

## wxBrowser and wxwML.

2005-01-22

Johan Lindberg (johan@pulp.se)

Pulp, [www.pulp.se](http://www.pulp.se)

### Introduction

---

In the summer of 2004 I had an interesting conversation with a couple of colleagues at work. The topic was the IBM iSeries' (AS/400) way of handling input/output via the 5250 protocol. We discussed the benefits and drawbacks of web applications and 5250-based applications and I started thinking about how a non-traditional web based application might work<sup>1</sup>.

I decided that this was an interesting enough subject to spend some time on, so I started designing the flow of the system and which parts it would consist of. I decided early on to use Python and wxWidgets because they're both available on more platforms than I care to think about, because I can compile a Windows executable of the application using py2exe and because the wxWidgets library use widgets from the platform it's run on, giving it a native look and feel.

After about a week I had enough sketches and code to call it a project<sup>2</sup> so I named it wxwML (because wxML<sup>3</sup> was taken :- ) which is short for wxWidgets Mark-up Language and wrote a note about it on my web site<sup>4</sup>.

---

<sup>1</sup> The wxBrowser is NOT an implementation of the 5250 protocol. I'm only mentioning it because that's where the idea came from in the first place.

<sup>2</sup> At the time of writing wxBrowser is available at SourceForge as part of the Serf Tools project (<http://sourceforge.net/projects/serf>)

<sup>3</sup> Weather eXchange Mark-up Language, the official Weather Risk Management Association (WRMA) protocol for electronic representation of weather contract details.

<sup>4</sup> <http://www.pulp.se/johan/>

### The wxBrowser

The wxBrowser application and the wxwML definition are the two most important parts of the project. The easiest way to understand the idea behind wxwML is to think of it as HTML for applications. A wxBrowser is to wxwML what a web browser is to HTML, JavaScript and CSS.

The wxBrowser interprets wxwML, translates it into python code and executes it but instead of rendering a "page" for the user to view, it generates a stand-alone<sup>5</sup> desktop application.

You can do most anything in wxwML that can be done with wxPython, there are of course a few limitations. For example, all GUI events are bound to a URL and handled by a predefined handler function, so there's really no way to perform the equivalent of client-side scripting<sup>6</sup>.

### An overview of the application lifecycle

When the wxBrowser is started it is "empty". To load an application you need to open it. Opening an application is very similar to opening a web page in a browser: you enter the URL and press OK (or presses <ENTER>). You can also supply the URL as a command line argument but if you do the wxBrowser won't be visible unless you specify the -v or --visible flag.

After an application has been loaded<sup>7</sup>, the wxBrowser waits for user input. If the wxwML document has specified that an event should be monitored, an HTTP request will be sent to the server when it occurs. The wxwML document specifies a URL and which parts of the GUI to be sent as arguments (via HTTP POST) to the server when the event occurs<sup>8</sup>.

When an event occurs, the wxBrowser sends an HTTP request and expects a wxwML document as a response. All of the elements in the document are converted to instructions that are executed within try/except blocks (any errors that occur are ignored). When the whole document has been processed the browser goes back to waiting for an event to happen. This process is repeated until you close the application frame<sup>9</sup>.

---

<sup>5</sup> Actually it generates a wx.Frame that acts as a stand-alone desktop application but the user will probably not tell the difference.

<sup>6</sup> This may change in future implementations.

<sup>7</sup> The wxwML application will have to be setup before it can be used. It's very important that the first document that is loaded is a document that creates a Frame of some sort containing all widgets that make up the application. Unlike a regular web application (based on HTML, JavaScript and CSS) the wxwML document doesn't have to specify any elements other than those it wishes to update, delete or add to the application. This means that the server has to keep track of whether or not the user already has a session or not, because if it doesn't, it must be redirected to the init page.

<sup>8</sup> It's probably NOT a good idea to handle events that occur frequently, such as mouse events or text input events, even though you can.

<sup>9</sup> Closing the web application's main frame also exits the wxBrowser application, unless the wxBrowser frame is visible.

## Deconstructing the wxBrowser

---

The wxBrowser contain four parts: an HTTP client, a wxwMLParser, a sandbox for running the application and a main frame, which contains information about the current session.

### The main frame

The main frame is mostly a frame that holds three different logs: an error log, a transaction log and a code log. Unless you start the wxBrowser with the `-v` or `--visible` option the main frame will not be visible. Double clicking the wxBrowser icon in the task bar toggles visibility for the wxBrowser main frame.

The wxBrowser's error log contains all exceptions that have occurred. Exceptions are printed with a trace back showing the line where the exception occurred. The transaction log shows the HTTP requests that have been sent. The code log shows all of the python code that has been executed so far. These logs are very useful when debugging, but a normal user probably don't want to see them.

### HTTP Client

The wxBrowser is very similar to a web browser in all aspects regarding communication. All traffic is sent using TCP/IP and the HTTP protocol. The wxBrowser's HTTP client handles coding and decoding of data as well as firewalls, proxy servers and HTTPS connections<sup>10</sup>.

### wxSandbox

The wxSandobx is a run time environment, within the wxBrowser, where the wx module has been imported, but nothing else. All of the code that is generated and executed is run in a try/except block in this environment. If an exception occurs its logged but otherwise ignored.

The sandbox is NOT a general-purpose restricted python environment. It's just set up in such a way that there's no way to access or execute anything outside the wx package.

---

<sup>10</sup> See future work; at the time of writing HTTPS and error handling have not been implemented. Much of the proxy stuff is still to be tested and finalized.

**wxwMLParser**

When a document is requested from a server and sent back to the wxBrowser it is sent to the wxwMLParser that processes it in three steps.

1) The DOM tree is traversed looking for any instances of elements with the prefix wx.

When it finds one it generates the code necessary to create that object, using the argument "name" to bind a reference to it (if no name is specified it defaults to Object1..n).

2) Once all wx widgets have been created and there are references to them in our environment the DOM tree is traversed again but this time looking for parent/child pairs that have, what I call, "convenience functions" defined.

3) After that, all specified function calls are handled and converted into python code, which is executed.

4) As a last step all frames have their Fit() and Show() methods called.

**Explaining wxwML**

---

The wxwML is defined such that all elements are either functions or objects.

**Hierarchy**

All objects are classified after their features, which decide how the creation of the object will take place and whether or not it can appear in certain contexts, or not. One such feature is whether or not the object is a container, whether or not it can be a parent etc. A wx.Panel is a container that can have children, a wx.BoxSizer is a container which cannot act as a parent. Other objects, such as wx.Colour, does not have a direct representation in the GUI. This hierarchy is at the moment defined within the wxBrowser but can hopefully be removed in future versions. See section Future work below.

A function is defined using either call or un/bind where call is a general-purpose function call element and un/bind is used to specify the un/binding of an event to a specific URL. Anything that uses the xml namespace prefix wx will be treated as an object. All objects are created and bound to a variable in the environment as follows:

The wxwML:

```
<wx:Object arg1="value1" ... argn="valuen" name="myobject">
  ...
</wx:Object>
```

is executed in the first step as:

```
myobject= wx.Object(arg1= value1, ..., argn="valuen")11
```

the second step (convenience functions) depends on what type of object it is.

The wxwML function call:

```
<call function="SetSelection" object="myobject" argument="0" />
```

is executed as:

```
myobject.SetSelection(0)
```

If you wish to use a complex datatype, you can use an argument element instead of the attribute.

The wxwML:

```
<function object="myobject" function="Append">
  <argument>
    <list>
      <value><string>Foo</string></value>
      <value><string>Bar</string></value>
      <value><string>Baz</string></value>
    </list>
  </argument>
</function>
```

will be executed as:

```
myobject.Append(list("Foo", "Bar", "Baz", ))12
```

The above is only useful when there is exactly 1 argument, if you wish to use 2 or more you'll have to specify a name attribute for each argument.

The available data types are: number (integer or decimal), string, list, tuple and dict<sup>13</sup>. When defining a list or a tuple each item must be wrapped in a value element. If a value is defined within a string element it will be quoted and all xml escape sequences (such as &amp;) is converted before it's sent to the interpreter.

---

<sup>11</sup> Arguments named value, name, label and title are quoted.

<sup>12</sup> Attributes are sent to the interpreter "as is" which means that the same can actually be achieved with: <function object="myobject" function="Append" argument="['Foo', 'Bar', 'Baz', ]" />

<sup>13</sup> Not all of these have been implemented at the time of writing.

Defining a dict is similar to list and tuple, only it uses the following:

```
<dict>
  <value key="foo"><string>bar</string></value>
</dict>
```

which is interpreted as:

```
dict(foo= "bar", )
```

### Convenience functions

There are a small number of convenience functions defined. These are included mostly because they represent common functions that would have been called explicitly anyway so at least now the wxwML document is a bit more readable. At the moment there's no way to tell the wxBrowser that you DON'T want to use them but this may change in future implementations. See the section Future work.

The first type of convenience function is calling SetValue(), SetLabel() or AppendItems()<sup>14</sup> with the data type element defined as a child of the object. Which function is called depends on what type of object is the parent.

The wxwML:

```
<wx:Choice ... name="choice1">
  <list>
    <value><number>1</number></value>
    <value><number>2</number></value>
    <value><number>3</number></value>
  </list>
</wx:Choice>
```

is converted to:

```
choice1= wx.Choice(...)
choice1.Clear()
choice1.AppendItems(list(1, 2, 3, ))
```

The second type of convenience functions is handling sizers. It works in two parts: adding each child object to a sizer and then setting the sizer of a container.

The following:

```
<wx:Panel ... name="panel1">
  <wx:BoxSizer orient="wx.VERTICAL" name="sizer1">
    <wx:StaticText ... name="static1" />
  </wx:BoxSizer>
</wx:Panel>
```

---

<sup>14</sup> The function Clear() is also called (before AppendItems).

```
is converted to:
# step 1 (creation code) ignored
...
Sizer1.Add(static1)
panel1.SetSizer(sizer1)
```

## Applications

---

The wxBrowser was written as an exercise in order to learn more about the wxWidgets library. I'm not sure if it will ever be useful for deploying real web based applications. I'll give it a try though.

I've written a sample application that can be found on <http://www.pulp.se/wx/contacts/>. At the moment it's very simple but I'll try to expand it as the wxBrowser evolves.

## Future work

---

### Importing subpackages of wx

At the moment, there is no way of using an object that's defined in a wx sub package such as the Scintilla text editor (`wx.stc.StyledTextCtrl`). This is a limitation that will probably be removed in a coming version where the element:

```
<wx:stc.StyledTextCtrl ...>
```

would result in two lines of python code: 1) `import wx.stc` (if it has not already been done) and 2) `... = wx.stc.StyledTextCtrl(...)` (the normal create/bind code)

### Allowing any function to be fetched for POST data in function call.

The version that is released at the time of writing uses a lookup to decide whether `GetValue()` or `GetSelection()` should be called to provide values for POST DATA. In future versions the code will most likely look something like this:

```
<bind ...>
  <argument name="Age">
    <function object="Choice1">GetStringSelection()</function>
  </argument>
</bind>
```

which will let you use any valid method on a widget to be used.

**Flexible convenience functions**

The convenience functions are defined within the wxBrowser, hopefully I'll be able to let them be defined in wxwML documents instead which would make the whole application a lot more dynamic.

**Handling proxies and https connections**

At the time of writing, I have not yet had the time to test the wxBrowser with a proxy server and using an HTTPS connection but it will be implemented in the next version.

**Reducing the need for hard coded information about the wxWidgets library.**

The wxBrowser relies on some meta information about the wxWidgets library. Things like whether or not a widget has a parent attribute, if it can be used as a container etc. At the moment this is coded into the application, which means that if the wx library change I have to change the meta information in the application.

I think that all of the information needed can be retrieved dynamically during run time. The goal is to reduce the hard coded meta information about the wxWidgets library to as close to nothing as possible.

**...And more**

There's a whole bunch of things on my wish list and I'll try to add them to the program as soon as possible. Here are some of the current top items. If you've got one, please let me know.

- Setting local variables
- Being able to use images and icons
- Convenience methods for wx.TreeCtrl
- Improve the HTTP client so that relative URLs can be used
- Remove the argument and value elements used as wrappers for datatypes
- Handle authorization (HTTP 401 errors).