parent]]]])	Places a slider control with a range defined by min and max.
CheckboxOption(name, label, [default, parent]])	A single checkbox option.
RadioButtonsOption(name, label, options, [default, [parent]])	Radio buttons. The options argument should be a list of strings; one per radio button.
ColorOption(name, label, [default, [parent]])	Color chooser.

`GetUnitName(dataset, options, id, [view=0])`
Returns a filename for the given unit id and view with current dataset and options. This should not include a file extension, which will be added by the exporter. *See Dataset.py for more information on the Dataset class.*

`GetImage(dataset, options, id, [view=0])`
Returns an OmniImage instance representing the given view for the unit specified by id given the provided data and options. The options value is a dictionary with the following format: {name : value, ...}. *See Dataset.py for more information on the Dataset class. See OmniImage.py for more information on the OmniImage class.*

Logging

Plugins can write message to the log file (plugins.log) by using the Log() function. The log function takes a plugin and a message as arguments. E.g., Log(self, “This is an error!”);

Dataset Class

The Dataset class provides a common interface to databases as well as flat data representations. The following methods are available:

`__init__(path)`
Constructor. Accepts an initial file path to load data from.

`Set(path)`
Load data from path.

`GetPath()`
Returns the path to the data file.

`GetType()`
Returns the dataset type. Either ‘sql’ or ‘flat’

`SetTableMappings(mappings)`
Set table mappings as a dictionary in the form: {realTable : [alias1, alias2, ...], ...}

`SetColumnMappings(mappings)`
Set column mappings as a dictionary in the form: {realColumn: [alias1, alias2, ...], ...}

`GetTableMappings()`
Get current table mappings as a dictionary in the form: {realTable : [alias1, alias2, ...], ...}

`GetColumnMappings()`
Get current column mappings as a dictionary in the form: {realColumn: [alias1, alias2, ...], ...}

`AddTableMapping(table, alias)`
Adds an alias for the given table.

`AddColumnMapping(self, column, alias)`
Adds an alias for the given column.

`MapTable(table)`
Returns the real table name for a given alias, or None if none exists.

`MapColumn(column)`
Returns the real column name for a given alias, or None if none exists.

HasTable(table, [map = False])
Returns True if the given table exists. If map is true, will pass the table through the alias map.

HasColumn(table, column, [map = False])
Returns True if the given column exists in the given table. If map is true, will pass both the table and column through the alias map.

GetTable(table, [map = True])
Returns an instance of Result representing the given table in the dataset, or None if the table doesn't exist. If map is true, will pass the table through the alias map.

ExecQuery(query)
Executes a SQL-formatted query on the dataset and returns an instance of Result representing the result set. Only implemented for 'sql' type datasets.

Result Class

Instances of Result are returned by some Dataset methods. The following methods are defined:

CountRows()
Returns the number of rows in the result set.

GetColumns()
Returns columns in the result set.

NextRow()
Returns the next row in the result set as a list or None if no more rows exist.

NextRowD()
Returns the next row in the result set as a dictionary by columns or None if no more rows exist.

OmniImage

The OmniImage class defines a format-independent image. Images constructed using the OmniImage class can be converted to vector PDFs, or bitmap files or a wxDC. The GetImage() method of a plugin class should return an instance of OmniImage. The following methods are available:

__init__([size, [background, [source]]])
Creates a new OmniImage instance. If given a size and background, will create a new image, if given a source, will use that.

Examples:
OmniImage((100,100), (255,255,255)) #Create new image 100px by 100px with a white background.

OmniImage(source="somefile.png") #Load image from a PNG file.
OmniImage("somefile.png") #Shortcut.

Copy()
Create and return a copy of this image.

New(size, [background])
Create a new image with the given size and background color.

Set(source)
Use the given source to define this image.

Load(path)
Load native format from file.

Save(path)
Save native format to file.

Draw(command)
Add a valid OIDrawType instance to the queue. The following classes are available:

OI_Bitmap(bitmap, location, [size, [filter]]) Draws a bitmap at the given location (specified as a 2-tuple). Optional size argument resizes the bitmap, and optional filter argument specifies the filter used to perform the resize operation. Options are kFilterNearest, kFilterBilinear, kFilterBicubic, kFilterAntialias, kFilterFastest, and kFilterBest. Default is kFilterBest.
OI_Line(start, end, [width, color]) Draws a line from start to end (both 2-tuples). Default line width is 1 and default color is (0,0,0).
OI_GradientLine(start, end, [width, [startColor, [endColor]]]) Draws a gradated line from start to end (both 2-tuples). Default line width is 1. Line color is determined according to startColor and endColor.
OI_Rectangle(start, end, [fill, [line, [lineColor]]]) Draws a rectangle with the upper left corner at start and the bottom right corner at end (both are 2-tuples). fill and lineColor default to (0,0,0) and line defaults to 0 (no line).
OI_Ellipse(start, end, [fill, [line, [lineColor]]]) Like OI_Rectangle but defines an ellipse.
OI_Polygon(points, [fill, [line, [lineColor]]]) Draws a polygon defined by points, which is a list of 2-tuples. fill and lineColor default to (0,0,0), line defaults to 0 (no line).

OI_String(location, string, [fontSize, [color, [size, [softWrap]]]])
Draws a string at the given location (specified as a 2-tuple). fontSize defaults to 12, and color defaults to (0,0,0). Optional size argument specifies a target size in pixels (as a 2-tuple) and defaults to None (no resize performed). Optional softWrap argument is not implemented. OI_String provides the method GetStringSize() which returns a 2-tuple representing the size of the string in pixels as it will be drawn.

Erase(command)
Remove a drawing command from the queue by instance or by index.

Resize(size)
Set a new size for this image.

ToBitmapRGB()
Get RGB bitmap data as a string.

ToBitmapRGBA()
Get RGBA bitmap data as a string.

ToWXBitmap()
Get a wx.Bitmap object representing this image.

ToDC()
Draw this image to the given wx.DC instance.

BitmapToFile()
Save this image as a bitmap image file.

VectorToFile(path, [format])
Save this image as a vector image file. (only PDF, currently)

NativeToFile(path)
Save this image in native format.

SetPIL(source)
Create a new image based on a PIL image.

SetPDF(source)
Create a new image based on a PDF image. (Not Implemented)

SetSVG(source)
Create a new image based on an SVG image. (Not Implemented)

SetFile(source)

Create a new image based on a bitmap file.

`SetNative(source)`
Create a new image based on the native format (in string form).

`SetBitmap(source, source)`
Create a new image based on a bitmap. (Not Implemented)

`ToPIL()`
Returns a PIL.Image instance.

`ToSVG()`
Returns an SVG-formatted image. (Not Implemented)

`ToPDF()`
Returns a path to a temporary file containing a PDF-formatted image.

`ToNative()`
Returns a string representation of this image in native format.

Example
See *neighbor.py* in the plugins folder for a fully-comment example plugin.