Pyjamas

Support this project

Python-based Web Application Development Framework

Downloa

About Overview Features Translator FAQ Getting Help Developing Roadmap

git clone git://

Pyjamas FAQ

This is the FAQ on Pyjamas, the python-to-javascript compiler and Python Web Widget Toolkit. For any questions not answered here please contact the pyjamasdev mailing list.

What is pyjamas?

pyjamas is a stand-alone python to javascript compiler, an AJAX framework / library and a Widget set API.

Is pyjamas like "yet another AJAX framework?"

Nope. "standard" AJAX frameworks and widget sets are written in javascript, a low-level alien language with weird complexities that is often confused with not only the browsers it runs on but also the DOM model set of functions that are provided as standard with the browsers. When you use a "standard" AJAX framework, there is an expectation that you will just "use" it - never look at the framework, never get involved with its development or improvement, and if that AJAX framework doesn't do what you want, tough luck - find another one.

Pyjamas is a totally different approach: it brings "Declarative Programming" (in python) to the Web. You are given a comprehensive set of base classes and modules - written almost 98.5% in python - and you can then write your own applications, classes and modules and call it "My Own AJAX Framework". And, as you are writing in a much higher level language than Javascript, it's a darn sight easier.

Is Pyjamas a server-side web framework?

No.

Is Pyjamas a client-side browser plugin?

No.

Wh

"Thar
seem
to do
- Frar

"Mora
there
1999,
is cut
edge
ouch
Pyjan
extrer
- Bria

Nev

Janu
contri
July
list ar
15th

Use

Manu
Pyjan
AJAX
Pyjan
discu
Mailir
throu
Legac
discu
Grou
Bug T
repor
Sourc
repos
GWT
since
of the
Pytho

No.

Does it fly?

No.

Is Pyjamas a bird or a plane?

No.

How come you can run python, then?

You can't: browsers only support javascript. So, all source code - libraries and applications - are translated into pure javascript. No python is left. at all. Otherwise, it wouldn't work, would it?

How does it work, then?

It's magic. smoke. mirrors. the usual stuff.

Do I have to know any Javascript?

Nope.

Do I have to know any Web Programming?

Nope.

It helps if you know a little bit about how CSS works, because the Pyjamas User-Interface widgets, like all Widget Sets, are designed around "function" and "purpose", not "form" or "look". It would also help if you had a basic grasp of a tiny bit of HTML, such as what <b> etc. does. For everything else, such as tables, there are Widgets such as Grid that are much more practical to use.

Why not?

Pyjamas Web development actually has more in common with Python-Qt4 and Python-GTK2 than it does with "traditional" web development. The base widgets (of which there are over 50) have almost exact corresponding analogues in Desktop Widget sets. The nice thing about Pyjamas, though, when compared to the Desktop Widget frameworks is that you get "true" HTML / CSS interoperability, and "true" plugin support such as Adobe Flash, all for free.

I've never done User-Interfaces before - how do I get started?

We get quite a number of people who have never done user-interface programming before in their lives on the pyjamas-dev mailing lists, let alone having done any web programming: you're in good company.

Exam
applic
T-Shi
with t

Oh

Site

Grafp
beam
Ticke
Naga

expectations that this is going to be horrendously painful, and you are at a distinct advantage because you've not experienced their disillusionment and frustration of the past decade, over CSS, browser and AJAX incompatibilities.

I want to include python module X in my application. can I?

Browsers only support javascript, and pyjs is a python-to-javascript compiler (strictly speaking, a language translator). Therefore, any python module, written in pure python, stands a good chance of working (such as PureMVC). For anything that's written in c, you stand absolutely zero chance, and that's the end of it. However, you can always use a python-based Web Server Framework, and execute whatever python module you choose, server-side, then ship whatever results you choose back to the browser, just like you would with any other web application.

The Pyjamas UI Widget Set and supporting DOM module and other infrastructure is, by the time it ends up as javascript, absolutely no different from any other javascript-based Web Browser Framework - **other** than the fact that the **original** language was Python, and other than the fact that Pyjamas Desktop exists. So, you **could** include python module X in your application, **if** you run it under Pyjamas Desktop - but that is strongly discouraged, for two reasons. Firstly, you stand zero chance of compiling the same application to javascript, should you ever wish to run it in a Web Browser. Secondly, the Web Engines used by Pyjamas Desktop are effectively single-threaded, and, just like with PyQt4 and PyGTK2, control must be handed back quickly to the internal top-level event-handling loop. So, if you execute some CPU-intensive code (in python module X), the entire application visually grinds to a halt and becomes totally unresponsive, just like an overloaded Web Browser, waiting for that code to complete. To avoid this situation, you should use the exact same asynchronous tricks used when code is executed in browsers: AJAX to communicate with a Web Server Framework. (yes, Pyjamas Desktop supports XMLHttpRequest: for the above reasons, it is strongly recommended that you use it).

What server-side web framework should I use?

Absolutely anything you like. Literally anything. CGI-Scripts, ASP/.NET, PHP, Apache, WSGI - you name it, you can use it. The reason is simple: the Pyjamas python code gets compiled to some HTML and some Javascript - that's it. So, just like *any* web pages, you simply need something that will load ... web pages. duh.

Ok - what's the *best* server-side web framework I should use?

This is a highly subjective question, but if you are new to web development, you will be asking these kinds of questions. Given that you're considering python, logically it makes sense to pick a python-based web framework. If you're going to do AJAX, you should really use JSONRPC. If you're using JSONRPC, you should pick a framework that is known to have mature JSONRPC server-side support, and that means Django, Web2Py, Flask, and many more. See the Pyjamas Wiki for links to various JSONRPC integration HOWTOs.

ORM in Django is excellent, the tutorials to get started are excellent, the developers and users on the (busy) IRC channel are very helpful and responsive.

I have an existing Django (or other app) that I want to convert to Pyjamas, how do I do that?

Unless you have already designed the application to exclusively use AJAX in one form or another (REST, JSONRPC etc.) then it's pretty much a complete redesign. The reason is simple: "standard" web applications load up one page at a time from the server, with possibly some bits of javascript thrown in. Pyjamas applications are loaded **once**, as they are **pure javascript**.

From that point onwards, unless you want another large javascript "hit", the entire application should use AJAX to fetch any data, including HTML. If that sounds like it's a bit different from "normal" web applications, you'd be right. It's more like "Desktop" application development.

So - you can convert the Django (or other web app) one page at a time, to a format which separates out the data from its display. In other words: by the time you're done, there will be **no** templates left, only JSONRPC services (or other AJAX service / system). If that sounds like a lot of work, then you can temporarily embed the templates on-demand into iframes, either using a `pyjamas.ui.HTMLPanel` or by using a `pyjamas.ui.Frame` widget. If you've been told that Frames are a bad idea, you'd be right in the "normal" web development world, but completely wrong in this "Desktop-like" case. Thanks to the Pyjamas API, it's possible to set width and height and to react to Window Resize events, and thus to change the entire layout of the application depending on the user preferences, requirements, browser type and window size (as is done in the Pyjamas Mail example).

I'm a bit underwhelmed by the apparent lack of documentation

Pyjamas is an independent free software project that is used by people who prefer to actually get on with the job of writing apps, rather than glorying or revelling in the frustration of "traditional" web development. Hitting your head against a wall continuously every day doesn't hurt any more when you stop doing it, and much of "traditional" web development - of dealing with all the browser incompatibilities at a low level - can be considered to be the same as brick-wall-nutting.

In other words, you're here gaining the benefit of the experience of people who have spent quite some time searching out - and then improving - the Pyjamas Technology, to make their own lives easier. If there's an area which isn't clearly documented, you might like to consider funding them to tell you about it.

That having been said, there is actually one heck of a lot of documentation: see the main Pyjamas site for links to the tutorials, the Pyjamas Book, the API reference guides, the Showcase example which includes source code examples for each type of Widget and Panel (that can be cut/pasted into your app), and much more. If that wasn't enough, the GWT User-Interface modules on which the Pyjamas UI modules are based are so similar that even the GWT documentation is still relevant and useful.

That's because there's a ton of documentation on the W3C standard DOM Model and the W3C standard browser XML standard, and duplicating that in the Pyjamas project would be a great deal of work and would imply that the Pyjamas project is somehow an authority over-and-above the W3C. Use a google search for "W3C DOM XML", "W3C DOM", "W3C DOM Javascript" and other such searches to find so much documentation and help on the subject that you'll be staring at it for days.

Then, start by looking at pyjamas/DOM.py and you will notice that there is a massive prevalance of javascript snippets (way more than any other pyjamas module). It should be plain as day that DOM.py is a way to make the W3C DOM model of web browsers appear to be easily accessible, in python.

I'm developing my app, I need help: what do I do?

Lots of things.

- First: read the question below. Try to resolve the issue yourself.
- Second: read the outstanding bugreports at pyjamas bugreports. Is there anything that vaguely looks related? Read up on it. Does the discussion there have any resolution?
- Third: decide whether the issue is best reported to the bugtracker or to pyjamas-dev or to both. The point of the bugtracker is to be a formal record of issues that need to be fixed. People's heads don't hold large amounts of information all the time, and priorities have to be assigned to tasks. If you want the issue to be fixed for you, you need to make life easier not less easy for the developers to fix it.
- Fourth: if you decide to contact pyjamas-dev, **before** you do so, there are some important things that you need to be aware of.
- pyjamas is a part-time community-driven project, not a corporate-funded one. your absolute top priority in contacting the development list is therefore to make it easy for people to respond. the way to make it easy for people to respond is to do your homework: to provide absolute unambiguous and full information necessary to get people "up to speed".
- To that end: if you have a problem, provide a complete worked example that can be downloaded and run as-is, with no additional effort required other than a few simple commands to install it and run it.
- Provide full context.
- Provide **full** context.
- Provide **full context**.
- Provide the version of the compiler; the version of python; the operating system; provide a complete list of all commands that had been run in order to get you to the point where you are, now.
- If you are running an out-of-date version of the compiler, update the application to the latest svn version of the compiler **before** contacting the list. There are insufficient resources to cater for older compilers. You are welcome to spend **your** time back-porting fixes to the older releases.
- In short: **think**. "Will the people receiving this have to be mind-readers in order to answer my question?". "What work will **they** have to do, to get **me** the results that **I** want?"

in advance for any help" is usually all it takes, making the difference between the reader deciding to provide a terse but informative reply or a more detailed (and probably more useful) one.
- When you get a response, **do what you've been recommended to do!** You got an answer, so why would you want to be disrespectful to the person who spent their personal time and money in composing a reply, by **not** following their advice? If you don't understand the reply, that's fine - *say so*: there's no harm in saying, "I don't understand".
- The other alternative is: if you're not going to do what you've been recommended to do, give *really* good reasons as to why. The point is: don't risk alienating the very people you need help from, in order to fix the problem. Entertain them with the antics that you're getting up to by getting into a horribly tangly mess: yes. Irritate them by wasting their time: no.
- Once you've managed to solve a problem (in part or in full), *say so*. And, if the solution involved following advice given, *say thank you!*. The former is important for anyone else who may encounter the same problem. The latter is important for if you want to ask their advice ever again. If you don't say "thank you", don't be surprised to get a curt response next time, if you get one at all.

[Personal note from lkcl: I've done a lot of free software development, and I've never, despite trying, been able to cover exactly everything, first time, on reporting a problem / asking for help. Software development is complex. "Full" reports are rare and time-consuming for the person writing them, but that's just part of ***your*** *job of writing* ***your*** *app. The key thing is that there should be enthusiasm behind what you're doing: a good developer on a free software project will be able to instantly tell that you're being enthusiastic (and are showing a willingness to adapt), and will likely make allowances that they would otherwise not tolerate. The bottom line is, therefore: use your judgement; provide as much information and context as you can; give some sort of indication of willingness to follow the advice given; follow the advice given; provide people with evidence that you've followed it (or an explanation as to why you haven't), and say thank you if the advice helped.]*

My application doesn't work. What do I do, and why isn't Pyjamas helping?

There are so many things you should be doing to determine the cause of "it ain't working" it's almost unreal. But before answering in more detail, it's important that you know what pyjamas is and is not. First off: pyjamas is a compiler, not a debugger. It's the equivalent of "gcc". Actually, if you know how gcc works, pyjsbuild is the equivalent of "gcc" and pyjscompile is the equivalent of, for example, "/usr/lib/gcc/x86_64-linux-gnu/4.3.3/cc1". In other words, gcc and pyjsbuild are the general-purpose "front-ends", and cc1 and pyjscompile are the actual nitty-gritty compilers.

You *need* to be aware of this because you cannot expect pyjamas, a specialist compiler, to also take on the task of being the debugger, or to incorporate the tasks performed by tools such as pylint, or to be the virtual machine under which the javascript executes, or to be a javascript code-compressor, or to do any other task *other* than actual compiling.

equivalent of adding "-g" to gcc. -d will enable a horrendous amount of extra output in the application, including actually placing the line numbers and module name into the javascript output, for each and every single line of code being executed (!) Also, a stacktrace will be recorded so that if there are any exceptions in your application, an Alert will come up with the full stacktrace (just like in "real" python applications).

- Watch the Javascript Console.
- Watch the Javascript Console.
- Watch the Javascript Console.
- Did we say, and emphasise enough, that you need to watch the Javascript Console?
- You need to watch the javascript console.
- You need to watch the javascript console because that is where runtime javascript errors are shown. Most web applications are written so badly that the errors are usually silently ignored by browsers, so as not to frighten users. As a developer, you cannot ignore the errors (not if you expect to actually be able to write an app, that is...)
- If you are using Firefox, install *both* Venkman **and** Firebug. You will need them both. Use them. get used to using them.
- If you are using IE, install the Script Debugger. use it.
- If you were debugging a c or c++ application, you would be using gdb. The javascript console, Firebug, Venkman, the Microsoft Script Debugger: these are all the equivalent of gdb. use them. You will suck at being a Web developer if you do not use these tools.
- Install Pyjamas Desktop (part of pyjamas) and run your application under that.

Overall, then, it's important to remember that pyjamas is a compiler, not an interpreter, not a debugger, syntax checker or anything else other than a compiler. This may be a bit of a shock if you are used to the python interpreter throwing helpful stack traces for you - which is what Pyjamas Desktop is there for: it actually runs Pyjamas applications *as real python*, not as javascript.

I want to throw some debug output onto the screen, how do I best do that?

With Pyjamas you can use full Python-style logging (Python's flexible event logging module has been ported to pyjs). Additional handlers allow you to display log messages in a dedicated *div* element in the HTML document, in the Web browser's error console, and with JavaScript's alert() function.

```
from pyjamas import logging
log = logging.getConsoleLogger()   # other loggers: Alert, Append, Print ...
...
log.error("Hello, here is an %s error", err_name)
```

For a good understanding of the logging module read the Python Logging HOWTO; most of it directly applies to Pyjamas. Additional Loggers provided by Pyjamas, and how you'd use them:

2. AppendLogger: `log = logging.getAppendLogger()`
   appends text to the end of the HTML document body. The *div* element holding the log messages can be accessed by its element ID ('logging_*loggername*', 'logging_pyjs' by default) for styling.
3. ConsoleLogger: `log = logging.getConsoleLogger()`
   passes log messages on to the error console of Firebug/Chrome/Opera/IE8+ using the console logging functions.
4. PrintLogger: `log = logging.getPrintLogger()`
   prints text to *cerr*, the default error output stream. This is a good choice when developing/testing with Pyjamas Desktop.
5. NullLogger: `log = logging.getLoggerForHandler(NullHandler())`
   disable logging completely and safely. Import this handler with: `from pyjamas.logging.handlers import NullHandler`

The UI Widgets create table layouts. I hate table layouts! Tables Suck!

In the Immortal Words of Bjorne Again, at the free concert on Parker's Piece, Cambridge, in the mid 1990's, *"Tough Titty: We Play Abba, Now"*. But seriously - this is a Desktop-like Widget Set. You use the high-level API. Why would you care if the low-level implementation, which you're never going to have to get involved with, uses HTML Tables instead of your personal preference, CSS "div" pseudo-tables?

Also, think about this: your personal preference, for CSS pseudo-tables, is based on an aversion that was brainwashed into you from having to hand-code and hard-code HTML markup. If you use hand-coded hard-coded HTML in Pyjamas for complex layout, you're *really* doing something wrong. The whole point of Pyjamas is to leverage the simplicity and ease of the Python programming language, along with a declarative programming style, to create an application in terms of Widgets, not in terms of HTML.

Not only that, but the whole reason why CSS pseudo-tables are recommended over HTML tables is because of lack of control over layout. Well, with the declarative programming style from Pyjamas UI Widgets, that lack of control when compared to static HTML is *more* than adequately compensated for. For example, you can easily use Window Resize Change Notify to completely restructure the application layout, dynamically, or you can just resize the Grid itself and its child widgets to fit 100% on-screen, as is done in the Pyjamas Mail example.

Then, also, there's what happens when you create a CSS pseudo-table, and a user shrinks the screen size. What happens? The layout goes completely haywire, because by hard-coding the width of the outer DIV, the inner items now have nowhere to go, and thus your site looks absolutely rubbish. Summary: CSS pseudo-tables are not the panacea you were expecting.

So, the short version is: Pyjamas U.I. Widgets provide far more flexibility and better layout than either of the two "plain" / "static" HTML solutions. Ultimately, if you *really* don't like the HTML-based widgets, feel free to create your own, and submit a patch.

Pyjamas is a compiler - it's a specialist tool that focusses specifically on the job of turning python into javascript. It does *not* include "compression" technology. That task is done by specialist technology such as the "YUI Javascript Compressor", the use of which will make it damn hard for you to debug an application as it completely trashes all of the names of functions and variables. So only make use of compressors if you absolutely have to.

Sujan Shakya has contributed a script to compress pyjs output using such a compressor (`contrib/pyjscompressor.py`). It reduces the output size to about 50%.

HTTP 1.1 GZip Compression is usually enabled by default in both browsers and Web Servers, giving anywhere between an 8:1 and a 10:1 compression ratio in network traffic of pyjamas-compiled output. So, although the output from pyjamas looks scarily large, it's actually not as bad as it looks. 500k is not uncommon: the actual amount of network traffic can be as little as 50k to 80k, which you can verify by installing a network sniffer on your network. So, you actually get the best of both worlds: human-readable javascript and efficient transfer of your application to the users.

The output from Pyjamas is still too verbose. What do I do?

If for example your app is so large that you are hitting, for example, the limits of GAE, you can switch on "dynamic module" compilation (pyjsbuild -m). This option will split your app into several shared javascript files, one per python module. It gives something like a 65% reduction in the size of the application cache files, across the five supported browsers (Opera, Safari, Netscape, Old Mozilla and IE). It's only just been added (to Pyjamas 0.5) so a) be advised b) watch this space.

I'm doing JSONRPC with pyjamas, but my app isn't working.

You should immediately stop trying to do too much, and eliminate pyjamas from the equation. Download a jsonrpc client (eg. python-json-rpc) and add some python-based client-side code to the *server-side* testing. As you are a good web developer, who follows good code standard practices, this should have already occurred to you and it just temporarily slipped your mind that there should be an automated test suite based on something like jsonrpclib.tgz which you can use to *prove* to yourself that the JSONRPC server-side services of your app are absolutely perfect. You already considered the distinct advantages of being able to do automated tests such as pretending to be a user, interacting with the server-side forms, adding and deleting of records etc., but you had juuust not quite got round to doing that yet. well - now you have to.

Once you have demonstrated to your satisfaction that there is nothing wrong with the JSONRPC services, you can then reintroduce Pyjamas, as a second client, into the equation. The difference between then and now is that you now *know* that there is nothing wrong with the JSONRPC services, server-side. So, anything that's wrong has to be something like you've specified the wrong URL in the use of the pyjamas.JSONService module. Check that the URL is the same as the one that you used in the client-side usage of the python jsonrpclib.

match the URL from where the web browser downloaded the HTML and the Javascript. In other words, if your app is downloaded from http://fred.foo.com /myapp.html, then you can **not** put http://fred.foo.com:8080/jsonservice as the URL for the JSONRPC service, you can **not** put http://somewhere.else.com as the URL. It **has** to be http://fred.foo.com/somewhere.

Welcome to the world where people used to use AJAX to do cross-site scripting attacks, downloading viruses and doing phishing attacks and other crap, but can't any more, and you, as a developer, have to pay for that.

I've upgraded to Pyjamas 0.5 (or above) and I get a blank screen.

You should have read the CHANGELOG, which says this:

```
convert app "loading" to require the use of "if __name__ == '__main__'"
bringing pyjamas into line with "standard" python.  apps can convert with
contrib/pyjamas_0.4-0.5_upgrade/add_name_main.py to add a small codefragment
at the end: "app = MyApp(); app.onModuleLoad()"
```

In other words, Pyjamas 0.4 and below used to spoon-feed you by calling a function "onModuleLoad()" in the exact same class as named after your application. As this is somewhat tacky, and places an unnecessary restriction onto how you can lay out your app, it was removed, and replaced with something that is much more akin to how a "standard" desktop python app would work.

The consequences are that your apps will, if you don't explicitly do anything, do absolutely nothing. So, when you get a blank screen, that is actually technically correct, as exactly the same thing would happen with a standard python app! Declare an instance of your app's class and call its onModuleLoad() function.

I've upgraded to svn r785 or greater (or Pyjamas 0.6) and I want to use PyJD

svn r785 or so started supporting XULRunner (python-xpcom) as well as pywebkitgtk, and underwent something of a minor code-shuffle, to make the style of PyJD apps look much more like PyQT4 / PyGTK2. So, for example, instead of gtk.main() you would do pyjd.run(). There's also some "setup" required, which kick-starts the underlying web engine into life. In a web browser (PyJS) of course the web browser is already running (duh) so pyjd.setup() and pyjd.run() do nothing, in PyJS. The functions still have to be there, of course, so that the app can be identical under both PyJD and PyJS. To convert apps to use PyJD, add three lines:

```
# at top:
import pyjd # this is a dummy in pyjs and has no effect

# rest of imports
...
...

# declare classes etc.
...
...

# at bottom:
    pyjd.setup("./public/{thehtmlloaderfile}.html")
    ...
    ...
    # last line
    pyjd.run() # equivalent to gtk.run and does nothing in pyjs
```

I love/prefer {insert AJAX / Javascript framework here}, how do I use it?

Not being funny or anything, but unless you have the resources of google or lots of money or lots of time, or you can gather enough people to make it so that everyone has less work to do: you don't.

huh? why?? Some of the widgets in DojoX / Ext-JS are really cute! I want them! waaah!

You are not in Kansas any more. Pyjamas is declarative-style programming, using a "real" programming language. All those widget-sets were designed to be driven from inside HTML (a style of web development which, using Pyjamas, you have just left far behind) and by inserting javascript snippets into the HTML. If you try that with a Pyjamas app, you are not only likely to get yourself into an awful mess, but also you will be unlikely to run the same (python-based) app under Pyjamas Desktop, as you will still have a Javascript dependency. You *can* run javascript under Pyjamas Desktop but it's not easy to interact with (the python code) which is why we went to all the trouble of replacing all the javascript with equivalent python, doing exactly the same things with python-DOM bindings rather than javascript-DOM bindings.

Shoe-horning an alien AJAX toolkit into Pyjamas takes a considerable amount of attention to detail, and is also likely to have unintended side-effects as the two interact. Not only that, but you are in for a bit of a shock when you actually start looking at the number of lines of code in some of these javascript "rich" widget tooklits, and an even bigger one when it comes to linking the two together.

Take gwt-ext as a successful example of a wrapper between GWT and ExtJS. The "startup" glue code is 8,000 lines of dedicated javascript, the sole purpose of which is to get extjs initialised in a way that's compatible with GWT. 8,000 lines of javascript is about 70% the size of the entire pyjamas codebase, including widget set and compiler, at the time of writing. Then, the "wrappers" are a further whopping 80,000 lines of Java-javascript hybrid.

So, at the end of all that work, you have a wrapper which is itself a monstrous hybrid to maintain, which will need changing every time the Javascript Widget Library API changes, and, on top of that, if the Javascript Widget Library doesn't do what you want, you are screwed, because the whole reason why you're here working with Pyjamas is because you prefer working with python instead of Javascript. And if there's a bug - one that wasn't introduced by the custom-written wrapper - in the Javascript Widget Library itself, then you still have to delve into Javascript and fix it.

Overall, then, does it not make much more sense to rewrite the "pretty" widget in improve it? Especially since the amount of code that you will be writing will be significantly less than if you tried the same thing in Java or Javascript.

So, instead of hurting yourself with javascript, perhaps you might like to consider enjoying Web programming, using python?

I'm installing hulahop and python-xpcom on Pyjamas Desktop on Ubuntu 9, but it don't work, it all broke!

Yep. Someone didn't bother to check that python-hulahop actually worked, in Ubuntu 9 before uploading it. You'll need to rebuild the python-hulahop package, from source:

```
>https://bugs.launchpad.net/ubuntu/+source/sugar-hulahop/+bug/390475
>
> Just to be clear I'm running from svn and it works on Ubuntu if you
> rebuild python-hulahop to get the python 2.6 bindings you don't need
> to set the python path to xulrunner.

nice one michael - thank you for investigating.
```

See the Pyjamas Wiki for contributed information about Pyjamas Desktop and Ubuntu.

I'm evaluating GWT and Pyjamas, and I don't know which one to choose.

You're comparing apples and oranges. You're comparing a community-led effort against a corporate-funded one; you're comparing a good couple of hundred thousand lines of code against tens of thousands; you're comparing a strongly-typed language against a dynamically-typed one; you're comparing a system which is tightly integrated into the Eclipse IDE against one which follows the unix ethos to "do one thing and do it well".

you really do need to look at the differences between the GWT infrastructure and the Pyjamas infrastructure. take a look for example at GWT's i18n and l10n internationalisation support: it's _tens_ of thousands of lines of code, and _hundreds_ of files, supporting over one hundred individual languages, each with number formats, date formats and so on. pyjamas internationalisation support: a pattern, comprising about .... 50 lines of code, because in over a year of development, only two people have ever asked about how to do internationalisation of applications.

our focus, as a community effort rather than a google-corporate-full- time-multi-man-team-funded effort, is therefore on the tools and the infrastructure that _we the contributors_ need and enjoy working on, which is a much smaller subset of functionality than "that which 'employees of google' are tasked with".

so, like lovelysystems, who have some amazing python and javascript engineers, you should utilise pyjamas if a) you feel that the use of GWT's infrastructure and libraries is not going to be worth tolerating the pain barrier of java b) you are prepared to dive in to the (very compact and readable) pyjamas libraries if the was the key reason why lovelysystems did not choose GWT, after around two months of full-time evaluation. JSONRPC highlighted severe flaws in java as a language, thanks to its lack of dynamic typing. the fact that a JSONRPC service could be implemented in fifty lines of python and could be used with one import, one statement and one decorator per function server-side, alongside almost as easy-to-use classes in the web browser, made the choice of pyjamas a no-brainer.

How can i "trust" the pyjamas compiler?

There's a simple answer: the regression tests these regression tests actually run under the standard python prompt, so you add a test, run it in standard python, then compile it and test it in the browser.

we are also working to get the regression tests to run under spidermonkey (the mozilla js command-tool), python-spidermonkey (the python bindings to the spidermonkey library) and pyv8 (the python bindings by flier liu - http://code.google.com/p/pyv8 - to the google v8 library).

once _that_ is done, then there is the plan to start running the STANDARD http://python.org regression tests, all 25,000+ of them, through the pyjamas compiler. yes, this is possible. basically, pyjamas becomes a python accelerator, by translating to javascript and using the JIT google v8 engine to execute the resultant code.

Ultimately, though: trust is always given. In other words: it's entirely up to you.

I'm getting memory leaks in IE6 under XP and Win2000

See MS KB974455. Basically: nothing to do with pyjamas. Apply known MS patches.

How do I help with development of Pyjamas?

For best practices, you should read the DEVELOPER.RULES file which comes with the standard pyjamas distribution. The web site is itself a pyjamas application, so the DEVELOPER.RULES even apply to contributions to the web site.

If you just want to submit a single modification, submit it as a patch, in diff format, at the pyjamas bugtracker and then it is best to join the mailing list, pyjamas-dev, and let people know. You should ideally use git diff or git-am. `(Note: if you are a Windows user, go and get git-win32: there is no excuse for not using the best tools for the job!)`

However, if you want to regularly submit patches, just ask on pyjamas-dev and you will be given an account, simple as that. You will be expected to follow, at all times, the DEVELOPER.RULES, which are really quite straightforward and are there to give you the freedom to do whatever you like, as long as it doesn't inconvenience anyone else (users or developers).

The local filesystem, referenced by file://, is treated as a hostile Cross-Site Scripting (XSS) environment. AJAX now entirely fails on Google Chrome, with urls starting with "file://". There are two different ways to get around this problem:

- Launch Google Chrome with the additional parameter `--allow-file-access-from-files` (or `--disable-web-security`, if you want to be rude), e.g.

  ```
  google-chrome --allow-file-access-from-files
  ```

- Or you start up a web server on your local system, and use http://127.0.0.1/

Please do not raise a bugreport with pyjamas: it is not our fault.

What are all those other pyjs websites I've crossed?

Pyjamas has been hosted on various locations in its lifetime. The current official one is pyj.be, but some others are (somehow) still alive as of 2012:

- Google Code is now only used as an issue tracker.
- Sourceforge was the previous git repository (now cleared).