WikiGateway: a library for interoperability and accelerated wiki development

Bayle Shanks
http://communitywiki.org/BayleShanks
Computational Neurobiology
University of California
San Diego, La Jolla, CA 92093
bshanks at ucsd.edu

ABSTRACT

WikiGateway is an open-source suite of tools for automated interaction with wikis:

- Python and Perl modules with functions like getPage, putPage, getRecentChanges, and more.
- A mechanism to add DAV, Atom, or XMLRPC capabilities to any supported wiki server.
- A command-line tool with functionality similar to the Perl and Python modules.
- Demo applications built on top of these tools include a wiki copy command, a spam-cleaning bot, and a tool to recursively upload text files inside a directory structure as wiki pages.

All WikiGateway tools are compatible with a number of different wiki engines. Developers can use WikiGateway to hide the differences between wiki engines and build applications which interoperate with many different wiki engines.

Categories and Subject Descriptors

D.2 [**Software engineering**]: Interoperability
; H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces; H.4 [**Information systems applications**]: Communications Applications

General Terms

Standardization, Design

Keywords

wiki, interwiki, interoperability, WikiGateway, client-side wiki, WikiClient, middleware, Atom, WebDAV, WikiRPCInterface, wiki XMLRPC

Related resources

Related resources may be found at `http://purl.net/bshanks/work/papers/wikigateway_wikisym05/`. These may include talk slides, amended or hypertext versions of this paper, and pointers to related work. It is suggested that the reader check there for updates before reading this paper.

1. INTRODUCTION

WikiGateway is a suite of tools that allows developers to write programs that act as clients to wiki servers. WikiGateway provides a unified interface to different types of wiki engines. "Wiki engine" refers to a type of wiki server software; for example, MoinMoin is a wiki engine.

There are a few different ways of communicating with wiki servers using WikiGateway tools. These options can be grouped in two categories; either the client software uses WikiGateway components installed on the local machine, or it communicates with a WikiGateway web service installed elsewhere.

The simplest way to interact with wikis is for software running on the client machine to call a WikiGateway library function or tool. WikiGateway provides modules for Python and Perl, and also a command-line client. Appendix A gives a demonstration of various WikiGateway Python module functions. Appendix B shows example invocations of the WikiGateway command-line client.

However, WikiGateway also allows people to run web servers that act as gateways; these intermediaries accept requests from the client using a standardized protocol, and translate them into requests to the actual wiki server (which may be running on yet another machine). WikiGateway provides tools to run servers that understand the WebDAV, Atom, or WikiRPCInterface protocols [37, 11, 21, 2, 13, 23]. WikiGateway also provides a server that exposes an XML-RPC interface to the Python WikiGateway module.

WikiGateway includes a few "demo applications" to show how it can be used. These include a despamming bot, a script to copy an entire wiki onto another wiki, and a script to upload a hierarchial directory structure of text files to a wiki (treating each text file as wiki markup source for a single wiki page).

This paper will begin by describing the reasons for building WikiGateway, including why WikiGateway will help wiki technology to develop faster, and what kinds of software WikiGateway will allow others to build. Next, potential disadvantages of building something like WikiGateway are explored. The design and implementation of WikiGateway are described, followed by a description of demo applications which were written as examples of WikiGateway usage. Finally, future plans for WikiGateway are discussed, including a list of additional demo applications to be developed.

2. MOTIVATIONS

There are quite a few reasons that a project like WikiGateway is needed. **First**, developers are already writing "screen-scraping"-style code to interact with wiki servers; it would be more efficient for the community to write this code "once and only once" in a single library. **Second**, there are standardized protocols that wiki servers could support, but often don't; WikiGateway provides gateway servers to allow clients which speak standard protocols to communicate with wiki servers. **Third**, making wikis more interoperable will aid communications not only with other types of tools, but also between different wiki sites. For example, giving programs the ability to automatically talk to wiki servers is a prerequisite for automated wiki page interchange. **Finally**, by allowing new ideas to be implemented as "universal" client-side tools rather than as server-side features, WikiGateway will help with the problem of the division of developer effort among different wiki engines. These motivations are elaborated upon below.

2.1 A central collection of screen-scraping code

Developers are already spending time implementing interfaces to wiki engines [22, 35, 20, 31, 19, 33, 32]. As long as developers are spending their time writing what is essentially "screen scraping" code to talk to specific wiki engines, why not collect all of these efforts into a single library?

In addition to making it easy for developers to know where to go to find wiki interface code, WikiGateway has the added advantage of providing clients with a single, unified API to use, regardless of which specific type of software the wiki server is running.

In the future, perhaps developers who want to write screen-scraping code for a particular wiki engine will spend their time writing a driver for WikiGateway.

2.2 Interoperability with standard protocols

Many useful software clients use protocols such as WebDAV or Atom. However, there are few wiki engines which support these protocols. WikiGateway provides a way for a DAV or Atom client to access a wiki server, even if the wiki engine running on the server does not know anything about DAV or Atom.

2.3 Wiki page interchange

One much-discussed feature of the future is the ability to programmatically move or copy a page from one wiki to another one, even if the source and destination wiki use different wiki software [17, 39, 27, 1, 16, 29, 7, 36]. This would require two things:

1. The existence of software that can read and write to a variety of different types of wiki engines.
2. The existence of software that knows how to convert the wiki markup between a variety of different types of wiki engine markup styles.

WikiGateway currently can do (1), and the plan is for it also to do (2) in the future (see Section 6, "Future Work").

2.4 Client-side tool development

Problem: fragmentation of developer effort

There are over 250 different wiki engines[1]. There are even quite a few different "popular" wiki engines[2][3]. Even though there are many wiki software developers, their combined development effort is split into hundreds of small projects (see Figure 1) [8].

One way to solve this problem would be to convince developers to pool their efforts and focus development on a small number of wiki engines. This is difficult because it would involve a lot of individual developers and small teams giving up creative control and joining large organizations.

Another answer is to allow developers who are working on a new feature to write a single implementation of that feature which will be compatible with multiple wiki engines. For many features, this could be done by implementing features client-side rather than server-side.

Solution: implement new features client-side

With the ability to download and upload the page source from remote wiki servers, it is now possible to implement wiki user interfaces (UIs) partially or completely on the client. Advanced features such as a revert button, SubscribedPages, "filtered Recent changes", a collaborative rating system for wiki pages, or "unified recent changes"[4], could all be written without altering the software running at the server.

There are also some features that seem to "naturally" belong in the client. Examples include a browsing interface based on an animated, clickable, graphical/spatial "map" of nearby wiki pages; or a Refactoring Browser, that is, a program specifically designed for refactoring text in wiki pages, with special tools for moving blocks of text around between and within pages [28].

[1]252 was the result of a very superficial count of the number of entries on http://c2.com/cgi/wiki?WikiEngines, on April 21, 2005, in the "sorted by language of implementation" section.

[2]This is hard to quantify, but [38] lists 9 wiki engines with over 100,000 Google hits; [41] lists 5 popular wiki engines; [15] lists 5 popular wiki engines; but these two 5-engine lists have only 2 engines in common.

[3]I don't mean to claim that these lists are authoritative or exhaustive, but they do show that it is not the case that aggregate developer effort is being concentrated on two or three wiki engines

[4] "Unified recent changes" is a single list of the RecentChanges of multiple wikis.

Client-side wiki software has not yet been widely developed because, before WikiGateway, it was difficult for client-side programs to communicate with the wiki server. Wiki servers are designed for human clients, not programs. Different wiki servers have different "protocols" for talking to them, each one using their own idiosyncratic set of CGI forms. This meant that it was very difficult to write client-side software that would be compatible with multiple types of wiki engines.

With WikiGateway, client-side tools which interoperate with many different types of wiki engines can be easily developed. This will allow developers to focus their time on interoperable client-side tools, rather than on wiki-engine specific server-side features. In this way, the fragmentation of the developer community can be overcome (see Figure 2).

Freedom for users to choose their wiki software

Today, if your favorite wiki doesn't implement a particular new feature, there is little that you can individually do. You must use the UI that your favorite wiki runs. The developers of the wiki engine and the administrator of your wiki must both take action to enable the end-user to use a given feature.

WikiGateway will allow individual users to use a client-side wiki UI of their choice to interact with their favorite wikis. No longer must all members of a community be bound to the same UI. However, this has disadvantages as well as advantages; see Section 3.2, "Loss of central control over UI".

Wiki software will develop faster

Today, when someone imagines a new wiki feature, it can be extremely difficult for that feature to become widely available. This is because three hurdles must be overcome:

1. The development teams of many wiki engines must be convinced to include the feature.
2. The feature must be implemented in many wiki engines.
3. Many wiki hosts (wiki server administrators) must be convinced to upgrade their wiki server software to a newer version incorporating the feature.

Figure 3 illustrates these problems.

Adding features on the client-side, rather than the server-side, using WikiGateway addresses all of these problems.

1. Because the software is client-side, the development process does not need to involve the developers of the server-side wiki engines.
2. Because WikiGateway hides the differences between wiki engines from the client-side application, the feature can be implemented only once, and yet work with many wiki engines.

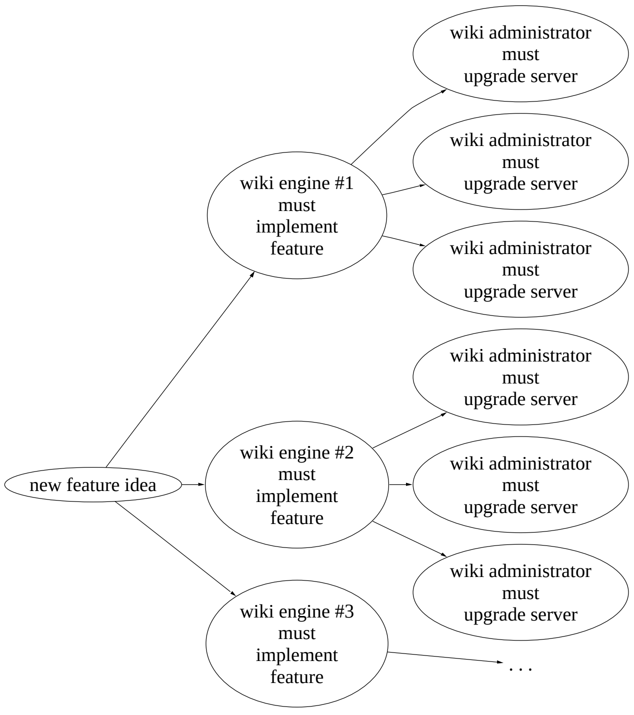

wiki administrator must upgrade server
wiki administrator must upgrade server
wiki engine #1 must implement feature
wiki administrator must upgrade server
wiki administrator must upgrade server
wiki engine #2 must implement feature
new feature idea
wiki administrator must upgrade server
wiki administrator must upgrade server
wiki engine #3 must implement feature
. . .

Figure 3: A lot of work has to be done in order to make a new wiki feature widely accessible. Many wiki engine development teams must implement the feature, and many site administrators must upgrade their wiki server.

3. Because the software is client-side, the wiki server administrator does not need to do anything to enable the new functionality.

With WikiGateway, new features will be able to be used without consulting fifty developer communities, without writing fifty implementations, and without waiting for fifty thousand wiki administrators to upgrade (contrast Figure 4 to Figure 3). People who don't care about new features can continue to use the standard UIs provided by wiki servers, while "power users" will be able to use client-side software providing them with the latest and greatest. Therefore, new feature ideas will be more quickly dispersed throughout the total wiki user base.

2.5 Concrete examples of what WikiGateway can be used for

See Sections 5 and 6.4 for more ideas about how WikiGateway could be used.

3. ARGUMENTS AGAINST WIKIGATEWAY

3.1 Security

It has always been possible to write bots that attack or spam wikis. It is likely that the availability of WikiGateway will assist in the development of some malicious wiki bots. It is possible, although unlikely, that it could even catalyze an avalanche of harmful wiki bots.

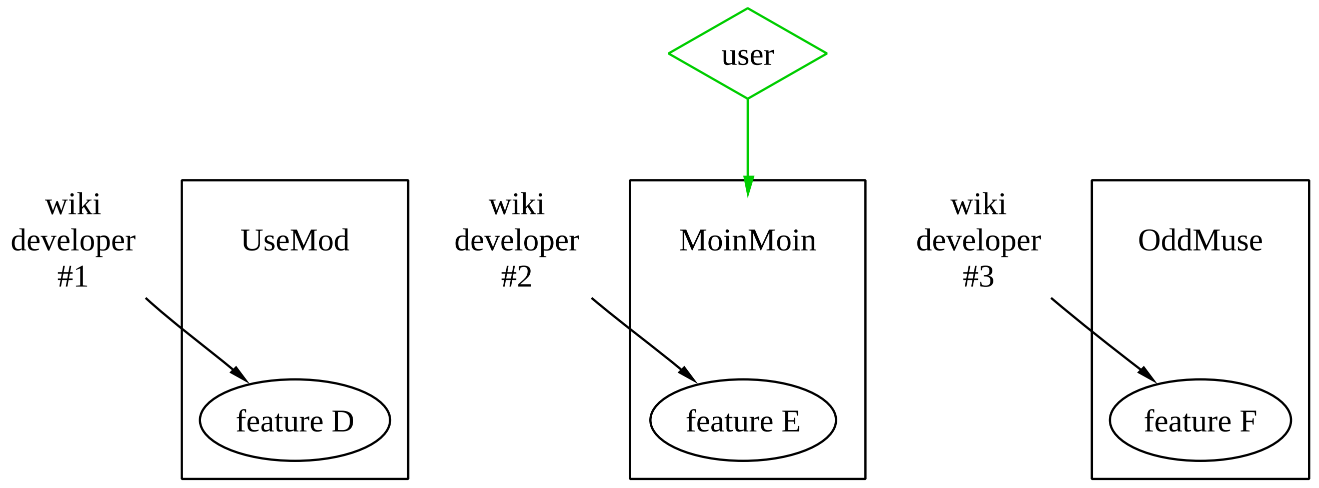


Figure 1: Because there are so many wiki engines, each engine only has a few developers working on it. The sum total of developer effort is large, but the developer effort spent on any single wiki engine is relatively small. So although there are many features implemented somewhere, each wiki offers only a limited set of features. Users who visit a particular wiki can only take advantage of a small proportion of potential features.

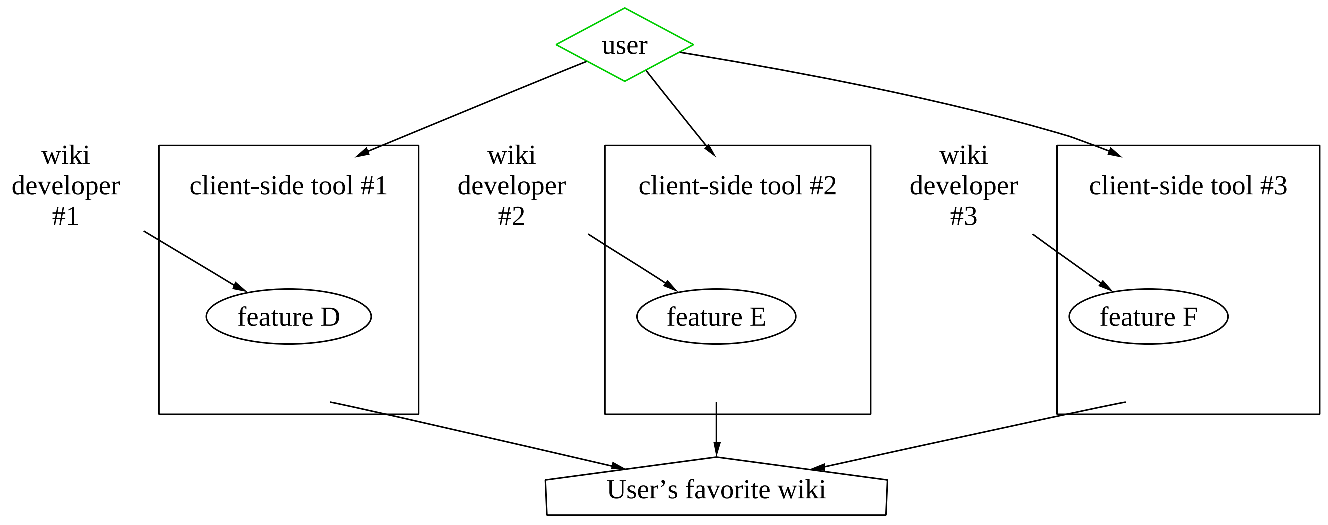


Figure 2: With WikiGateway, developers can work on tools which interoperate with many wiki engines. Developers' time is better spent because the software they create can be used by anyone, not just the users of a particular wiki engine. Users who visit a particular wiki can take advantage of a wide selection of tools.

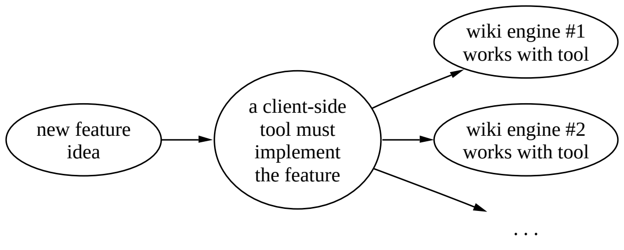

new feature idea
a client-side tool must implement the feature
wiki engine #1 works with tool
wiki engine #2 works with tool
...

Figure 4: With client-side development, neither the developers of all of the wiki engines, nor the administrators of all of the wiki sites need to do anything. A client-side tool must be developed only once, and then users can use the tool on many wikis.

However, not developing WikiGateway would not help, in the long run. Eventually, wiki attack bots would be written either way. Eventually, wiki servers must implement security systems that allow wiki communities to be highly resistant to robotic attack. WikiGateway may force wikis to become secure sooner, but there was never any way to avoid the need to become secure.

In fact, it's possible that if WikiGateway were never developed, the cracker or spammer communities might eventually have developed a similar set of tools to assist in writing malicious bots.

On the other hand, developing WikiGateway provides a number of advantages for wiki communities (see Section 2, "Motivations"). It is even possible that WikiGateway itself could assist in the development of software for wiki security; for example, the WikiGateway project has produced a spam cleaning bot. Unfortunately, unlike many aspects of wikis which could be done at the client, security is, fundamentally, best handled by wiki server software. This is because wiki servers have the last word on which changes get made to the page database.

An additional argument in favor of developing WikiGateway is that it would be easy for individual site administrators to "turn off support for editing via WikiGateway". WikiGateway is hard-wired with algorithms to recognize the relevant components of the edit forms of each supported wiki. A wiki administrator could simply alter the wiki's code so as to make the edit form unrecognizable by WikiGateway. One could argue that the attacker could always rewrite WikiGateway to parse the new edit form, however, if s/he is willing to do that, then s/he could have done it without WikiGateway in the first place.

However, I would hope that site administrators don't take this route, as it deprives their legitimate users of the opportunity to make use of tools build on WikiGateway. A better solution would be to make WikiGateway unusable for anonymous or newly registered users, but to make it usable for trusted users with user accounts.

In summary, since the fundamental security problems exist either way, we may as well develop WikiGateway technology and reap the unique benefits that it can provide.

3.2 Loss of central control over UI

One of the motivations discussed above is "Freedom for users to choose their wiki software". But there are disadvantages as well as advantages to this.

No way to constrain the user

When designing software to support online communities, it may not always be optimal to make every potential action as easy as possible for the user. Sometimes it may be better to make an action annoying or difficult to do for the sake of the group [26]. An extreme example is that a wiki interface with a "Delete all pages" button is not good for the community, even though all it does is provide a shortcut to something that any user could do anyway.

When the entire community is forced to use one UI, it is possible to make some actions annoying or difficult by design. However, when each user can choose their own UI, then it is possible that some users will choose to use UIs that provide shortcuts for actions that "should" be difficult.

Loss of common context

In wiki communities, the UI is part of the common context that the whole community shares [24]. Common context is essential for communication, understanding, and empathy. If everyone views a wiki through a different UI, the unity of the community may be seriously threatened.

Imagine, for example, that some people used a standard wiki server web interface, and others used an interface whose RecentChanges displayed only those changes which were "rated highly" by their friends, while a third group of users viewed the wiki through a wiki to email gateway that emailed diffs of changes to a user-specified list of subscribed pages. One can imagine that these three groups may evolve very different views of what is going on at a given time, and of what the communal norms are.

These two concerns may seem destabilizing, but they are really just instances of decentralization of power, from the wiki administrator to the wiki users. Wikis are unique in that they are extremely decentralized websites, compared to traditional websites in which the webmaster has unique power to write the web content. Yet wikis have taught us that even extreme decentralization can be efficient and secure.

In summary, wiki communities may have to adapt and manage the consequences of decentralizing power over the UI caused by WikiGateway. However, it is likely that they will be able to do so, and that the benefits of WikiGateway will outweigh the costs.

4. ARCHITECTURE OF WIKIGATEWAY

The core of WikiGateway is the Python module[5]. All of the other components are either wrappers around or callers of the Python module, although some of them depend on it indirectly. If the reader has not already done so, s/he is strongly urged to glance at Appendix A in order to get

[5]WikiGateway's core was originally written in Perl, but it has been rewritten in Python.

a concrete feel for the sort of functionality that the core Python WikiGateway module provides.

4.1 The Python module

The Python module is based on a collection of wiki engine-specific *drivers* which implement basic I/O functions for accessing the engines which they handle.

The design goals of the Python module are as follows:

1. The module should be easy to use.
2. The drivers should be easy to develop, even if the developer is a one-off contributor who has little knowledge of the WikiGateway framework.

How the caller uses the module

A caller can interact with the WikiGateway module either in an object-oriented fashion, or in a procedural fashion. For one-off interactions with a remote wiki, the procedural style is more concise:

```
WikiGateway.getRecentChanges(
'http://interwiki.sourceforge.net/cgi-bin/wiki.pl',
'oddmuse1',
'April 11, 2005')
```

The first two arguments in the procedural syntax are always the URL of the remote wiki, and its wiki engine type. After this come any arguments specific to the function; in the example, getRecentChanges takes a timestamp to indicate how far back you wish to receive changes.

For extended interaction with a remote wiki, the object-oriented style is more convenient:

```
wg = WikiGateway.WikiGateway(
'http://interwiki.sourceforge.net/cgi-bin/wiki.pl',
'oddmuse1')
for change in wg.getRecentChanges('April 11, 2005'):
    pageName = change['name']
    print wg.getPage(pageName)
```

WikiGateway may raise exceptions or errors in the course of execution. Exception classes may be found in the module WikiGateway.Errors. Examples are `ReadError`, `EditError` and its subclass `EditConflictError`.

Implementation details

The procedural-style functions are implemented by instantiating an object and then calling the appropriate method on that object.

WikiGateway objects are constructed using two arguments, the URL and the wiki engine identifier. They are actually constructed using a class factory. At run-time, the class factory looks at the wiki engine identifier (and possibly the URL) and decides which of many wiki engine-specific drivers will be used. Each driver is a class. The class factory function `WikiGateway.WikiGateway` instantiates an object of the appropriate class and returns it to the caller.

For an overview of the classes used internally by WikiGateway, see Figure 5. All driver classes are subclasses of `WikiGateway._WikiGatewayBase`, which provides common utility routines which may utilize and alter instance variables of the WikiGateway object.

`WikiGateway._WikiGatewayBase` is a subclass of `WikiGateway._HighLevelFunctions`, which provides default implementations of any functionality that may be built in terms of calls to lower-level methods. An example is `revertToVersion`, which is built in terms of the low level methods `getPageVersion` and `putPage`.

Another commonly used module is `WikiGateway._utils`, which provides static utility functions for use by the driver modules. For example, the function `WikiGateway._utils.getURL_orReadError` fetches a URL or raises a `WikiGateway.Errors.ReadError` if something goes wrong.

The driver modules themselves contain implementations of all of the low-level methods which are specific to each wiki engine.

The Python module is installed using the standard Python distutils installation procedure.

4.2 The Perl module

The Perl module `Wiki::Gateway` is a wrapper around the Python module `WikiGateway`. The Perl module `Wiki::Gateway` can be downloaded and installed from CPAN[9].

Implementation details

The wrapper is constructed using `Inline::Python`. It adds functionality to the `Inline::Python` wrapper by individually wrapping each function with code to intercept Python exceptions and store information about them which can be retrieved by calling the function `Wiki::Gateway::getLastExceptionType()`. In addition, it works around an `Inline::Python` bug with the treatment of unicode by encoding all unicode into ASCII before it passes through the `Inline::Python` interface.

4.3 The command-line client

Appendix B shows some example invocations of the command-line client. Note that, for convenience, the command-line client can refer to a preferences file named `.intermap` to resolve InterWiki-like shortcuts to URLs.

The command-line client is written in Perl using the Perl Wiki::Gateway module.

4.4 Gateway servers

WikiGateway includes gateway servers to support the WebDAV, Atom, WikiRPCInterface2, and XMLRPC protocols.

What are gateway servers?

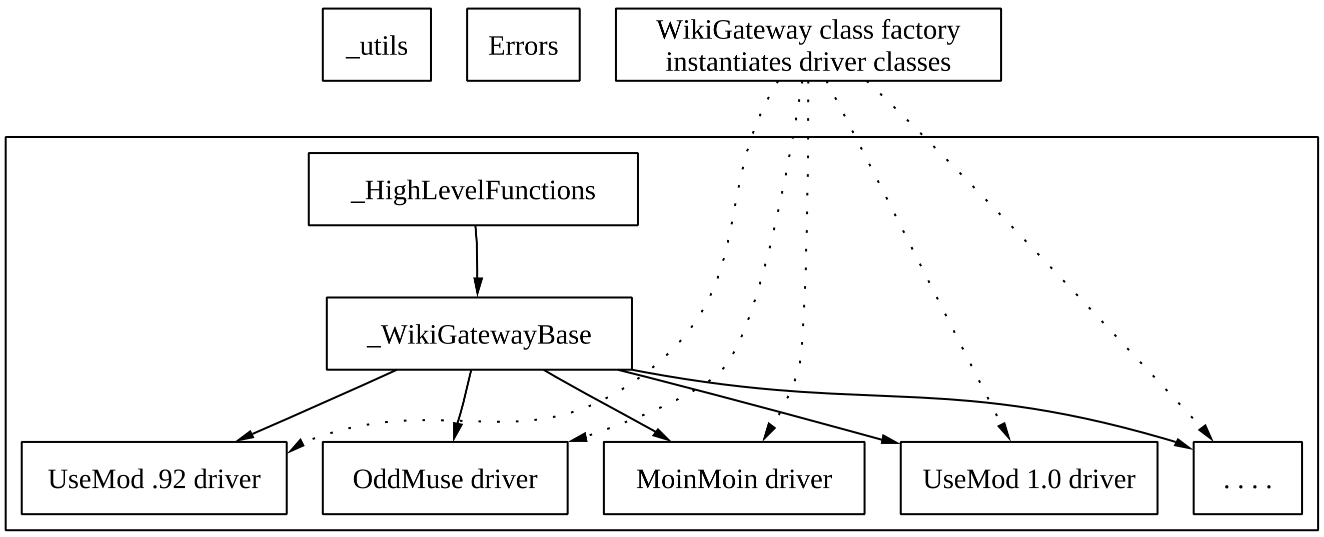
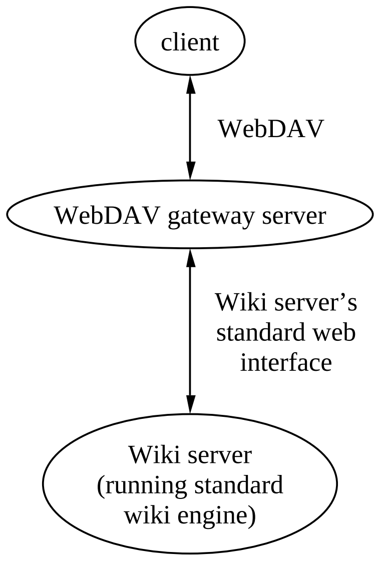


Figure 5: The internal classes of the Python WikiGateway module. Solid lines indicate class inheritance. Dotted lines illustrate that the class factory selects, instantiates, and returns a driver object when the caller creates a "WikiGateway" object. All classes can use `_utils` and `Errors`.



client
WebDAV
WebDAV gateway server
Wiki server's standard web interface
Wiki server (running standard wiki engine)

Figure 6: The gateway servers allow clients to use standard protocols to interact with wiki servers. The wiki servers don't have to implement the protocols themselves; the gateway server acts as a translator to translate the standard protocol into actions on the wiki server's normal web interface. In this example, the gateway server is making a normal wiki server act as a DAV resource.

Instead of forcing the client to use special WikiGateway libraries in order to communicate with wiki servers, there are times when one would want make WikiGateway transparent to the client and allow the client to use a standard protocol to communicate with the wiki server. This is the motivation behind the gateway servers, which are independently-running servers acting as middlemen between the client and the wiki server. For example, consider the WebDAV gateway server. The client sends WebDAV requests to the WebDAV gateway. The WebDAV gateway translates these requests into the HTTP forms requests used by the wiki server. From the client's point of view, it is communicating with a wiki server that knows WebDAV. From the server's point of view, it is communicating with a human using its standard web interface. Figure 6 illustrates this.

This method allows one to use client software that wasn't intentionally designed for wikis. This is useful because there is more software written for protocols such as WebDAV and Atom than there is software written especially for wikis. See "Mounting a remote wiki as a filesystem" in Section 5 for an example.

Gateways allow third-parties to provide web services

Note that with a gateway server, neither the the client nor the server need to be modified or even to know that they are dealing with WikiGateway. The client only needs to use a standard protocol such as WebDAV. The server just has to keep doing what it's always doing; serving users with a web interface.

This approach allows a third-party, that is, someone who is in charge of neither the client nor the server, to effectively "WebDAV enable" the wiki server. Any member of a wiki community can provide this service without the need for the wiki server administrator to do anything.

Implementation details

The DAV server makes it appear as if a remote wiki server is a DAV resource. It is a WebDAV server built on top of PythonDAV and the WikiGateway Python module.

The Atom server makes it appear as if a remote wiki server is Atom enabled. The version of the Atom specification used is [12][6]. It is an Atom server build on top of the Perl module XML::Atom::Server2 and the WikiGateway Perl module.

XML::Atom::Server2 was built for this project and provides an "Atom server" class which may be used as a superclass to build an Atom server. The class contains most of the needed logic; all that the caller needs to add is the backend. In this case, the backend consists of calls to the wiki server using WikiGateway. XML::Atom::Server2 is built on top of the Perl module XML::Atom::Server, which uses a more bare-bones approach.

The WikiRPCInterface2 server makes it appear as if a remote wiki server supports the nascent WikiRPCInterface protocol, version 2.

The XMLRPC server exposes the functions of the WikiGateway Python module as an XMLRPC interface. The server administrator presets the target wiki during configuration. The client can then call XMLRPC methods like getPage and putPage on the server.

The XMLRPC server and the WikiRPCInterface2 server are actually the same program, used with different configuration options.

4.5 Unit tests

Many of the core parts of WikiGateway are equipped with a unit testing framework. Specifically, the most important functions in the Python module, the Perl module, the command-line wiki client, the WikiRPCInterface2 gateway server, and the Atom gateway server all have unit tests.

The unit test framework is object-oriented and hierarchial. The top-most class is `TestReadableWriteableCollection`, which is an abstract class for testing access to a collection of documents. Lower-level classes implement the read/write primitives depending on which component is being tested. Since most WikiGateway components are concerned with reading and writing collections of documents, this allows much of the unit testing code to be reused across different WikiGateway components.

5. DEMO APPLICATIONS

Some software tools have been created to demonstrate the wide range of uses to which WikiGateway may be put. These tools also provide "sample code" for other developers using WikiGateway to look at.

wikicp

`wikicp` is a script to copy all of the pages from a source wiki onto a target wiki.

[6]more recent versions of the Atom specification now exist [14]

spamclean

`spamclean` is a bot to detect and revert spam on a remote wiki. `spamclean` identifies spam by comparing it to a regular-expression-based content blacklist which is downloaded from the internet from a user-specified location. The blacklist is in OddMuse format[5], and one place to get one is [30]. The bot can be configured to ask a user about each piece of spam (interactive mode), or to revert spam automatically.

Other configuration options include the length of time to check for spam, whether to delete pages which have no spamless versions, and the text to put in the summary line upon reversion. The summary line may include variables such as information about the version to which the page is reverting.

`spamclean` reverts spammed pages to the most recent spamless version found[7].

`spamclean` produces a log which notes, for each page with spam, the offending spam, the diff between the spammed version and the clean version, and the action taken.

pushWebsiteToWiki

`pushWebsiteToWiki` is a script that uploads a directory of text files into a wiki. Each file in the directory is treated as a text file containing the wiki markup source of a wiki page. Subdirectories are also recursively uploaded. The delimiter '-' is used in page names to indicate from which subdirectory the page came. For example, if `website` is the directory being uploaded, then a file named `website/books.txt` would become the markup source for the wiki page "`books`". A file in a subdirectory with path `website/ideas/robertsRules.txt` would become the wiki page "`ideas-robertsRules`".

In addition, `pushWebsiteToWiki` creates indices for each subdirectory. For example, the page "`ideas`" would contain an auto-generated index of the subdirectory `website/ideas`.

Mounting a remote wiki as a filesystem

I have used WikiGateway in conjunction with `davfs`[10] to mount a remote wiki running OddMuse. I was able to use the standard operating command `ls` to list the pages on the wiki, and to read and write the wiki pages with text editors as if they were files on my hard drive. When I operated on the file, `davfs` translated the operation into a WebDAV request which was sent to a WebDAV gateway server which is part of the WikiGateway project. The WebDAV gateway server translated the incoming DAV request into an HTTP request that the OddMuse server could understand. So, even though `davfs` didn't know about wikis and OddMuse doesn't speak WebDAV, and even though I did not have administrative access to the remote server running OddMuse, WikiGateway enabled me to use `davfs`.

6. FUTURE WORK

6.1 Support for more wiki engines

[7]Some users have requested a feature where the IP addresses of those who generated the spam are remembered, and the page is then reverted to the latest spam-free version edited by a non-spammer. A similar bot that already has this feature is WikiMinion [32].

Although a major goal of WikiGateway is to achieve portability across many different wiki engines, in fact currently there are only drivers for four types of wiki engines (UseMod[8], OddMuse, MediaWiki, and MoinMoin). I felt that building an extensible framework which made it easy to write drivers, which had unit tests, and which provided a way to let clients use protocols such as DAV or Atom was a more pressing need than initially supporting a large number of wiki engines.

However, now that the framework is in place, I plan to add at least two more drivers by October 2005, and many more in the long run.

6.2 Wiki page interchange

Because WikiGateway is a central collection of algorithms for reading and writing to various types of wiki engines, it is also a natural place to collect algorithms for converting wiki markup between wiki engines. This would allow WikiGateway to offer a "copy wiki page" command that could copy a wiki page from one wiki to another and automatically convert markup styles as necessary.

There is no consensus on how best to do this. One could imagine writing converters for each pair of wiki engines, or one could imagine converting all markup types to a single "canonical" markup, and then converting from the canonical markup to the target markup type. One advantage of using a canonical markup type is that only $O(n)$ conversion algorithms need to be written, as opposed to $O(n^2)$ algorithms if each pair of wiki engines must have a specialized conversion algorithm (where n is the number of supported wiki engines) [4]. Some have suggested that this canonical standard should be XHTML (which is generated by most wikis already), and that the conversion algorithms should be written as XSLTs [18, 3].

WikiGateway could accommodate either strategy, or a mixture of both. Since each driver is independent of the others, different wiki engines could have different conversion routines for their markup. Depending on the particular source and target style, a conversion algorithm could be either "direct" or make use of a "canonical" intermediary.

However, because of the object-oriented structure of the core Python module, functions could also be made available to all drivers which encapsulate common motifs such as XSLT processing.

6.3 Auto-detection of wiki engine type

Currently, a client using most WikiGateway components must tell WikiGateway what sort of software is running on the wiki server. This is inconvenient. It should be possible for WikiGateway to automatically determine what type of wiki server it is dealing with by sending it a series of test requests and analyzing the responses.

For many wiki engines it would probably be sufficient to analyze the HTML of the front page of the wiki. For some, the HTML of the edit form might be needed.

[8]There are three drivers for UseMod, for versions .91, .92, and 1.0.

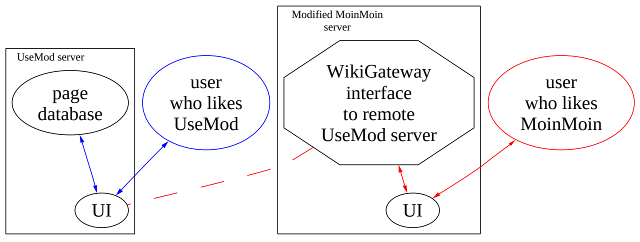


Figure 7: An example of the WikiWindow concept. A user interacts with a modified MoinMoin server; the page database is stored on a UseMod server. Red lines indicate the communications paths involving the red user. Blue lines indicate the communications paths involving the blue user. The dotted line represents a WikiGateway-based interface between the modified MoinMoin server and the UseMod server.

6.4 More demo applications

WikiWindow

WikiWindow is the name for a feature that would allow multiple wiki engines to be used with a single wiki. For example, MeatballWiki runs the UseMod wiki engine. However, with WikiWindow, a user could opt to view and edit MeatballWiki through a MoinMoin server. This would allow users to choose which type of wiki engine software they prefer.

How would this be done? MeatballWiki would be hosted on a standard UseMod wiki engine. On another server, someone would run a specially modified MoinMoin engine. The MoinMoin engine would be modified so that, instead of querying its own database for the wiki page source, it would instead query the real MeatballWiki UseMod server, using WikiGateway (see Figure 7). The UseMod server would return the rendered page text, and the MoinMoin engine would then add its own header and footer, etc.

Wiki mode for Emacs

Alex Schroeder, David Hansen, Pierre Gaston, and Deepak Goel have written an Emacs mode especially for browsing and editing wikis [34]. It is compatible with OddMuse and UseMod wikis. By replacing the UseMod/OddMuse-specific code with calls to WikiGateway, I will make this Emacs mode compatible with all WikiGateway-supported wikis.

Wiki client

At some point, I will implement a very simple wiki client to demonstrate the potential that WikiGateway holds for wiki clients.

Unified, filtered RecentChanges

Users who regularly read a couple of busy wikis have a lot to keep up with. A tool which aggregates changes from many wikis and which allows the user to apply flexible, powerful filtering criteria could allow a busy user to better focus her or his time.

WikiSync

It would be useful to have a tool that allows one to "check-out" a copy of a wiki for offline editing, to edit the wiki pages as text files using standard word processing applications, and then to "resync" the local database with the online wiki [40][6]. This would be useful not only for people actually browsing offline, such as commuters, but also to people who would like to refactor a few related pages with a standard word processor.

A tool like this already exists, but only for Mediawiki [20].

6.5 More unit tests

Presently, there are unit tests for the most fundamental WikiGateway components, but not for all components (such as the demo applications). Eventually, all components will have unit tests.

6.6 Final goal: obsolete WikiGateway

Ideally, wiki engines themselves would provide a way for clients to access them. If all wiki engines supported WikiRPCInterface, for example, there would be no need for WikiGateway. I would prefer for something like this to happen rather than everyone relying upon WikiGateway.

This ideal situation is not likely to come about by itself, however. Here are some of the reasons that wiki servers are not about to support standard protocols.

First, there are not many clients yet. But no one will write clients until there are wiki servers for these clients to access. This is a chicken-or-the-egg problem.

Second, even if there were clients, many wiki developers would prefer to focus their effort on other features. Wiki developers do not want to add extra features they aren't excited about and that may not even be used; that's bad FeatureKarma[25].

Third, there are many competing protocols. DAV is a web standard, but is thought to be too heavyweight, and does not support versioning. WebDAV+DeltaV supports versioning, but is more heavyweight, and has almost no implementations. Atom is simpler than DAV but not yet as widespread or standardized, and does not support versioning. WikiRPCInterface is lightweight and supports versioning, but is not widely known, is not RESTful, and is not used for anything besides wikis (hindering interoperability). Even if we were to convince all wiki developers to go out and implement a protocol immediately, it is not clear which one should be used.

WikiGateway is an interim solution. First, WikiGateway solves the chicken-or-the-egg problem by allowing clients to access wiki servers today, not just tomorrow. Second, WikiGateway does not require any code to be added to wiki engines. Third, WikiGateway supports all three of the above-mentioned protocols, allowing us to defer the decision.

In addition, WikiGateway may make it easier for the necessary protocols to evolve by providing a de-facto reference implementation.

Therefore, WikiGateway makes it more likely that the ideal scenario, in which wiki engines support programmatic access directly, will happen.

7. CONCLUSIONS

WikiGateway is a library which allows software to access wiki servers. The method of access may be a client-side tool (Perl, Python, or command-line) or a standard protocol such as DAV, Atom, or WikiRPCInterface. It is relatively easy for a developers to write a driver for WikiGateway to make it compatible with a particular wiki engine.

WikiGateway will make wiki servers more interoperable with other software, including other wiki servers, wiki client software, and wiki-agnostic software using generic protocols such as WebDAV.

WikiGateway makes it possible to add new features on client-side tools without modifying the wiki server. Since these tools will be compatible with many different wiki servers, it will be possible to overcome the fragmentation of the developer community into hundreds of different wiki engines. This will accelerate the development of wiki technology.

8. ACKNOWLEDGMENTS

L. M. Orchard and David Jacoby contributed the code on which the original Perl version of WikiGateway was based.

Isolani contributed the code upon which the Atom gateway server was based. Alex Schroeder suggested building the Atom gateway server. Christian Scholz wrote Python DAVserver, and Benjamin Trott wrote XML::Atom::Server.

Although I have heard that WikiRPCInterface started on JSPWiki, I don't know who originated it, but thank you to them.

Although the documentation is ambiguous, I believe that `davfs` is written by SungHun Kim and Ric Castillo.

Lion Kimbro helped with testing.

David Cary, Alex Schroeder, Mark Dilley, some anonymous gnomes[9], and especially Dana Dahlstrom helped to edit this paper.

Thanks also to the members of MeatballWiki and CommunityWiki for many helpful discussions.

9. REFERENCES

[1] M. Altheim. InterWikiMarkupLanguage (IWML): A Common Interchange Syntax for Wiki. http://www.altheim.com/specs/iwml/. Viewed on April 26, 2005.

[2] Atom wiki. http://www.intertwingly.net/wiki/pie/FrontPage.

[3] D. Ayers. InterWiki Decoder. http://dannyayers.com/archives/002321.html. Viewed on April 26, 2005.

[9]also known as reviewers

[4] D. Ayers. RE: Re: XHTML. http://dannyayers.com/archives/002321.html, 02 2004. (email).

[5] CommunityWiki. BannedContentDiscussion. http://communitywiki.org/BannedContentDiscussion. Viewed on April 21, 2005.

[6] CommunityWiki. OfflineWiki. http://communitywiki.org/OfflineWiki. Viewed on August 29, 2005.

[7] CommunityWiki. PlanBWikiModules. http://communitywiki.org/PlanBWikiModules. Viewed on April 26, 2005.

[8] CommunityWiki. TooManyWikiEngines. http://communitywiki.org/TooManyWikiEngines. Viewed on April 21, 2005.

[9] http://cpan.org.

[10] http://dav.sourceforge.net/.

[11] Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen. HTTP Extensions for Distributed Authoring – WEBDAV. http://www.webdav.org/specs/rfc2518.html, Feb. 1999.

[12] J. Gregorio. The AtomAPI. http://bitworking.org/projects/atom/draft-gregorio-09.html, Dec. 2003. Old draft of Atom API.

[13] J. Gregorio. The AtomAPI: Publishing Web Content with XML and HTTP. In *XML 2004 Proceedings*. IDEAlliance, 2004.

[14] J. Gregorio and R. Sayre. The Atom Publishing Protocol. http://www.ietf.org/internet-drafts/draft-ietf-atompub-protocol-03.txt, Mar. 2005. Work in progress.

[15] R. F. Halvorsen. TheStateOfWiki. http://heim.ifi.uio.no/~aurilla/web/rune.pdf. Viewed on April 21, 2005.

[16] interwiki-discuss mailing list. http://lists.sourceforge.net/lists/listinfo/interwiki-discuss.

[17] InterWikiWiki. WikiPageInterchange. interwiki.wiki.taoriver.net/moin.cgi/WikiPageInterchange. Currently down until administrator has time to upgrade to new version of MoinMoin. See http://web.archive.org/web/20041012084649/http://interwiki.wiki.taoriver.net/moin.cgi/WikiPageInterchange in the meantime.

[18] InterWikiWiki. XhtmlInterWikiMarkupStandard. interwiki.wiki.taoriver.net/moin.cgi/XhtmlInterWikiMarkupStandard. Currently down until administrator has time to upgrade to new version of MoinMoin. See http://web.archive.org/web/20041009232512/interwiki.wiki.taoriver.net/moin.cgi/XhtmlInterWikiMarkupStandard in the meantime.

[19] D. Jacoby. UnifiedRecentChanges. http://csociety.ecn.purdue.edu/~jacoby/UnifiedRecentChanges/. Can parse RecentChanges from UseMod and fetch them from PhpWiki.

[20] M. Jaroski. WWW::Mediawiki (and wix). Provides an interface to Mediawiki, and builds rudimentary RCS-like functionality on top of it.

[21] E. J. W. Jr. and Y. Y. Goland. WebDAV: A network protocol for remote collaborative authoring on the Web. In *ECSCW*, pages 291–, 1999.

[22] JspWiki. WikiRPCInterface. http://www.jspwiki.org/Wiki.jsp?page=WikiRPCInterface. Specification. WikiRPCInterface has been implemented for OpenWiki, TWiki, UseModWiki, MoinMoin, and PhpWiki. Viewed on April 21, 2005.

[23] JspWiki. WikiRPCInterface 2. http://www.jspwiki.org/Wiki.jsp?page=WikiRPCInterface2. Specification. Viewed on April 21, 2005.

[24] MeatballWiki. CommonContext. http://www.usemod.com/cgi-bin/mb.pl?CommonContext. Viewed on April 21, 2005.

[25] MeatballWiki. FeatureKarma. http://www.usemod.com/cgi-bin/mb.pl?FeatureKarma. Viewed on Aug 15, 2005.

[26] MeatballWiki. PricklyHedge. http://www.usemod.com/cgi-bin/mb.pl?PricklyHedge. Viewed on April 21, 2005.

[27] MeatballWiki. WikiInterchangeFormat. http://www.usemod.com/cgi-bin/mb.pl?WikiInterchangeFormat. Viewed on April 26, 2005.

[28] MeatballWiki. WikiRefactoringBrowser. http://www.usemod.com/cgi-bin/mb.pl?WikiRefactoringBrowser. Viewed on April 21, 2005.

[29] MeatballWiki. WikiXmlDtd. http://www.usemod.com/cgi-bin/mb.pl?WikiXmlDtd. Viewed on April 26, 2005.

[30] OddmuseWiki. BannedContent. http://www.oddmuse.org/cgi-bin/wiki?BannedContent. Viewed on April 21, 2005.

[31] Pywikipediabot. http://pywikipediabot.sourceforge.net/. Provides an interface to Mediawiki, along with lots of other functions useful for writing Wikipedia bots.

[32] RichardP. WikiMinion. http://www.nooranch.com/synaesmedia/wiki/wiki.cgi?WikiMinion. Spam cleaning bot. Interfaces with MoinMoin, OddMuse, OpenWiki, PurpleWiki, and UseMod.

[33] J. Schaefer and A. Schroeder. Automatic Posting and Uploading scripts for OddMuse. http://www.oddmuse.org/cgi-bin/oddmuse/Automatic_Posting_and_Uploading. A series of scripts to communicate with OddMuse.

[34] A. Schroeder, D. Hansen, P. Gaston, and D. Goel. SimpleWikiEditMode. http://www.emacswiki.org/cgi-bin/wiki/SimpleWikiEditMode. Emacs mode for browsing and editing UseMod and OddMuse wikis.

[35] E. Summers. WWW::Wikipedia. Perl module which provides an interface to Wikipedia.

[36] TwikiWiki. RenderOnceReadMostly. http://twiki.org/cgi-bin/view/Codev/RenderOnceReadMostly. Viewed on April 26, 2005.

[37] http://www.webdav.org.

[38] WikiWiki. WikiEnginePopularity. http://c2.com/cgi/wiki?WikiEnginePopularity. Viewed on April 21, 2005.

[39] WikiWiki. WikiInterchangeFormat. http://c2.com/cgi/wiki?WikiInterchangeFormat. Viewed on April 26, 2005.

[40] WikiWiki. WikiSync. http://c2.com/cgi/wiki?WikiSync. Viewed on April 27, 2005.

[41] WorldWideWiki. PopularWikis. http://www.worldwidewiki.net/wiki/PopularWikis. Viewed on April 21, 2005.

APPENDIX

A. Demonstration of the Python WikiGateway module

For readability, longer responses were truncated, blank lines added, and a subset of a longer session was selected for inclusion.

```
Python 2.3.4c1 (#2, May 13 2004, 21:46:36)
[GCC 3.3.3 (Debian 20040429)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import WikiGateway
>>> print WikiGateway.getPage(r’http://interwiki.sourceforge.net/cgi-bin/wiki.pl’, ’oddmuse1’, ’SandBox’)
This is a test of <nowiki>WikiGateway</nowiki>. Test 2.


>>> wg = WikiGateway.WikiGateway(’http://interwiki.sourceforge.net/cgi-bin/wiki.pl’, ’oddmuse1’)


>>> wg.getAllPages()
[’AboutInterWikiSoftware’, ’AdvantagesOfAGateway’, ’AndrewGray’, ’AtomGateway’, ’AtomServerModule’, ...(truncated).


>>> wg.getRecentChanges(’April 11, 2005’)
[{’comment’: u’Automated update of spam blacklist’, ’name’: u’BannedContent’, ’importance’: u’major’,
’lastModified’: u’2005-04-19T12:11:52+00:00’, ’version’: u’57’}, {’comment’: u’revert to revision 39’,
’name’: u’SpamClean’, ’importance’: u’major’, ’lastModified’: u’2005-04-17T08:00:54+00:00’,
’version’: u’41’}, {’name’: u’SandBox’, ’importance’: u’major’, ’lastModified’: u’2005-04-16T17:58:31+00:00’,
’version’: u’439’}, {’name’: u’WikiSandBox’, ’importance’: u’major’, ’lastModified’: u’2005-04-16T10:31:32+00:00’,
’version’: u’1’}]


>>> wg.putPage(’SandBox’, ’py wg test’)


>>> wg.getPageInfo(’SandBox’)
{’date’: <DateTime object for ’2005-04-21 16:19:00.00’ at 407596b0>, ’comment’: ’’, ’version’: 440,
’author’: ’user-10cmeae.cable.mindspring.com’}

>>> wg.getPageInfoVersion(’SandBox’, 429)
{’date’: <DateTime object for ’2005-04-11 15:03:00.00’ at 4077c4b8>, ’comment’: ’’, ...(truncated)...}


>>> wg.getPageHistoryInfo(’SandBox’){416: {’date’: <DateTime object for ’2005-04-05 16:25:00.00’ at 4077c7c8>,
’comment’: ’’, ’version’: 416, ’author’: ’user-10cmeae.cable.mindspring.com’}, 417: {’date’:
<DateTime object for ’2005-04-10 04:46:00.00’ at 4077c790>, ’comment’: ’’, ’version’: 417,
’author’: ’user-10cmeae.cable.mindspring.com’}, ...(truncated)...}


>>> wg.getPageVersion(’SandBox’,439)
’This is a test of <nowiki>WikiGateway</nowiki>. Test 2.\n’


>>> wg.getPageHTMLVersion(’SandBox’,439)
’<p>This is a test of WikiGateway. Test 2.</p>’
```

B. Sample invocations of the command-line wiki client "wikiclient"

InterWiki prefixes are resolved using a file ".intermap" in the user's home directory.

```
wikiclient --type=usemod1 read MeatBall:SandBox
wikiclient --type=usemod1 read http://www.usemod.com/cgi-bin/mb.pl:SandBox
wikiclient --type=usemod1 write http://interwiki.sourceforge.net/cgi-bin/wiki.pl:SandBox --summary="just a test"
   <the text to be written is read from STDIN>
wikiclient --type=usemod1 rc MeatBall
wikiclient --type=usemod1 rc:11 MeatBall
wikiclient --type=usemod1 allpages http://interwiki.sourceforge.net/cgi-bin/wiki.pl
wikiclient --type=usemod1 info MeatBall:SandBox
wikiclient --type=usemod1 info MeatBall:SandBox --version=2384
wikiclient --type=usemod1 info MeatBall:SandBox --version=last
```