VAO ds9 wrapper Documentation

Release 0.2

Omar Laurino

July 09, 2013

CONTENTS

vaods9 is a simple wrapper module around the SAMPy package to provide ds9 specific functionality such as `set` and `get` functions.

CHAPTER

ONE

DOCUMENTATION

1.1 Example Usage

This example shows the basic usage of the `vaods9` module.

1.1.1 Startup

first of all, let's import the `vaods9` module:

```
>>> import vaods9
```

let's get an instance of a new **ds9** client (if a SAMP hub is not running, a new one is started):

```
>>> client = vaods9.Client()
```

launch **ds9** now: it should automatically connect to the running hub. While testing the module, you might want to run a graphical SAMP hub monitor like the one shipped with Topcat, just to be sure **ds9** is connected to the hub.

1.1.2 Basic Image Load

let's send a fits image to **ds9**. You can send both relative or absolute paths to a local file, or a URL to a remote file. Please be sure that **ds9** can read the URL (e.g. **ds9** doesn't support https):

```
>>> client.send_fits_image('an_image', 'WFPC2u5780205r_c0fx.fits')
```

The first argument is an arbitrary string defining an ID. I actually think **ds9** ignores it.

1.1.3 XPA like "set" commands

The `vaods9` module allows you to use XPA-like `set` and `get` syntax to interact with **ds9**. The message types enabled are described on the ds9 SAMP website. Now, let's send some `set()`

commands to **ds9**:

```
>>> client.set('cmap Heat')
# change the color map

>>> client.set('zoom to fit')
# zoom to fit the image panel

>>> client.set('scale zscale')
# change scale to zscale

>>> client.set('contour yes')
# draw contours

>>> client.set('contour save ds9.con wcs fk5')
# save contours to file
```

1.1.4 XPA like "get" commands

To retrieve information from **ds9** you have three options.

The first one is the most simple, but the least flexible. You can use this method inside python scripts, when you need the script to wait for the **ds9** response. The `get_now()` function, in fact, let you call **ds9** synchronously: the program waits for **ds9** to respond (at least unless a timeout occurs) and the response is returned to the program directly.

For example:

```
>>> contour_scale = client.get_now('contour scale')
>>> print contour_scale
```

The default timeout is 1000 ms, but you can define a different timeout:

```
>>> contour_scale = client.get_now('contour scale', timeout=500)
```

A `basic_get()` call allows to asynchronously retrieve a value from **ds9**. This can be useful during interactive sessions, in that it doesn't block the python console. You can perform other operations while waiting for a response from **ds9**, and then retrieve the response:

```
>>> client.basic_get('contour scale')
```

The value is stored in the `client.last_response` value:

```
>>> print client.last_response
```

Under the hood, the `vaods9` module is handling the SAMP message reply and it is storing it in the last_response variable.

These two methods might be enough for basic usage.

However, with a slightly more advanced approach, you might write your own handler function, that will be called by the module in the background, as soon as an answer is received from **ds9** (i.e. asynchronously). All that you have to do is to define a function that takes just one argument, like the following:

```
>>> def echo(ds9_response):
>>>     print ds9_response
```

Be sure to exit from the function definition and get the python prompt before continuing.

Now, we will make a *full* `get()` call to **ds9**. This time, we will tell python that we want the echo function to handle the **ds9** response automatically:

```
>>> client.get('contour scale', echo)
# notice that we need just the function name, without quotes.
```

you should see the **ds9** output echoed on your screen.

1.1.5 Cleanup

before closing the session let's do some `cleanup()`:

```
>>> client.cleanup()
```

The client will disconnect from any running hub. If we started a new hub, it will be closed. If you forget to "cleanup" the interactive python session will hang and won't let you return you to the shell prompt.

1.1.6 Gotchas

Here are details on couple of issues you might encounter. See also *Gotchas, Tips and Tricks*.

`set` vs `send_fits_image`

If you have a background in using XPA messaging or if you've been reading the reference materials on the ds9 SAMP website then you might encounter an unexpected behavior when loading images from python using the `set()` instead of the `send_fits_image()` commands.

The unexpected behavior occurs when attempting to load image via `set()` like this:

```
>>> client.set('file WFPC2u5780205r_c0fx.fits')
```

and **ds9** does not react, failing silently. One common cause of this behavior is that the XPA like `set()` assumes that **ds9** is looking in the same local directory as the python command line. This is not a default behavior. One solution is to always pass `set()` filenames with absolute paths. Another simple check and fix for this is to point **ds9** at the local python directory:

```
>>> client.basic_get('cd')
>>> p = os.path.abspath(os.curdir)
>>> if client.last_response != p:
>>>     client.set('cd %s' % p)
```

This `set()` behavior differs from the `send_fits_image` function (or any of the `send_...()` functions) because `send_fits_image` internally converts filenames to absolute paths before broadcasting them as messages to **ds9** [1]. The XPA-like `set()` is too generically defined to do this kind of checking.

1.2 Gotchas, Tips and Tricks

1.3 vaods9 API

[1] The `send_...()` functions broadcast their messages to **all** applications that are listening for those kind of messages.

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*