Python Webkit DOM Bindings

The Python Webkit DOM Project makes python a full peer of javascript when it comes to accessing and manipulating the full features available to Webkit, such as HTML5. Everything that can be done with javascript, such as getElementsbyTagName and appendChild, event callbacks through onclick, timeout callbacks through window.setTimeout, and even AJAX using XMLHttpRequest, can also be done from python.

Why is this important - why is it a "big deal"?

Browser engines are amongst the most powerful GUI engines available, with the world's top software organisations and corporations focussing a lot of effort into competing to be "the best". This competition results in some incredibly exciting developments... that are then restricted to being exclusively used from one of the world's most awkward and obtuse programming languages: javascript. So the real question is: why should this state of affairs be tolerated?

Python is a fast-growing language that is highly compact, powerful and elegant. Javascript is obtuse, was unfinished at the time it was released, comes with dire warnings that developers should read the book "Javascript, the good bits", and the only reason it's in popular and prevalent use in web browsers is because there are no other ubiquitous options (certainly no really good free software ones).

Javascript is often confused with the W3C DOM standards, however the DOM TR standards and the HTML5 standard actually do not have anything to do with javascript. Any programming language can in fact use the functions specified in the DOM TR and HTML5 standards: javascript is just the most common case. So it just makes sense to make available the incredible features of HTML5, such as 2D SVG Canvas, 3D WebGL, HTML5 Video, Web "Editor" features and so on, to other programming languages.

The Python Webkit Project therefore makes available all these HTML5 features in a "declarative" programming style (as opposed to an embedded "<script language="python"> style) so that developers can use Webkit as, effectively, a GUI Toolkit engine. The advantages of this approach, over for example using PyGTK2, PyQT4 or any other GUI Toolkit, are numerous. The most important is that the Webkit engine becomes a "one-stop shop" for UI features that would otherwise require multiple Widget libraries and/or plugins. For example: to embed HTML and/or Flash plugins into a PyGTK2 application, it is necessary to install gtkmozembed to plug in the entire firefox web browser! It is then impossible to then interact with the embedded HTML's DOM: there is no event handling - nothing.

What is Python-Webkit?

Python-Webkit is a python extension to Webkit to add full, complete access to Webkit's DOM - Document Object Model. On its own, however, Python-Webkit

doesn't actually do anything, because it is only through WebkitDFB, WebkitGTK or WebkitQt4 that Webkit "Document Objects" are actually created (and displayed, on-screen). Thus it is necessary to make a small patch to each of PyWebkitGTK and PyWebkitQt4, to "break out" access to the DOM, but for WebkitDFB, as it is very new, has its own c-based python module, included as part of PythonWebkit.

Both PyWebkitDFB and PyWebkitGTK have been done, already.

How do I get Python-Webkit?

The Python Webkit DOM Bindings are in development: it's possible to follow or contribute; the source code is available through the git repository.

```
git clone git://git.savannah.gnu.org/pythonwebkit.git
```

For the pywebkitgtk version, please ensure that you check out the "python_codegen" branch:

```
git checkout python_codegen
```

and for the DirectFB (experimental) version, please ensure that you check out the "python_codegen-directfb-2010_11_18" branch:

```
git checkout python_codegen-directfb-2010_11_18
```

If you fail to check out the correct branch, you will end up with a "vanilla" version of webkit, which will simply not provide you with python bindings.

Status

The current status is that the SVG 2D Canvas element is not available, and CSS Styles have to be modified through the style.setProperty function (rather than being accessible as python properties). Other than these gotchas, the Python Webkit bindings are in a useable state, providing full W3C compliant access to the full set of DOM functionality (except SVG Canvas) as provided by Webkit, and ordinarily only accessible via javascript.

The current status of the PyWebkitDFB port is that it is absolutely basic and is "barely but obviously functional", because WebkitDFB itself is still definitely in development. The reason why it is worthwhile mentioning, despite the underlying engine (WebkitDFB) being in development, is the absolutely stunning startup time (measurable in the 10s of milliseconds range on decent x86 hardware). Also it brings home the fact that the PythonWebkit DOM bindings themselves are independent and useable, having been written in a way that makes the actual widget set itself effectively irrelevant.

In short: WebkitDFB proves and demonstrates that PythonWebkit is widget-set agnostic.

How do I use Python-Webkit?

The best example to examine is a fully-functioning platform-independent UI Widget Set called Pyjamas. As anyone who has developed a javascript-based web application without using a Web Framework can attest to, if you don't use a web framework, you very quickly

end up writing your own, and re-learning and re-implementing both
well-known "gotchas" and programming patterns.

As Pyjamas is, at present, literally the only python-based Web Engine
GUI framework in existence, it makes sense to start from there. However,
anyone wishing to roll their own framework is recommended to start from
the pyjd/pywebkitgtk.py script. Here is a simple example of modifying
that script to show the addition of a button, a text node and even event
handling. To any Web Developer who has used javascript, this should
look incredibly familiar:

```
    def _button_click_event(self, event):
        print "button click", event

    def _mouse_over_event(self, event):
        print "mouse over", event, event.x, event.y

    def _view_load_finished_cb(self, view, frame):

        doc = frame.get_dom_document()
        nodes = doc.getElementsByTagName('body')
        body = nodes.item(0)

        d = doc.createElement("div")
        b = doc.createElement("Button")
        b.innerHTML = "hello"
        b.onclick = self._button_click_event
        d.appendChild(b)
        txt = doc.createTextNode("hello world")
        body.appendChild(txt)
        body.appendChild(d)
        body.tabIndex = 5
        #body.addEventListener("mouseover", self._mouse_over_event, False)
        body.onmouseover = self._mouse_over_event
```

Whilst the above is suitable for PyWebkitGTK, for WebkitDFB there is an
example demo/browser.py in the pythonwebkit source tree: exactly the same
tricks as above can be done as with PyWebkitGTK, or you can start instead
from pyjd/pywebkitdfb.py and go from there. The only difference is that
PyWebkitDFB gets the document with a function called "GetDomDocument"
and the DOMWindow object with "GetDomWindow". From there, the standard
exact same pythonwebkit DOM objects are returned: the API will remain
identical between widget sets, as it is part of the W3C DOM standards.

Building and Build dependencies

There are presently two options: PyWebkitGTK and PyWebkitDFB.
Only *one* of these is needed.

Building and Build dependencies for PyWebkitGTK

It is best to follow the build instructions on the main webkit web site,
with the additional requirements of adding python development headers and
the use of PLY (a lexer / parser required to interpret the IDL files).
The python header file requirements and PLY build requirements can be
fulfilled on debian for example with:

```
    apt-get install python-dev python-ply
```

It is worth noting that a very recent version of libsoup is required,
so it is best to get the latest git from http://live.gnome.org/LibSoup

and then get version 2.29.90 using "git checkout LIBSOUP_2_29_90".
Do not attempt to use any other version of libsoup without good reason.
It is also adviseable to use ./autogen.sh --without-gnome in order
to avoid unnecessary build and runtime gnome dependencies.

Also, on debian, a trick can be performed which will get most of the
dependencies and build requirements, by utilising the fact that libwebkit
is already available as a debian package:

```
apt-get build-essential
apt-get build-dep libwebkit-1.0-2
apt-get install libwebkit-1.0-2
apt-get remove libwebkit-1.0-2
```

It seems strange to install then remove libwebkit, but it pulls in and
then leaves all of the dependencies still installed. Do not try this with
aptitude or with any other apt package manager: explicitly use apt-get.
Other package managers may have a habit of being "helpful", by "cleaning up"
after themselves, and removing the very dependencies just installed.

Once the build dependencies have been installed, the source code can be
compiled: bear in mind that the "python_codegen" branch is needed from the
pythonwebkit git repository.

It is recommended to create a subdirectory "build", to cd into it, and
then use "../autogen.sh". The .gitignore is set up, already, to ignore
the "build" subdirectory. You should not need to pass any parameters
to "autogen.sh": the defaults should be set up to provide everything
needed, already.

Building and Build dependencies for PyWebkitDFB

If using PyWebkitDFB, it is, obviously, not necessary to install PyWebkitGTK,
but is *is* necessary to get the dependencies for WebkitDFB. Perhaps
unsurprisingly, this means getting DirectFB. On Debian, this can be
accomplished easily with:

```
apt-get install libdirectfb-dev
```

*Note: It's also recommended to install libdirectfb-extras, as this includes
an X11 plugin which is easier to test with. Edit ~/.directfb and add
these two lines:*

```
system=x11
force-windowed
```

You will still need python-ply, python-dev, and almost the same build
dependencies: again, either follow the instructions on the webkit web
site or follow the above "trick" if using debian.

Also required is the absolute "basic" window engine, Lite 0.8.10. LiTE
0.8.10 is an absolute requirement **not** any other version because it
includes libleck, a strange and wonderful minimalist widget set on top of
LiTE. Unpack, configure, build and install in the usual way by following
the standard method for autoconf'd packages: ./configure etc. etc.

Note that support for libcurl has been added as a preference over libsoup:
you should however still be able to compile with libsoup if that is
preferred. Otherwise, install libcurl4 with, for example on Debian:

```
apt-get install libcurl4-gnutls-dev
```

(Support for libcurl was added due to a clash between the LiTE toolkit

*event loop and the glib/gobject event loop that libsoup relies on.
It was simpler to cut out libsoup and use libcurl instead than it was
to hack together a reliable non-thread-based way for these two event
loops to coexist cleanly)*

Actually compiling pythonwebkit for DFB requires radically different options
from compiling for GTK, so there is a file called CONFIG in the source.
You can examine or execute the CONFIG file in the pythonwebkit DFB branch
as it contains the present set of options that the developers are using for
building. It is best, again, to create a subdirectory "build" and then
do "sh ../CONFIG", then follow the standard procedure to make and install.

At the end of the install, webkitdfb.so (a c-based python module) will
be installed in the python site packages (or dist packages if on debian).
This is all that is needed: you should be able to run the dfb demo browser.py
found in the pythonwebkit source code or use webkitdfb as a pyjd runtime.

Building a patched version of PyWebkitGTK

PLEASE NOTE: as of 11 may 2011, pywebkitgtk is no longer required,
as the pythonwebkit gtk build will now build a stand-alone python module
which can be imported in python as "import pywebkitgtk". As it is still
possible (but unnecessary) to follow the instructions below, they are
still available, here.

Once pythonwebkit has been installed, it is possible to build a patched
version of pywebkitgtk that takes advantage of the new python DOM bindings.
Obtain pywebkitgtk from http://code.google.com/p/pywebkitgtk and apply
this small patch:

    pywebkitgtk.patch

There is also a version specifically for pywebkitgtk 1.1.8, which is the
"stable" version of pywebkitgtk that, importantly, does not have the
GObject Webkit Bindings in it (which will interfere with pythonwebkit).

    pywebkitgtk-1.1.8.patch

Once applied, it is then possible to follow the standard build and
installation procedures for pywebkitgtk. It is then possible to
begin accessing the DOM of the pywebkitgtk WebFrame object, through
the addition of the three extra functions provided by the above patch.

An alternative to obtaining the "original" version and applying the patch
manually is to obtain a pre-patched version from github:

    https://github.com/lkcl/pywebkitgtk/tree/pythonwebkitgtk_1_1_8

The git clone command is:

    git clone git://github.com/lkcl/pywebkitgtk.git

Once cloned, check out the pythonwebitgtk_1_1_8 branch, with the command
"git checkout pythonwebkitgtk_1_1_8". Then, simply proceed with
standard build procedures (./autogen.sh, ./configure, make, make install
etc. etc. all standard stuff, blah blah).

Building a patched version of PyWebkitQt4

Augmenting PyWebkitQt4 to also benefit from the Python Webkit DOM bindings
is on the roadmap, but is not as high a priority. However, judging from the
tiny size of the pywebkitgtk.patch, it is anticipated to be a relatively

small task.

Note about the Source Code

If you read the code, you will note that the term "open source" appears often. We did not put it there. We are free software activists, working for computer users' freedom. We don't describe our work as "open source" because that term reflects different views which don't include our commitment to freedom. For more explanation, see http://www.gnu.org/philosophy/open-source-misses-the-point.html.

This project is meant to extend WebKit. WebKit is a free software package, and we're happy to contribute to it even though its developers endorse open source rather than our views. Most of the code here comes from their version, and that's why the term "opens source" appears in it. We wouldn't have written that, but we didn't change it in their code.

We did, however, change the term "Linux" to "GNU/Linux" in many places where it refers to the GNU operating system with Linux kernel, rather than to Linux itself. See http://www.gnu.org/gnu/gnu-linux-faq.html.

Why not use PyWebkitGTK as-is, or PyWebkitQT4?

These are interesting questions. If you wish to simply "embed" webkit as a browser widget, with zero interaction, purely to display HTML or other web-based media, then you have no need of the Python Webkit DOM bindings, and PyWebkitGTK (unmodified) or PyWebkitQt4 will suit your requirements perfectly.

If however you wish to interact with the DOM model, for example to add in a python function which is called when the mouse is moved, or if input is pasted into a textbox, then you will find very quickly that vanilla PyWebkitGTK and PyWebkitQT4 are hopelessly inadequate for the purpose, for quite different reasons.

PyWebkitGTK does have gobject bindings, and there is also a massive patch to PyWebkitGTK to then provide python bindings on top of these gobject bindings. However, due to bullying carried out in 2008 by specific Webkit developers, the original gobject bindings were not fully incorporated into Webkit. The PyWebkitGTK python bindings have to be specifically tailored to the gobject bindings in order to work. Whilst Free Software developers are working hard to take over the incorporation of gobject bindings into Webkit, they lag considerably behind the original development work carried out in 2008, and thus it makes no sense to keep the PyWebkitGTK bindings up-to-date on something that is both incomplete and still being developed.

The developers of PyWebkitQt4 took a particularly strange route to bringing python to Webkit: execution of javascript snippets that can then have the results partially-converted to python data types! The real problem, aside from the performance issues associated with execution of javascript, comes when more complex data types such as HTMLElements and HTMLElementCollections (e.g. table rows) are returned. At this point it becomes clear that PyWebkitQt4 simply cannot cope. (anyone looking for further explanation should examine Pyjamas pyjd/pyqt4.py, paying attention to CellsProxy, RowsProxy and the way that Document.createElement works).

So it is worth noting that, for different reasons in each case, there is a performance hit in using either PyWebkitGTK Python bindings or PyWebkitQt4

Python bindings: the former due to two levels of indirection (via gobject), and the latter due to it being based on interpretation and execution of javascript code snippets.

The Python Webkit Project provide direct access to the Webkit DOM functions without any javascript being involved, and without gobject or any other intermediate coding layer being called.

Why "redevelop" Python Webkit DOM instead of helping with PyW

Short answer: time. Python Webkit has taken a mere two weeks to get to a useful and useable state, by leveraging and adapting code-generators from the xulrunner project (xpidl.py) and python-gobject (codegen) to create an entirely new and python-based Code Generator.

WebKit's existing bindings Code Generator is written in perl, and has fast become incredibly complex and unwieldy (14 levels of indentation in a single function is not uncommon). Additionally, the gobject bindings which are still in development deviate from the original work from 2008, and yet still provide less features! Thus there would be significant extra work to bring them up-to-date - work which would have to fit in with the gobject development schedule and goals. Also, having new Free Software Developers, the gobject bindings now have different goals which have been explicitly stated as incompatible with the goal of making python a full and complete peer of javascript.

Furthermore, the code coverage and uses to which the gobject bindings have been put, to date, are far less demanding than that of the Pyjamas Project (which requires absolute 100% functional equivalence and interoperability with what is made available in javascript to a web browser). Thus, the new Free Software Developers simply have not encountered issues which were required to be solved in 2008 (from the original development work). Major issues such as race conditions and clashes between GTK and Webkit simply have not been encountered - at all - by the Free Software Developers now responsible for the Webkit gobject bindings. With less experience and no project (other than Pyjamas) requiring such extensive and complete features, the Free Software Developers responsible for the Webkit gobject bindings cannot gain traction to get the Webkit gobject bindings into shape.

So a combination of factors led to a decision to start again from a stronger base, putting python as its main focus (instead of gobject). By leveraging the excellent work from xpidl, codegen and pywebkitgtk, the initial development of the Python Webkit DOM bindings took only two weeks to complete. Independent of the former gobject bindings, further progress is anticipated to be just as rapid, with the end-result using less CPU cycles due to not having the gobject bindings as an intermediate layer.

PythonWebkit being a peer of javascript, a developer may use existing knowledge and existing documentation on javascript-based web development and simply adapt that to python. So even if the gobject bindings were brought up-to-scratch, web developers would still get nasty surprises on finding a particular function or particular functionality missing, and would find it hard to find good technical documentation informing them how to cope, due to cross-interference on google searches for matches where javascript is much higher up the pagerank. Anyone who has endeavoured to look for information on Mozilla XPCOM c++ interface functions using google searches will understand and appreciate this problem. This is primarily why the Python Webkit DOM Project aims for full de-facto javascript-led standards compliance over-and-above strict W3C DOM standards compliance: to make sure that developers can convert their javascript-based applications over to python with the minimum of fuss.

Update 25th Nov 2010:

PyWebkitDFB gives another strong justification for the decision to develop "independent" python webkit bindings. Experimentation has shown that on an embedded 400mhz ARM926 board, a plain vanilla QtWebkit web browser such as arora, compiled with embedded support (DirectFB), takes well over 30 seconds to start up, and consumes a whopping 200mb of memory whilst operational. Both WebkitEFL (Enlightenment Foundation Libraries) and GTKLauncher demo browsers took some 6-8 seconds to start up, and consumed around 130mb of memory whilst operational. WebkitDFB demo browser took a mere 1 (*one*) second to start up, and utilised around 128mb of memory whilst operational (leaking about 1mb per page clicked, due to it still being experimental, of course...)

The point is that if all that is required is a powerful engine for rendering and manipulating HTML5 using python instead of javascript, Qt4 is an incredibly bad choice, WebkitDFB would make an excellent one if it was finished, but thanks to the widget-set independence of the pythonwebkit bindings, PyWebkitGTK can be chosen in the interim, secure in the knowledge that the application being developed *will* work in exactly the same way.

(Of course, if an entirely cross-platform independent HTML5-based Free Software widget set is required then Pyjamas Desktop is definitely worth investigating, as the differences and discrepancies between wildly disparate browser engines across a wide range of modern desktop platforms have been entirely abstracted out. Even Firefox and MSHTML engines are available as pyjd python engines).