



Groowiki Users Guide

Date (YYYY-MM-DD): 2009-07-27
Author: Peter Verhas
Language: English
State: DRAFT
Security: CONFIDENTIAL / BIZALMAS
Project: GROOWIKI
Version: 1.1
Document ID: 1.3.6.1.4.1.13923.0.75397328
Revisions:

Version	Date	Author	Comment
1.0	2009-03-22	Peter Verhas	Initial version.
1.1	2009-07-27	Peter Verhas	

This document, file, any kind of material (referred as object hereto) and any information enclosed within it, contains restricted and/or privileged information and is intended only for authorized screening and/or confidential presentation at VVSC discretion. If you are not the intended observer of this object, you must not disseminate, modify, copy/plagiarize or take action in reliance upon it, unless permitted by VVSC. None of the materials provided on this object may be used, reproduced or transmitted, in any form or by any means, electronic or mechanical, including recording or the use of any information storage and retrieval system, without written permission from VVSC. If you received this object in error please notify VVSC immediately. (legal_report@verhas.com) The confidential nature of and/or privilege in the object enclosed is not waived or lost as a result of a mistake or error in this object. VVSC accepts no liability whatsoever, whether it was caused by: 1. Accessing or other related actions to this object. 2. Any links, and/or materials provided/attached to this object. VVSC assumes that all users understand risks involved within this object and/or its attached materials. All rights reserved. All other brands, logos and products are trademarks or registered trademarks of their respective companies. This legal text is presented in this object in English as well as Hungarian. In case there are differences either in content, meaning or format between the versions the one permitting less freedom for the accessors of the object prevails. Hungary, November 1, 2006.

...

Ez a dokumentum, fájl, vagy bármilyen más anyag (objektum) és minden ebben foglalt információ titkos és/vagy bizalmas információkat tartalmaz, amelyet csak arra feljogosított személyek tekinthetnek meg. Ha az objektumot nem Önnek szánták nem másolhatja le, nem adhatja tovább, nem módosíthatja, és nem használhatja fel semmilyen módon, csak ha a felhasználásra engedélyt kapott a VVSM-től. Semmilyen ezen objektumban található, vagy kapcsolódó anyag nem másolható és nem továbbítható semmilyen formában, semmilyen módon, elektronikusan vagy mechanikusan, beleértve a felvétel készítést, adattárolási és visszakeresési rendszerek használatát, a VVSM írásbeli engedélye nélkül. Amennyiben véletlenül jutott ehhez az objektumhoz kérjük haladéktalanul értesítse a VVSM-et. (legal_report@verhas.com) Az objektum bizalmassága titkossága nemvész el, és nem csökken a dokumentumban történő bármilyen hibától vagy sérüléstől. A VVSM minden felelősséget elutasít, bármely eseménnyel vagy történéssel kapcsolatban, amelyek az objektumhoz történt hozzáférés során történt. A VVSM feltételezi, hogy minden felhasználó tisztában van minden veszéllyel, amelyet ennek az objektumnak a használata jelenthet. Az objektumban szereplő minden márka, logó a megfelelő tulajdonosok márkanéve vagy bejegyzett márkanéve. Az objektumban ez a jogi szöveg angolul és magyarul is szerepel. Amennyiben a két szöveg között tartalom, formátumban vagy egyéb módon eltérés van az a mérvadó, amelyik az objektumot használó számára kevesebb jogosultságot ad. Magyarország, 2006. november 1.



Contents

1.Introduction.....	3
2.Basic features.....	3
3.Architecture.....	3
4.Installation under Tomcat.....	3
5.Configuration.....	3
6.URL Structure.....	5
7.Document upload.....	6
8.Templates and CSS.....	7
8.1.Action/Business Type Templates.....	7
8.2.General templates.....	8
8.3.Template Parameters.....	8
8.4.Template helper class.....	10
9.Defining Action Classes.....	10



1. Introduction

This is the Users' Guide of the program Groowiki. The document describes the architecture of the program, the installation, configuration, the use, template, CSS and programming.

The use of the program is very simple. Groowiki is, simply to say, a web front-end for Subversion. What you can see is a web interface that one can navigate along the directories and files and you can download or upload files and set, alter file and directory properties.

These are the basic features of Groowiki out of the box. However in addition to these you can edit the title and the description of the files and the directories and define the “business type” for each. This way you can create business specific behavior for your actual implementation of Groowiki. It is a simple “application platform” that supports radeox¹ formatting Wiki pages, work-flow, versioning of documents, Java and groovy plugins, fine-grained access control and many other features.

2. Basic features

Using Groowiki you can manage your files, manage their versions and their meta data in a convenient way using a well crafted Web interface. You can upload, download files, have a look at the list of the files in the directories, navigate between the directories and look and modify the meta data of each item (be file or directory).

When you log into the web page of Groowiki you see the root directory listing:



Demonstration repository

This is a demonstration repository

Groowiki

Its only aim is to demonstrate the capabilities of

Do you like the logo?

To have a feel of Groowiki click on the names of the directories and file, read the descriptions and try to edit the descriptions titles and other properties.

Why don't you edit this text clicking on [edit] and then insert your name into the text:

This page was last edited by **Verhás Péter** DO NOT FORGET to write some comment for your change into the 'log message' field before you press the button 'send'.

After you edited this form click on the directory 'test1FolderNoBusinessType'. Please notice the navigational stripe above the directory title.

[edit]

PROPERTIES

File name	author	revision	message	date	size (in bytes)
test1FolderNoBusinessType/	verhas	205		2009-05-08	0
test2FolderBusinessTypeInvoice/	verhas	195		2009-05-08	0
test.txt	verhas	183	click on the	2009-05-08	30

Total 3 entries = 2 directories and 1 plain files

 © Verhás & Verhás Software Craftsmen 2006-2009

1 Some of the features may be available in later versions only.



On the page you can see the title and the description of the directory (this is the top level directory) and the list of the files. You can navigate to the files and the directories clicking on the links in the file list. If we click on the link 'file.txt' (which is a sample file in the root directory of the sample repository) then we will get to the following page:

The screenshot shows the Groowiki interface for a file named 'test.txt'. At the top is the Groowiki logo, which includes a soccer ball and a smiley face. Below the logo is a blue header bar with the text '/ test.txt'. The main content area has a title 'A special file that contains all Hungarian letters' and a description: 'The content of this file is' followed by the Hungarian text 'árviztűró tukórfúrógép'. Below this, it explains that it means 'food resistant mirror drilling machine' and that it contains all accented characters of the Hungarian language. There is a green download button and an '[edit]' link. A 'PROPERTIES' section shows a table with one row: 'just-some-property' and 'The value of the property can be any string.' At the bottom, there is a copyright notice: '© Verhás & Verhás Software Craftsmen 2006-2009'.

On that page you can see the title, description and the properties of the file. This is an ordinary file without so called business type. To download the file (or in case of HTML and text files to see the content in the browser) click on the green download button. You can see a link [\[edit\]](#) that you can click on to get to the editing page:

This screenshot shows the same Groowiki file page for 'test.txt', but in edit mode. The Groowiki logo and header are the same. The main content area now shows the file's content rendered as HTML code, including the title and description. Below the content, there is a 'Description format:' dropdown set to 'html' and a 'log message:' input field with a 'send' button. The 'PROPERTIES' section is expanded, showing a table with columns for 'name' and 'value'. Below the table, there is a 'Log message:' input field and a large text area for editing the file's content. At the bottom, there is a detailed instruction: 'You can also set value of existing properties or create new properties. On the left write the name of the existing or new property and on the right the desired value. You can set at most 10 properties at a time. You can not set binary values and it is recommended NOT to use accented letters in the property names.'

On the top you can edit the title and the description of the file and on the lower part you can edit the properties. To set the value of a property type the name of the property on the left and the value of it to the right. If the property already existed the new value will override the old.

Soon we will see that properties are usually updated using business type edit forms providing a much more convenient way for editing. Even though it is recommended that you start Groowiki now and try to navigate and edit files and directories in the sample repository.



3. Architecture

Groowiki is a servlet application. It is written in Java and runs in a servlet container like Tomcat. Groowiki does not have own storage (as of version 1.2.0). It connects to an SVN repository and as a proxy in front of the repository it acts as a front end.

To be further detailed.

4. Installation under Tomcat

To install Groowiki under Tomcat 5.5 unzip the content of the WAR file into a directory named `/opt/groowiki`. The actual name of the directory may be different, like `/usr/local`. Whichever it is choose the directory where you usually install your applications.

It is usually a good practice to extract the WAR file into a directory that has a name something like `groowiki-1.0.0` that contains the versioning information in the file name and create a soft link from it to the directory named `groowiki`. This helps the upgrade process from one version to the other.

Set the owner and the group of the directory and the files so that application tomcat can read the file. The UNIX commands will look something like this.

```
# mkdir /opt/groowiki-1.0.0
# ln -s /opt/groowiki-1.0.0 /opt/groowiki
# chown -R tomcat:users /opt/groowiki-1.0.0
```

We assume that Tomcat is installed in the directory `/opt/tomcat`. Create a file named

```
/opt/tomcat/conf/Catalina/localhost/groowiki.xml
```

and edit it to have the following content:

```
<Context path="/groowiki" docBase="/opt/groowiki/" debug="0" reloadable="false">
</Context>
```

Doing so will make the application appear in the Tomcat web application list. However do NOT start the application in Tomcat yet. Configure Groowiki before starting it.

5. Configuration

5.1. Configuring Commons

Edit the file `/opt/groowiki/WEB-INF/classes/config.xml` to contain the following:

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>
  <system/>
  <xml fileName="/opt/groowiki/WEB-INF/classes/groowiki.xml"/>
  <xml fileName="/opt/groowiki/WEB-INF/classes/mimeTypes.xml"/>
  <xml fileName="/opt/groowiki/WEB-INF/businessTypes.xml"/>
  <xml fileName="/opt/groowiki/WEB-INF/plugins.xml"/>
</configuration>
```

The two `fileName` parameters should point to the two files `groowiki.xml`, `mimeTypes.xml`, `businessTypes.xml` and `plugins.xml`. These are configuration files, which contains XML formatted description of the environment and the parameters that Groowiki has to follow. You can split up the configuration into as many file as you wish, and there is no restriction what parameters to put into which file. If you want you can compile the whole configuration into a single file or you can split up the



configuration into many files.

The file `mimeTypes.xml` contains the list of the mime types that are supported by Groowiki. This is used to set the content type parameter of the http answer, when a file is downloaded. The content type is determined by the extension of the file name, and the appropriate content type is selected from the configuration. There are a plenty of mime types configured in the file, probably you need not edit this file.

5.2. Main configuration

The file `groowiki.xml` contains the configuration of the application. A sample content of the configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Document   : groowiki.xml
  Created on : February 13, 2009
  Author    : Peter Verhas
  Description:
             groowiki sample configuration
-->
<configuration>
  <repository>
    <charset>ISO-8859-2</charset>
    <url>https://mySVNserver.com/doc/</url>
    <!-- professional edition and above START -->
      ... other options for professional and above editions ...
    <!-- professional edition and above END -->
  </repository>
  <velocity>
    <templates>
      <root>/opt/groowiki/WEB-INF/classes/templates/</root>
    </templates>
  </velocity>
  <descriptionFormats>
    <plain class="com.verhas.groowiki.wikiformatters.Plain"/>
    <html class="com.verhas.groowiki.wikiformatters.Html"/>
  </descriptionFormats>
</configuration>
```

The configuration is an XML file that is handled by the `apache.commons.config` package. You have to set the configuration parameters that are listed in color **red** in the list above and you may consider to alter the values that are colored **blue**. (If the actual configuration file contains more keys in this part, don't worry. These will be explained soon in this document.)

The configuration parameters are the followings:

- `url` (under `repository`) should define the URL of the repository that groowiki connects to.
- `velocity.templates.root` should define the directory where the velocity templates are. The trailing `/` is a must.
- `charset` (under `repository`) should define the character set that the repository uses. The repository actually does not use any character set, the files and the properties are simple byte streams for the repository. On the other hand the client applications display and store the national characters like ő,ú,ŵ, Ж using some character set. By default Groowiki uses utf-8 thus if you enter a revision log message or property value containing national character then it will



be stored in the repository utf-8 encoded bytes and the other way around: Groowiki will convert the bytes as utf-8 encoded byte stream. Using this configuration you can override this behaviour. Hint: if you use TortoiseSVN you may want to use ISO-8859-1 in western Europe and ISO-8859-2 in central and east Europe. If you speak/use only ASCII English then forget this option.

- `descriptionFormats` should list the different formats that are supported by the file and directory descriptions. Groowiki 1.0.0 supports only two formats: plain and html. The attribute `class` defines the Java class that handles the conversion of these formats when the description is displayed. You better do not alter these configuration options.
- `security.forbidHtmlError` can be either `true` or `false`. The default value is `false`. If this value is set to `true` then Groowiki will not send a simple “page can not be displayed” message to the client browser instead of the detailed text of the exception. It is recommended to use the default value in development environment and use the value `true` in real operation. Exceptions are always listed in the log files in full detail.
- `maxUploadableSize` can define the maximum size of a file in bytes that groowiki accepts as a new file or a new version of a file already in the repository. This is not a recommended way to upload files using the web interface, but rather through normal SVN ways. Using the web interface is slower and you can only upload limited number of files in a single commit. The default value is 100MB.
- `maxFileStoredInMemory` when a file is uploaded this many bytes are stored in the memory of the server. If a file is larger than this then the upload process writes the bytes into a temporary file and afterwards groowiki reads the bytes from the temporary file. The default value for this parameter is 100MB. Setting this parameter larger than the parameter `maxUploadableSize` is senseless.

5.3. Configuring license

A license is a plain text file that you have to copy to a location where it can be accessed by Groowiki application to read.

```
<edition>
  <license>
    <file>/opt/groowiki/licence-2009-08-18-vvsc.coded.professional.txt</file>
  </license>
</edition>
```

Groowiki is looking for the license file in the configuration under the key `edition.license.file`. Write the full path to the license file into this configuration key. If there is no license configured, or the license is invalid (file does not exist, badly formatted or the license expired) then Groowiki will start with the Community edition features. The edition groowiki is performing you can see on the opening page in the footer after you logged in.

5.4. Authentication configuration

Community edition of Groowiki relies on the authentication of the underlying SVN repository. The user name and the password provided on the web interface by the user is used to access the repository. In professional, enterprise and ASP editions there are more options. Any of these editions are



compatible with the community edition though, if you start groowiki with a professional or higher level license using the community configuration the execution should be the same.

If you use the professional edition you can specify what user name and password to use when accessing the repository. This means that the users are authenticated against a different authentication mode and the repository is accessed a service user name and password. This way professional edition need not disclose the user names and passwords that give direct access to the repository via the direct http, svn or other protocols.

```
<repository>
  <url>https://svn.verhas.com/cs/demo/</url>
  <charset>UTF-8</charset>
  <username>vvsctestaccountRW</username>
  <password>ahki_ssudd$j29/%</password>
  <plugin>
    <username></username>
    <password></password>
  </plugin>
</repository>
<authentication>
  <type>file</type>
  <file>/opt/groowiki/users.txt</file>
</authentication>
```

Professional edition and higher will use the configured value of the key `repository.username` as user name and `repository.password` as password to access the repository. If they are not defined then the user name and the password specified by the user on the web interface will be used. You can also define a separate user name and password pair optionally `repository.plugin.username` as user name and `repository.plugin.password` as password to access the repository when a plugin is executing. That way plugins may get different rights accessing the repository (usually less restricted). If these two keys are not defined then the user name and password inherited from the earlier mentioned keys, or from the web interface.

Enterprise edition gives even more flexibility. Although you can configure the enterprise edition the same way as the professional edition you can also define service user name and password for each plugin individually. If you add the configuration tag 'repository' to the configuration of a plugin then you can define the user name, password and even the URL of the repository for that very specific repository. An example:

```
<sampleGroovy id="com.verhas.groowiki.sampleGroovy" type="groovy">
  <repository>
    <username>vvsctestaccountRW</username>
    <password>sheep123</password>
    <url>http://myserver.com.intranet/svn/repo</url>
  </repository>
  <script>samplePlugin.groovy</script>
</sampleGroovy>
```

The reason to configure a plugin so detailed can be various. You can fine tune the access control of the plugins differently, each plugin may get permission to access only the resources it really needs. You can write a plugin that pulls information from another repository. You can access the repository by different protocol. For example the users access the repository through Groowiki using their user name and password via http Web DAV that enforces access control, while a plugin (but not all) can access the repository using the `file://` protocol because this very plugin is trusted and the direct access is faster.



If the repository is accessed using service account there is a need for a user database that the actual provided user name and password can be checked against. This can be done under the key `authentication`. The value of the configuration key `authentication.type` has to be `file` or `crypt`. Later releases will support more user stores. The key `authentication.file` should give the full path to the file where the user names and passwords are stored. This file is read each time a user logs in. Any change in this file will become effect immediately.

If the type is `file` then the content of the file should be similar to

```
peter:kakukk  
pityu:mityu  
erzsi:karog
```

each line containing a user name, a colon character and the password in clear text..

If the type is `crypt` then the content of the file should be similar to

```
peter:rZ/dwMl2T8enI  
pityu:M3vyOhFxLXiaw  
erzsi:jIStnBGtFha7g
```

each line containing a user name, a colon character and the password encrypted. The encryption and the file format is the same that is used under UNIX so you can create and maintain such a file using the UNIX `htpasswd` tool. (The sample files above contain the same passwords, so you can try the 'crypt' authentication copy-pasting the content of the second example and using the passwords presented as clear text in the first example.)

The ASP edition gives you total freedom to define a Java class that can handle the login any way it wants. All you need is a Java class on the classpath that implements the interface `com.verhas.groowiki.edition.ASPAuthenticator` (defined in Groowiki-editions JAR file) and configure the name of the class in the configuration file under the key `authentication.authenticator.class`:

```
<authentication>  
  <authenticator>  
    <class>com.verhas.groowiki.authenticator.sample.Authenticator</class>  
  </authenticator>  
</authentication>
```

The interface is very simple:

```
package com.verhas.groowiki.edition;  
import com.verhas.authentication.AuthData;  
public interface ASPAuthenticator {  
    public boolean authenticate(AuthData authData);  
}
```

The used `AuthData` is also an interface that can be used by the class to access user name and password entered by the user:

```
package com.verhas.authentication;  
public interface AuthData {  
    public String getLoginPassword();  
    public String getLoginUsername();  
    ...  
}
```

There is a sample implementation in the groowiki source



```
com.verhas.groowiki.authenticator.sample.Authenticator:
```

```
package com.verhas.groowiki.authenticator.sample;
import com.verhas.authentication.AuthData;
import com.verhas.groowiki.edition.ASPAuthenticator;
public class Authenticator implements ASPAuthenticator {
    public boolean authenticate(AuthData authData) {
        return true;
    }
}
```

This very simple class simply lets just anyone to log in using any user name and password (even empty ones).

5.5. Plugin Configuration

Plugins have to be configured into Groowiki. Groowiki will not try to invoke a Java class or Groovy script unless it is configured in the configuration file. This ensures that only plugins that the system manager approved are executed.

Plugins are configured in the file `plugins.xml`.

```
<configuration>
  <plugins>
    <sample id="com.verhas.groowiki.sample-plugin" type="java">
      <class>com.verhas.groowiki.plugin.SamplePlugin</class>
    </sample>
    <sampleGroovy id="com.verhas.groowiki.sampleGroovy" type="groovy">
      <script>samplePlugin.groovy</script>
    </sampleGroovy>
  </plugins>
</configuration>
```

The plugins are configured in XML tags under the key 'plugins'. Each plugin should have a different key: 'sample', 'sampleGroovy'. You can use arbitrary names so long as long each plugin has its own tag name.

Plugins are identified by their 'id'. The 'id' is an attribute string and is be referenced in the velocity template calling the `$helper.invoke` method. Currently there are two type of plugins handled. These are 'java' and 'groovy'. Java plugins are written in Java, while Groovy plugins are written in Groovy. The Java plugins should be configured for their implementing class, while Groovy scripts should be configured for their script.

```
<configuration>
  <groovy>
    <mode>develop</mode>
    <classpath>
      <dir>/opt/groowiki/WEB-INF/groovy/classes/</dir>
    </classpath>
    <scripts>/opt/groowiki/WEB-INF/groovy/scripts/</scripts>
  </groovy>
</configuration>
```

The Groovy execution environment may be configured under the key 'groovy'. The directory where the scripts are SHOULD be configured. In addition to the Java classes that are available in Groowiki addition directories may be configured for the Groovy files that contain groovy classes. The mode (default is develop) may define whether the groovy interpreter is embedded and operated in develop more (debugging and continuous script recompilation) or deploy (no debugging or recompilation). Community edition supports only 'develop' mode. Professional, Enterprise and ASP editions support 'deploy' mode as well.



The actual configuration may contain more parameters than this documentation. The sample configuration files contains documentation on the possible keys and values as XML comments in it.

5.6. Log4j configuration

Groowiki uses the log4j logging subsystem. The configuration of this subsystem should be done via a separate configuration file which has nothing to do with Groowiki at all (see the documentation on how to configure the logging subsystem later). However the library SVNKit used by Groowiki is logging using the JDK logging subsystem. To unify the logging Groowiki redirects the logging of the SVNKit logging into log4j. The following few configuration keys have to be used in the `groowiki.xml` file (and *not* in the `log4j.properties` file) to configure how this redirection is performed.

Note that this logging is DEBUG log and is useful when you hunt a bug that you believe is lurking in the underlying networking layer or inside SVNKit. Therefore the default value for these values is `false`: do not log anything. You may also see the default configured into the `groowiki.xml` file something like

```
<svnkit>
  <log>
    <all>false</all>
    <exceptions>false</exceptions>
    <messages>false</messages>
    <network>false</network>
  </log>
</svnkit>
```

this is equivalent to the default value, which is when this xml fragment is not present in the configuration.

- `svnkit.log.all` switch SVNKit debug logging on or off. False means that there will be no logging and true means that there will be logging. Also if this key is defined then the value of this key will be the default value for the following configuration keys: `svnkit.log.exceptions`, `svnkit.log.messages` and `svnkit.log.network`. In other words you can
 - switch on or off all SVNKit logging or you can
 - switch exception, message and network traffic logging on individually, or
 - switch on all logging and switch off exception, message and network traffic logging individually.
- `svnkit.log.exceptions` log SVNKit exceptions into the log. This configuration key has nothing to do with other Groowiki exception logging, only those exception logs that come from the SVNKit library.
- `svnkit.log.messages` log normal log messages from the SVNKit library into the log file.
- `svnkit.log.network` log the network traffic that SVNKit performs. Note that this logging may generate a really huge amount of log. Use this with caution.

In addition to configuring Groowiki you also have to configure the underlying logging system. Groowiki uses the apache log4j logging and it is configured using the file `/opt/groowiki/WEB-INF/classes/log4j.properties` (as a matter of fact any `log4j.properties` fits that is on the classpath).



A sample content of the `log4j.properties` file:

```
# the root logger log level and the name of the appender
log4j.rootLogger=DEBUG,debuglog

# debuglog appender type
log4j.appender.debuglog=org.apache.log4j.DailyRollingFileAppender
log4j.appender.debuglog.DatePattern=.yyyy-MM-dd
log4j.appender.debuglog.File=/opt/groowiki-1.0.0/gw.log
log4j.appender.debuglog.Threshold=debug
log4j.appender.debuglog.layout=org.apache.log4j.PatternLayout
log4j.appender.debuglog.layout.ConversionPattern=%d [%t] %-5p %c (%F:%L) - %m%n
```

You are free to configure the logger (see the documentation of `log4j` on how to). The most important step however is to alter the

```
log4j.appender.debuglog.File=/opt/groowiki-1.0.0/gw.log
```

so that it points to a file (need not exist, it will be created) that will hold the log file. Note that the user running the Tomcat server should be able to write the directory where the log file is to be created. In the examples above we assumed that the application Tomcat is executed by the user `tomcat` in the user group `users`. Still using the example above you can check:

```
# ls -ld /opt/groowiki-1.0.0/
drwxr-xr-x 5 tomcat users 312 2009-03-27 19:08 /opt/groowiki-1.0.0/
```

that the directory exists and is writable by `tomcat`.

6. URL Structure

Groowiki has a very well defined URL structure. When an application extends Groowiki it may rely on the structure of the URL. Whenever Groowiki has to display some information it does as a response to a GET http request. The URL should look something like this:

`http://host:port/application_path/command/path:revision?params`

As an example:

`http://localhost:8084/gw/view/folder1/a/sss:110`

the URL will display the content or the properties of the directory or file 'sss' that is in the directory 'folder1/a'. The revision it will display is 110.

The parts of the URL are the following:

- **host** is the name of the host.
- **port** is the number of the port the servlet container or application server is listening to
- **application_path** is the path to the Groowiki application. This is usually configured in the servlet-mapping tag of the `web.xml` file. This may however depend on the servlet container type and version you use to execute Groowiki. This path also contains the part of the URL where you deployed Groowiki. (Note that version 1.0.0 could only be deployed as ROOT servlet.)
- **command** is the command you ask Groowiki to execute. The default command is 'view', but there are also predefined actions 'edit' and 'download' in Groowiki. In addition to these you can define more actions providing templates for them. 'view' being the default means that Groowiki



will perform the action 'view' if it finds an action that it does not know.

- **path** is the path of the actual document that Groowiki has to perform some action. As you can see in the example this is a full path inside the repository to the file or directory.
- **revision** is repository revision number. This part of the URL is separated by the previous part by a colon and this is optional. If the revision is not specified, or is not a number, or badly formatted in any way then the HEAD revision (the latest version) of the document will be used.
- **params** contain optional parameters in the usual cgi format: 'name=value&name=value' The use of these parameters is up to the plugin that implements an action. The only predefined parameter is 'ap=1' which is used by the built-in action 'view'. If this parameter is defined then the action will display all svn properties including those that contain a colon in their names. Such properties are usually svn or Groowiki internal properties.

Whenever a modification has to be done to the repository Groowiki should be queried with an http POST request. In this case the URL has the same structure, however some of the parameters, namely command and revision are ignored. Simply to say whenever a html form is created in a template the action URL should be '?'. Note that `enctype="multipart/form-data"` is also a must. Do not add encoding modifier to it.

7. Document upload

Groowiki handles document upload and modification. There are templates that display forms that let users modify document (file, or directory) SVN properties, set new properties, create directories and to upload new documents. Each of those forms use http POST method that are handles by Groowiki.

This section defines the form fields that Groowiki handles when a http POST operation is performed.

The encoding of the file should be `multipart/form-data`. Groowiki does not handle `application/x-www-form-urlencoded` form data. The simplest way is to specify the form header as:

```
<form name="upload" action="?" method="POST" enctype="multipart/form-data">
</form>
```

The fields that may be present in the form are the following:

- `logm` should contain the log message. Each form post modifies the SVN repository creating a new revision. This is the message that appears in the history of the SVN repository.
- ~~`path` should contain the path of the directory where the modification take place. Each POST operation should refer to a path using this parameter and all file and directory names are relative to this path.~~ This parameter is ignored in versions after 1.0.1 Instead the http request URL is used to determine the path.
- `nameXX` contains the name of the file or directory to be created uploaded or modified.
- `fileXX` contains the file uploaded. If the parameter `nameXX` is missing then last part of the path of the `fileXX` is used instead.
- `dirnXX` contains the name of the directory to be created.
- `pronXX.YY` contains the name of a property. The property will belong to the file or directory named by the parameter `nameXX` or `dirnXX`. `YY` is a serial number for the property. This way there can be several properties set for a file or directory when the file is uploaded or directory is



created or some time later when they already exist.

- `provXX.YY` contains the value for the property for which the name was specified in the parameter `pronXX.YY`

The parameters above contain `XX` and `YY` in their notation. These values are numbers. For example a form may contain the parameters (`path`, `logm`, `name0`, `file0`, `dirn1`, `pron1.0`, `prov1.0`). The numbers are not used for ordering the parameters. They are only to identify the pairing of the parameters. You can choose arbitrary numbers. The only requirement is that `nameX` belongs to file `X` and property name `pronX.Y` belongs to property value `provX.Y`

8. Templates and CSS

Groowiki creates the web pages that it displays in the users' browser from velocity template files. The location of the templates is configured in the configuration key `velocity.templates.root`. This directory contains the general VM files and there should also be some subdirectories that contain specific templates. ~~The current implementation (1.0.0) contains the CSS definitions included by the templates in the `template-style.vm`.~~ The current implementation (1.0.1) contains the CSS definitions in the file `'css/default.css'`. If you want to change the appearance of Groowiki you need to change only this template.

8.1. Business Type

Groowiki work with files that have properties. Any file can have any number of properties so long as long each property has a name. Therefore, as an example, you can store the issuer, recipient, issue date, complete date, payment date, payment type, netto amount, tax and so on for invoice documents. Whenever you upload the scanned image of an invoice you can press the link to edit the meta data and you can name each property and the value.

After a while it becomes cumbersome, and it is also error prone to type repeatedly the names of the properties. To ease data recording Groowiki lets you define the business type of a document, be it file or directory. When a document has a business type Groowiki knows what properties the document should have (the business type definition in the configuration file defines it) and will render a special form that contains fields specially for those properties that are needed for the type. This way you can upload the scanned invoice image and after clicking on "edit properties" you get a screen that displays input fields tailored for the special need of an invoice. Date fields will have click-able date selector links, numeric fields will format the numbers to thousands and so on.

Business types are defined in the configuration file and in some editions of Groowiki in files in the repository. The business type of a directory is defined by the property `'groowiki:businessType'`, the property should contains the name of the business type, in case of an invoice directory (whatever that may mean) the string `'invoice'`.

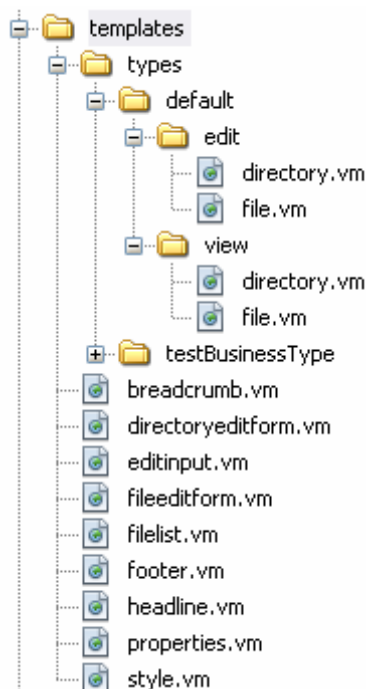
The business type of a file may also be defined by the property `'groowiki:businessType'`, but if it is not defined by this property then the property `'groowiki:containedBusinessTypes'` of the containing directory will be used. If neither property is defined then the `'default'` business type is used, which does not know anything about the required properties.



You can also edit a document's properties if it had the business type 'x' even if it does not appending the CGI parameter 'bt=x' at the end of the URL. This way you can, for example, have invoices, contracts, agreements, and you may have an extra business type 'genericDocument' that is common for each and lets you edit registry number of the document without setting it's business type to 'genericDocument'.

8.2. Action/Business Type Templates

If you look at the distribution of Groowiki 1.0.0 you can see the following files:



You can see the general VM templates like `style.vm`, `properties.vm` and other files in the root directory of the templates. These are templates that are included by other templates and contain CSS or html code that are general and used by many other templates.

The directory 'types' contains the templates that are specific to business types. Recall that each directory or file in Groowiki can have a business type. The business type is stored in the svn property `groowiki:businessType`. When a directory or file is displayed then Groowiki will use the template that fits the business type found in that property of the file or directory.

More precisely let's say that we invoked Groowiki with the URL

`http://localhost/gw/doit/dir/file_or_dir`

In this case groowiki will have a look at the file or directory 'dir/file_or_dir' in the repository and fetch the SVN property `groowiki:businessType`. Let's say that the value of this property is 'my_special'. Groowiki will try to open the template `types/my_special/doit/file.vm` if `file_or_dir` is a file and `types/my_special/doit/directory.vm` if `file_or_dir` is a directory.

This way one can implement different templates that can be selected based on the business type and the actual command. If some of the templates are missing then Groowiki will try to use another template as best it can guess instead of the one missing.

The formal algorithm of template search is the following:

1. Look for the template `types/'businessType'/'action'/[directory|file].vm`
2. If the template does not exist then try to find a template based on the inherited business types. This step is implemented in version 1.1.0 and later.
3. If the template does not exist then try `types/'businessType'/view/[directory|file].vm`
4. If the template does not exist then try `types/default/'action'/[directory|file].vm`
5. If the template does not exist then try `types/default/view/[directory|file].vm`

Note that the template searched for in the fourth step should exist as this template is part of the standard installation.



Using this information you can already create applications simply crafting templates that display different edit forms and view pages without any programming. All you have to do is create templates for the different business types and actions and use the POST/upload feature of Groowiki. But if even that is too much then starting with version 1.1.0 you need only define the business type in the configuration and Groowiki will use a default template that matches the fields of the business type.

8.3. Default Business Type Handling


Whenever Groowiki sees that you want to 'view' or 'edit' a directory or file that has business type (or if you define the business type on the cgi parameter 'bt') then it will display a special web page that fits the parameters of the business type unless you provided a customized template. The specialized page for edit will contain fields that match the type of the fields that are defined in the business type. Our favorite example is the business type 'invoice'. It contains all properties that are relevant for an invoice. When the properties of a document that belongs to an 'invoice' business type are displayed the following table will appear.

nuf_older/table_valign.html

PROPERTIES	
Issuer	
Recipient	
Invoice Identifier	
Issue Date	2009-06-03
Complete Date	2009-06-06
Payment Date	2009-05-29
Payment Type	cash
Description of the Invoice	
Netto Amount	
Value Added Tax	
Brutto Amount	
Is the Invoice paid	
Comment	

All these properties will be displayed even though the Issuer, Recipient and some other properties are not defined for the document. When you edit such a document you will get something like:



PROPERTIES	
Issuer	
Recipient	
Invoice Identifier	
Issue Date	2009-06-03 select
Complete Date	2009-06-06 select
Payment Date	2009-05-29 select
Payment Type	cash 
Description of the Invoice	
Netto Amount	
Value Added Tax	
Brutto Amount	
Is the Invoice paid	<input type="radio"/> yes <input type="radio"/> no
Comment	
Set	

In this form in the gray areas you can type the values for the properties and you can click on select to get a date selector popup. JavaScript will also help to format the numbers for the fields that contain money values and help avoid wrong format. You need not remember what properties an invoice usually have.

8.4. General templates

The general templates are in the directory configured by the configuration key `velocity.templates.root`. These are the following (release 1.0.0):

- `breadcrumb.vm` contains the html text for the top line path that contains each directory down to the actual one displayed and that helps the user to navigate upward in the directory structure.
- `directoryeditform.vm` displays the form that is used to alter a directory.
- `editinput.vm` displays the fields of a file or directory title and description to edit.
- `fileeditform.vm` contains a html form that should be used to set the properties of a file. At a time you can set at most 10 properties. Have a look at the template how to change this property (hint: search `$prN`).
- `filelist.vm` contains a table that list the files. This is used when a directory is displayed.
- `footer.vm` contains the footer copyright text.
- `headline.vm` contains the headline, usually the Groowiki logo.
- `properties.vm` contains html text that displays in a loop the properties. These can be properties of a file or a directory.
- `style.vm` contains the CSS code for the templates. This template is included by each other non-fractional template.
- `BusinessTypeEditForm.vm` contains an edit form for directories or files that have their business type defined. This file is included by `types/default/edit/file.vm` and



types/default/edit/directory.vm and let's you to avoid crafting special templates for each business type.

8.5. Template Parameters

Velocity templates used by Groowiki rely on the parameters that Groowiki provides. In this section we list all the parameters that are available in the templates (as of version 1.0.0):

- `descriptionNotSpecified` true if the description of the document to be displayed is not defined and false otherwise
- `titleNotSpecified` true if the title of the document to be displayed is not defined and false otherwise
- `description` contains the description of the document
- `title` contains the title of the document
- `parentsExist` is true if there is a parent element of the document to be displayed and false if the document to be displayed is the root directory of the repository
- `parentList` contains a list. Each element of the list is a map with two values: `name` and `path`. The `'name'` element contains the name of the actual element, `'path'` contains the full path to that element. For example we want to display the document `'p1/p2/p3'`. In this case the length of the list `parentList` is 2. The first element contains `name='p1'` and `path='p1'`, the second element contains `name='p2'`, and `path='p1/p2'`. For more information on how to use this parameter have a look at the template `breadcrumb.vm`
- `UUID` contains the repository UUID. This parameter is available, but it is not used by standard templates.
- `root` contains the repository root. That is the directory where the repository is.
- `headRevision` contains the revision number of the head revision
- `helper` contains a helper object. The documentation of the use of the helper object is detailed in a separate section.
- `date` contains an instance of standard velocity `DateTool`. For more information see the standard documentation of the velocity tool `DateTool` on the web.
- `math` contains an instance of standard velocity `MathTool`. For more information see the standard documentation of the velocity tool `MathTool` on the web.
- `number` contains an instance of standard velocity `NumberTool`. For more information see the standard documentation of the velocity tool `NumberTool` on the web.
- `render` contains an instance of standard velocity `RenderTool`. For more information see the standard documentation of the velocity tool `RenderTool` on the web.
- `esc` contains an instance of standard velocity `EscapeTool`. For more information see the standard documentation of the velocity tool `EscapeTool` on the web.
- `list` contains an instance of standard velocity `ListTool`. For more information see the standard documentation of the velocity tool `ListTool` on the web.
- `sorter` contains an instance of standard velocity `SortTool`. For more information see the standard documentation of the velocity tool `SortTool` on the web.
- `iterator` contains an instance of standard velocity `IteratorTool`. For more information see the standard documentation of the velocity tool `IteratorTool` on the web.
- `files` contains a list of maps. Each map contains the following
 - `name` contains the name of the document to be displayed
 - `author` contains the author of the document to be displayed
 - `revision` contains the revision of the document to be displayed



- `commitMessage` contains the commit message of the revision when the entry last time was modified
- `date` contains the date of the entry
- `size` contains the size of the entry
- `hasProperties` is true if the document has properties and false otherwise
 - `relativePath` contains the relative path of the entry
 - `isDir` is true if the entry is a directory
 - `isFile` is true if the entry is a plain file
- `numberOfFiles` contains a number that is the number of the files in the directory currently displayed
- `numberOfDirs` contains a number that is the number of directories in the directory currently displayed
- `total` contains the number `numberOfFiles + numberOfDirs`
- `revision` is the revision of the document that was requested to be displayed.
- `actualRevision` is the revision of the document to be displayed. This may be different from 'revision'. For example the URL specifies that the document revision 110 should be displayed. However the document was modified in the revision 108 and next time only in the revision 120. The document displayed has the `actualRevision` 108 even though the requested revision was 110, because the version of the document at the revision 110 of the repository is the version of the document that was uploaded in the revision 108.
- `revisionSpecified` is true if the revision was explicitly specified in the URL
- `properties` contains a Map containing the properties and their respective values. If the optional parameter 'allProperties' is not specified in the URL then this map contains only the normal properties, otherwise it contains all the properties even the ones that contain a colon in their names.
- `allProperties` contains a Map containing the properties and their respective values. All properties are listed in this Map.
- `result` contains the text returned by the action object plugin.

8.6. Template helper class

An instance of the helper class can be accessed from the templates using the parameter helper. Currently there are only a few methods that can be used in this class. These are the following:

- **`String createLink(String action, String path, String revision)`** returns the URL of a link to the document that is identified by the path and the action 'action' is to be performed. If the revision is null or empty string then the URL will be created for the latest revision. (Since 1.0.0)
- **`String createFormOptionField(List<String> options, String actualValue)`** returns a string that contains html form option values with a 'selected' keyword in the option that contains the value `actualValue`. (Since 1.0.0)
- **`String createImageLink(String imageName)`** returns the URL of a link to an image. The images are stored in the `/images` directory of the web application Groowiki. (available since 1.0.1)
- **`Object invoke(String pluginName, String... params)`** invokes the plugin with the parameters. The name of the plugin has to be the string that is configured for the plugin in the configuration file in the 'id' attribute. The parameters can be arbitrary strings. The return value is up to the plugin. This is usually a string that gets into the output, but it may also be some



object that can have methods to further call from the template via velocity.

9. Defining Action Classes²

The use of actions is deprecated as of 1.2.0. It is still there but it is not supported. There is no known action class implementation that got into production. Instead of action classes use plugins.

For the sake of simplicity in the above examples we did not present the configuration of the actions. However if you look at the real `groowiki.xml` file you will see some code similar to this:

```
<actions>
  <default package="com.verhas.groowiki.types.default">
    <!-- note that there is no 't'. It is 'default'
         because 'default' is Java reserved word -->
    <file>
      <view class="ViewFile"/>
      <download class="DownloadFile"/>
      <edit class="EditFile"/>
    </file>
    <directory>
      <view class="ViewDirectory"/>
    </directory>
    <post>
      <view class="ViewFile"/>
    </post>
  </default>
</actions>
```

This xml fragment defines different Java classes that are instantiated and some methods are executed at different occurrences. You can define actions to be executed for the different business types. By default there are action classes defined for the default business type. You can redefine these classes bravely. The default implementations do not really perform any action and in case you configure a class that can not be loaded then Groowiki will load the default classes anyway (after logging the error).

There are Action classes defined for the default business type for actions performed with files, actions performed with directories and for POST operations. In case of POST operation directories and files may be modified together therefore these actions should be complex handling the situation.

To ease configuration the XML tag that holds the name of the business type may specify the package where the class files are. You may do this if the classes are all in one package or in sub packages of a single package.

In the attribute “class” of the tag that is the name of the action you have to define the Java class. If you did not define the package then you have to define the class with the full path.

Note that this is your responsibility to put the class and jar files containing your Java classes in the servlet container so that Groowiki can load them.

Action classes should implement the interface `com.verhas.groowiki.Action`. The easiest way to accomplish this is to extend the abstract class `com.verhas.groowiki.AbstractAction`.

2 Note that action classes and the API available for them is experimental and may change in the future in a backward incompatible manner.



For more information have a look at the JavaDoc of the interface `com.verhas.groowiki.Action`.

__END__

