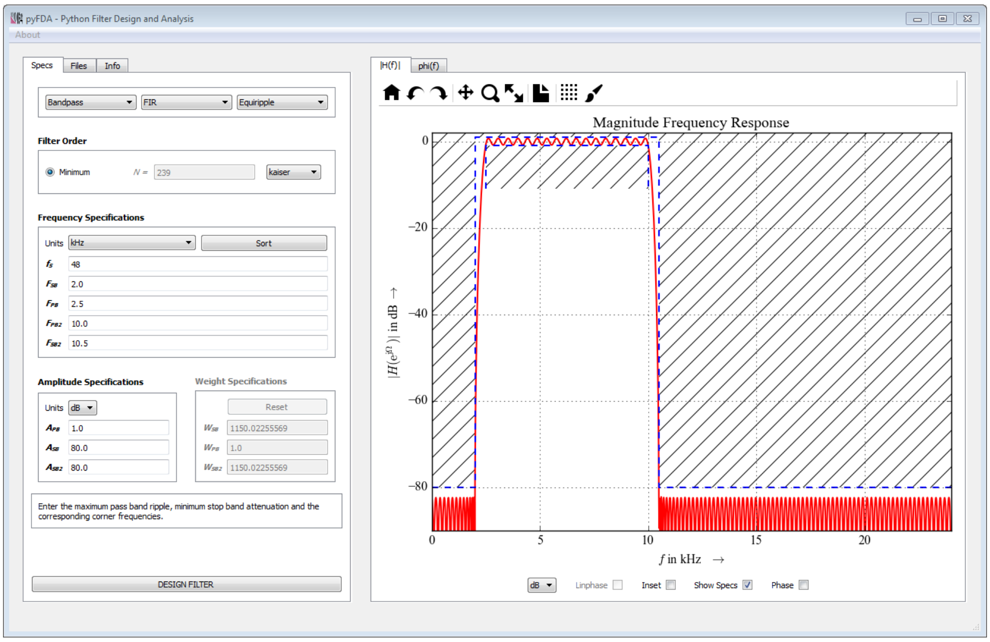
pyFDA: Software Architecture and API

Christian Münker



Overview

15. Mai 2015

Christian.Muenker@hm.edu

Inhaltsverzeichnis

1 Overview

2 Input Widgets

3 Plot Widgets

4 Filter Objects

A new filter objects `my_filter.py` can be added easily to an pyFDA installation by

- copying it to the `filter_widgets` directory
- adding a line with the filename to the list of filter files `Init.txt` in the same directory.

When starting pyFDA, `filter_tree_builder.py` is run, extracting the relevant information and building a hierachical tree in `filter_broker.py`.

The structure of a filter file and the attributes and methods that need to be provided are described in this section.

4.1 Who needs you?

A filter design object is instantiated dynamically every time the filter design method is changed in

`input_widgets/input_filter.py` in **`SelectFilter.setDesignMethod()`**

The handle to this object is stored in `filterbroker.py` in `filObj`.

The actual design methods (LP, HP, ...) are called dynamically in `input_widgets/input_specs.py` in **`InputSpecs.startDesignFilt()`**.

An example for a design method is

```
def LPman(self, fil_dict):
        self.get_params(fil_dict)
        self.save(fil_dict, sig.ellip(self.N, self.A_PB, self.A_SB, self.F_PB,
                btype='low', analog = False, output = frmt))
```

with the single parameter `fil_dict`, that supplies the global filter dictionary containing all parameters and the designed filter as well.

The local helper function `get_params()` extracts parameters from the global filter dictionary and scales the parameters if required (as in the case for ellip routines):

```
def get_params(self, fil_dict):
 """
 Translate parameters from the passed dictionary to instance
 parameters, scaling / transforming them if needed.
 """
 self.N     = fil_dict['N']
 self.F_PB  = fil_dict['F_PB'] * 2 # Frequencies are normalized to f_Nyq
```

The local helper function `save()` saves the filter design back to the dictionary and the filter order and corner frequencies if they have been calculated by a minimum order algorithm.

```
def save(self, fil_dict, arg):
  """
  Store results of filter design in the global filter dictionary. Corner
  frequencies calculated for minimum filter order are also stored in the
  dictionary to allow for a smooth manual filter design.
  """
  pyfda_lib.save_fil(fil_dict, arg, frmt, __name__)

  if self.F_PBC is not None: # has corner frequency been calculated?
    fil_dict['N'] = self.N # yes, update filterbroker
    if np.isscalar(self.F_PBC): # HP or LP - a single corner frequency
      fil_dict['F_PBC'] = self.F_PBC / 2.
    else: # BP or BS - two corner frequencies (BP or BS)
      fil_dict['F_PBC'] = self.F_PBC[0] / 2.
      fil_dict['F_PBC2'] = self.F_PBC[1] / 2.
```

4.2 Info strings

All information that is displayed in `input_widget/input_info.py` in a `QtGui.QTextBrowser()` widget is provided in the multi line strings **`self.info`** and **`self.info_doc`** in MarkDown format. They are analyzed and converted to HTML using `publish_string` from `docutils.core`. `self.info` contains self-written information on the filter design method, `self.info_doc` optionally collects python docstrings. See an excerpt from `ellip.py`:

```
self.info = """
**Elliptic filters**

(also known as Cauer filters) have a constant ripple :math:'A_PB' resp.
:math:'A_SB' in both pass- and stopband(s).

For the filter design, the order :math:'N', minimum stopband attenuation
```

```
:math:‘A_SB‘, the passband ripple :math:‘A_PB‘ and
the critical frequency / frequencies :math:‘F_PB‘ where the gain drops below
:math:‘-A_PB‘ have to be specified.

**Design routines:**

‘‘scipy.signal.ellip()‘‘
‘‘scipy.signal.ellipord()‘‘
"""

self.info_doc = []
self.info_doc.append('ellip()\n========')
self.info_doc.append(sig.ellip.__doc__)
self.info_doc.append('ellipord()\n==========')
self.info_doc.append(ellipord.__doc__)
```