Design: Kamajii

Linux Workshop – Summer 2003

Submitted to: Eddie Aronovich

Date: 31/8/2003

Submitted by: Dror Sherman and Netta Gavrieli

1 TOC

2 Features

Kamajii is an open-source project designed to enable users to easily and yet flexibly receive **user-defined** relevant web data from different websites which will be managed using an extendible configuration language. Its main goal is the ability of sending users the relevant information on a "push" basis, directly to their cell-phones or emails.

Kamajii will support the following features:

2.1 Architecture

2.1.1 Multi Platform

Kamajii will work both on Linux and on Windows.
It will thus be able to interact with different browsers (at least Mozilla and IE) in the process of site-seeing or during activation of the recording mechanism.
Kamajii Messenger scripts will be available to both platforms (e.g. sendmail.pl will activate different code on Linux and on Windows)

2.1.2 Multi User

It will be possible to run Kamajii on a web-server (e.g. Apache). Its configuration will be managed using CGI scripts run by the web-server. They will allow new users to add themselves in order to receive notifications on the web-sites they are interested in. This is useful for example for users who do not have a permanent Internet connection.
For CGI details, see section 5.7.

Another way to run Kamajii with multiple users is on a server in which the users have shared directories. Several users with directories on the server can have the same instance of Kamajii monitor the web-sites that they choose.

2.2 GUI Based Configuration

Configuration will be saved in configuration files, which will be editable using a user-friendly GUI. For details, see section 5.3.

2.3 Site Definition Language

Kamajii will read site information from configuration files, which will be written in an extendible language: It will use a generic set of terms capable of describing login procedures, page entities to alert on, the format of the alert and more. For details, see section 5.8.5.

2.4 SSL Support

Kamajii will support both HTTP and HTTPS sites.

2.5 Cookie Support

Kamajii will support sending cookies to the sites as well as reading the Set-Cookie header and acting on it as a browser would.

2.6 Multiple Output Destinations

Kamajii will be capable of sending alerts using both e-mail and SMS. It will be modular enough to allow adding a new output destination easily. For details, see sub-section 5.8.2.2 (configuration file), 5.4 (configuration GUI) and the messenger section (4.3).

2.7 Language

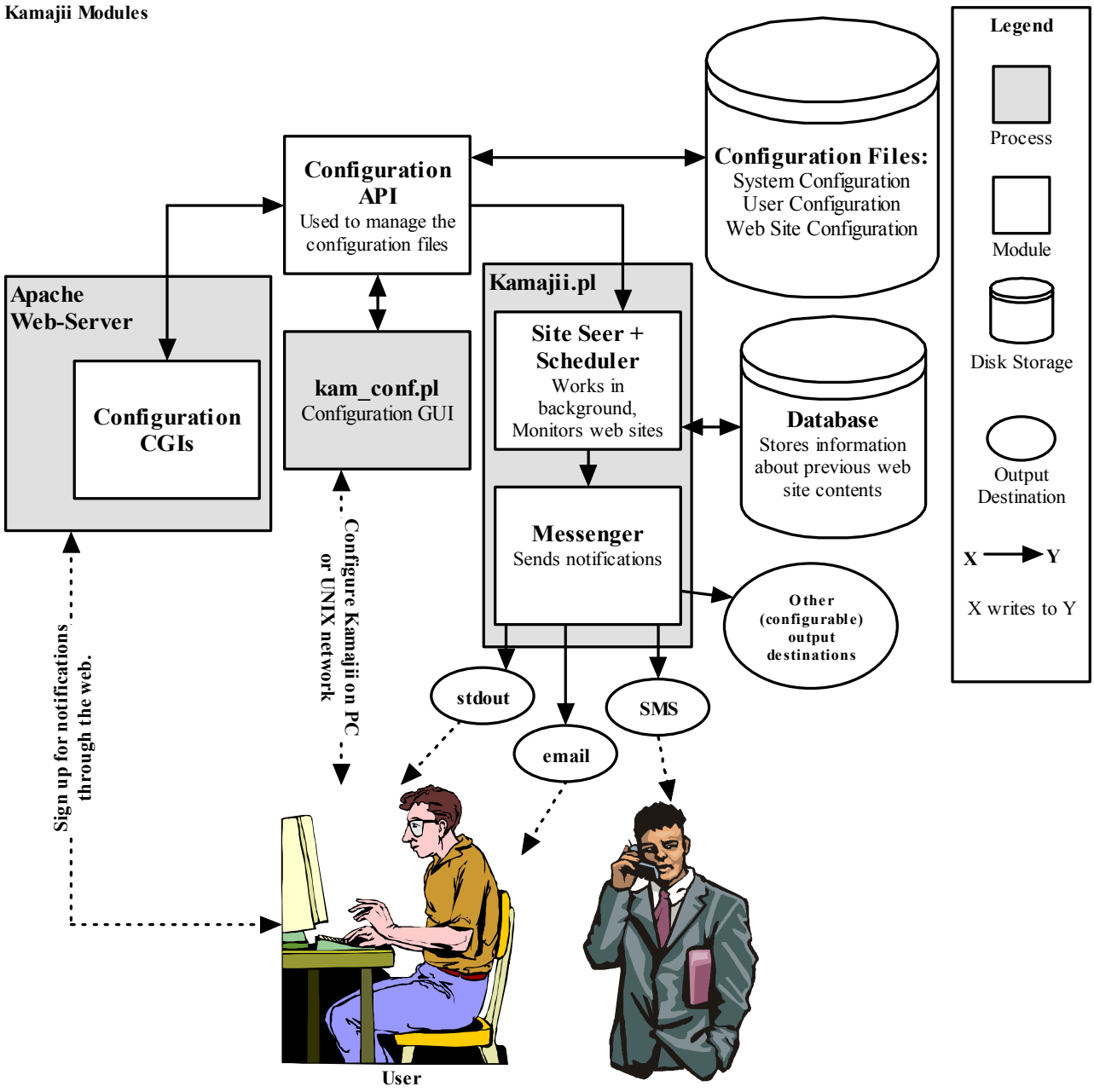
Kamajii will send notifications in English and in Hebrew. Additional languages will be added if necessary. This will be supported by the messenger and by the default output destinations.

2.8 Recording Mechanism (optional)

Instead of learning the site definition language described above, the user of Kamajii will have an option to browse the site himself and record his actions. The configuration file will be generated for him automatically.
This feature is very low-priority, and is not described in this document. It might be added, depending on the time we have.

3 High-Level Architecture

3.1 Diagram



3.2 Explanation

The diagram describes the relationship between the different modules of Kamajii, and between Kamajii and external programs such as the SMS sender, the email sender and the Apache web-server.

The core module of Kamajii is the Site Seer. The Site Seer is responsible for checking the sites (as requested from it in the configuration), comparing their contents with the contents that were saved in the database, and sending the output to the Messenger. The Site Seer is called by the Scheduler, which wakes it up every once in a while (as defined in the configuration).

The Messenger can send the output to various destinations. These destinations are simply command lines, so any program that can output information can be combined with Kamajii. By default, Kamajii will be able to send data via SMS, email and to the standard output. For example, with an SNMP trap Kamajii's output could be sent to a beeper.

Kamajii's configuration is saved in files, which define system configuration, user-specific configuration and web-site configuration. Files that contain sensitive information are encrypted. The configuration may be accessed only by the Configuration API. This API is used by the core modules of Kamajii (described above) and by the configuration modules to get and change configuration files.

There are two different ways to configure Kamajii. The GUI can be run on a computer that can access the configuration files directly: Either when a user installs Kamajii for his own use on his computer, or when Kamajii is installed for multiple users by the administrator of a UNIX server. The web-configuration CGIs can be used when Kamajii is installed on a computer that is running the Apache web-server. They enable new users to register themselves for Kamajii's notifications even if they cannot keep their PC connected to the internet at all times.

3.3 Programming Language

Kamajii will be written in Perl. We chose Perl for several reasons:

- Perl comes with an extensive set of libraries. This will minimize the time we will waste writing code that has been already written many times, leaving us more time to add the new and interesting features we want to support.
- Perl is a very powerful regular-expression parser. This will come in handy in Kamajii, which analyzes web-pages and looks for changes in them.
- Perl is cross-platform, so both the core and the GUI should work on Windows and on Linux, as well as on other platforms (although we only intend to test our code on Linux and on Windows).

Kamajii will use severl perl modules. To run Kamajii, the user is required to install these modules (if they were not installed previously – some are installed by default, depending on the OS and perl distribution). All modules can be found in CPAN and can be installed automatically on Windows with ActivePerl's package manager.
The following perl modules will be used:

- XML::Simple – Reads and writes the XML configuration files.
- libwww-perl – Used to manage HTTP connections
- HTML::Parser – Used to parse the HTML reply from the web server.
- Algorithm::Diff – Used to check for changes that need to be alerted on.
- Crypt::RSA – Used for public-key encryption of the user's password.

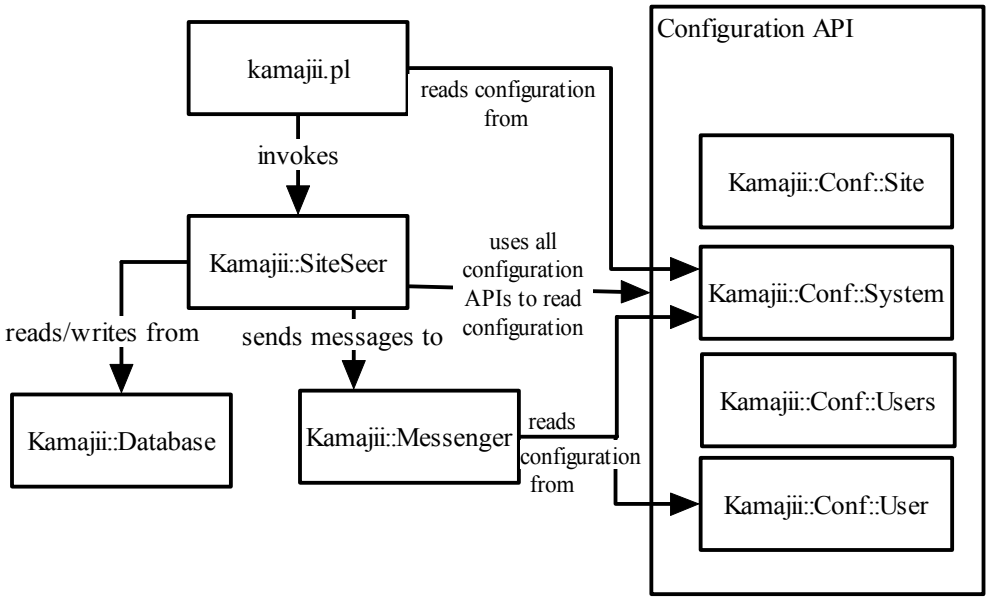
- Crypt::TripleDES – Used for symmetric encryption of the system's private key, the database and the user configuration files.
- Tk – Used for the Kamajii GUI.
- CGI – Used for the Kamajii Configuration CGIs.

4 kamajii.pl: Detailed Design.

kamajii.pl is Kamajii's main. It runs in the background, monitors sites and sends notifications about them. kamajii.pl consists of the kamajii.pl file (also known as the Scheduler) and of the following modules:

- Kamajii::Database – Handles the site information database.
- Kamajii::SiteSeer – Monitors sites.
- Kamajii::Messenger – Sends messages to multiple output destinations.

The relationships between these modules to one another and to other modules in the system are described in the following diagram:



4.1 Kamajii::Database

Kamajii's database is a big file that the Site Seer uses to save data about the contents of pages it monitors. The database is saved in the XML format. The XML database is not written to the disk as it is but is first encrypted with the system password (see 5.1).

4.1.1 File Format

The database is arranged with a hierarchy: The entire database is surrounded by <db>…</db> tags.

Under the db tags there can be <user name="some name">…</user> tags. These define the beginning of the configuration block of the user with the specified name.

Inside the users' blocks there can be <site name="some name">…</site> tags. The site names are the names of the site configuration files (not including the .xml).

Inside the site blocks are the parameters defined by Site Seer for this user and site, and their contents.

- Scalar parameters are defined as follows:
  <parameter name="some name">…value…</parameter>
- List parameters are defined as follows:
  <parameter name="some name">
  <row>…value1…</row>
  <row>…value2…</row>
  …
  </parameter>
- Table parameters are defined as follows:
  <parameter name="some name">
  <row>
  <column>…value1</column>
  <column>…value2</column>
  …
  </row>
  <row>
  …
  </row>
  </parameter>

4.1.2 Kamajii::Database::open <sys-password>

This interface creates a new Kamajii::Database object. It opens the database file, decrypts it and reads it into its internal structures. The database file is always located under the same directory as system.xml. If it does not exist it is created with the permissions defined in 5.8.1.1.
Usage example: $db = Kamajii::Database::open ("secret");

4.1.3 save

This interface updates the file on the disk with the changes to the database. It reformats the XML data, encrypts it and writes it to the disk.
Usage example: $db->save();

4.1.4 close [<save>]

This interface closes the database file, optionally saving the changes first. If the <save> argument is not passed, the database is not saved. If it is passed with the value 1, the database is saved.
Usage example:
$db->close(); # Discards the changes
$db->close(1); # Saves changes

4.1.5 get_parameter <user> <site> <parameter>

This gets the value of a parameter for a specific user's site. If the parameter, the user or the site does not exist in the database, the function returns the undefined value. Otherwise, it returns a reference to the parameter's value: A scalar parameter gets a scalar reference, a list parameter gets a reference to a list and a table parameter gets a reference to a list of lists.

4.1.6 set_parameter <user> <site> <parameter> <value>

This sets the value of a parameter for a specific user's site. If the parameter, the user or the site block does not exist in the database, the function first creates them. It then replaces the parameter's value with the given one, which can be a scalar reference, a reference to a list or a reference to a list of lists (a table).

4.1.7 delete_parameter <user> <site> <parameter>

This deletes a parameter from the database.

4.1.8 delete_site <user> <site>

This deletes a site block from the database.

4.1.9 delete_user <user>

This deletes a user block from the database.

4.1.10 delete_params_not_in <user> <site> <list of parameters>

This deletes all the saved parameters for a user and site, except the ones in the given list. It can be used by the Site Seer to delete all parameters that were not referenced in the last iteration, so as to avoid old irrelevant parameters filling up the database.

4.1.11 delete_sites_not_in <user> <list of site names>

This deletes all the site blocks for a user, except the ones in the given list. It can be used by the Site Seer to delete all sites that are still in the database even though their users do not want to monitor them any more.

4.1.12 delete_users_not_in <user list>

This deletes all the users from the database, except the ones in the given list. It can be used by the Site Seer to delete all users that do not exist any more.

4.2 Scheduler

The scheduler is a very simple module. It runs the main loop of Kamajii, which starts the Site Seer in specified periods of time. The scheduler is implemented in the Kamajii's main file – kamajii.pl.

An alternative to writing the scheduler could have been using the Linux cron, but we decided against it for better compatibility with different operating systems.

Flow of the scheduler

The Scheduler runs the following in a loop, until the $quit variable, which is configured by signal handlers, is defined:

- Save the current time into the variable $before_time.
- Run the Site Seer.
- Read the "cycle_time" parameter from the system configuration file (using Kamajii::Conf::System).
- Save the current time into the variable $after_time.
- Sleep for [cycle_time – ($after_time - $before_time)] seconds. If this number is negative, do not sleep.

4.3 Kamajii::Messenger

The Messenger exports one function, which is called by the SiteSeer. The name of this function is SendMessage and it accepts the following arguments:

- $system_config: An open Kamajii::Conf::System object.
- $user_config: An open Kamajii::Conf::User object.
- %messages: A hash containing pairs of :
    - A destination as a key (e.g. "SMS")
    - A list of messages as a value. A message is a hash that can have the following keys:
        - content: The content of the message.
        - urgency: The message urgency.

According to the $system_config and $user_config variables the function is responsible for fulfilling the following tasks for every list of messages:

- Sorting the messages according to their urgency levels.
- Concatenating the messages into groups that are no longer than the maximum length that a message to the relevant destination may have. The messages are separated by a seperator string. The maximum length and separator string are read from $system_config. If a message is too long, it is split to smaller messages.
- Eventually, the appropriate command line (as read from $system_config – defaults are detailed below) will be used in a loop to send each of the sorted messages.

Default supported output destinations (see 5.8.2.2 for command lines):

4.3.1 SMS

The sendsms is a perl script that can send SMS messages to any Israeli cellular phone. All four Israeli providers are supported: Cellcom, Orange (Partner), Pelephone and Amigo. The home-page of sendsms is: http://nadav.harel.org.il/software/sendsms/.

Kamajii's SMS sending is based on this script, with a slight enhancement: Since the number of SMS messages that can be sent with sendsms for each account is limited, each user will provide his own SMS-sending account with the company that provides his cellphone. sendsms currently only supports account configuration in a file in the home directory of the user that runs it.

To support defining the account in the command line, we will add the following command line switch to sendsms to override account definition in the configuration file: “-a <provider> <user> <password>".

sendsms uses the operators' (cellcom, orange, pelephone, ICQ) SMS sending dedicated web-pages contained in their sites. **The script produces the appropriate POST and/or GET HTTP requests**, fills the required site forms, and use the input parameters when needed. The final result of the HTTP request-response 'conversation' will be submission of the form which is responsible for actually sending the SMS to the user. In some cases since the SMS is sent from the user's account, each SMS is going to cost him the regular operator SMS fee: Even though sendsms tries to verify that sending a message is still free, it makes no guarantee that this does not change.

This method of sending relies on the operator's SMS webpage to stay constant. If the page is changed, sendsms will fail. Such a failure will translate to an error message appearing in the standard error, and the system maintainer that notices it will have to update the code. We do hope these web-pages will change rarely.

Possible enhancements:

- Since the process of parsing and analyzing the operator's website is somewhat similar to the process of analyzing the user's websites in search for new information, we may allow the user to use our recording mechanism (if it is implemented) in order to reconfigure the connection to the operator's site.
- In the future, Kamajii may be able to work with advanced web-services using interfaces such as SOAP, saving the need of dealing with HTML based parsing. This can be achieved with websites which expose web-services interfaces.

4.3.2 sendmail.pl

Kamajii's email sending is much more 'straight-forward':
On Linux, calling the script sendmail.pl will be simply equivalent to calling a standard 'mail' command line. On Windows sendsms.pl will translate to a bit more complex script which will be based on using the SMTP protocol and use some available modules (e.g. net::SMTP supplied by activePerl), which implement a client interface to the SMTP and ESMTP protocol, enabling a perl5 application to talk to SMTP servers, adhering to the concepts of the SMTP protocol as described in RFC821.

Command line:
sendmail.pl <mail server> <from address> <to address> <title> <message>

- The message may contain the sequence “\n”, which is converted to an actual line-break in the e-mail message.

4.3.3 Standard Output

Using the “echo” command, Kamajii’s messages can be written to the standard output. The standard output can be redirected to log files. These log files, can later on be screened using varied filters, mainly for the use of the Kamajii server maintainer.

4.3.4 And More…

Kamajii Messenger may be extended to include more output modules by simply adding command lines to the configuration (and sometimes writing a script to do the sending).

Some ideas:

- Sending mirs messages.
- Sending EMS and MMS messages.
- Sending Windows messages via "net send".
- Writing to specific log-files (without redirecting Kamajii's standard output).

4.4 Kamajii::SiteSeer

The Site Seer exports one function, which is called by the Scheduler. The name of this function is SiteSeer, and it receives one optional argument – a Kamajii::Conf::System object, opened for reading.

The Site Seer is responsible for scanning all of the users' requested sites (as written in the configuration), matching the required site data to the data written in the database, and, in case of a change – updating the database and sending the relevant notifications to the user by calling the messenger.

If a signal handler sets the $quit global variable, the Site Seer makes sure its database is consistent and quits at the first time it can.

The flow of the SiteSeer function is:

- Open the system configuration file if it was not passed as an argument.
- Use Kamajii::Database::open to open the database.
- Call Kamajii::Conf::Users::get to get the list of users and their information.
- For each user:
    - Use Kamajii::Conf::User::open to open the user's configuration for reading.
    - For each site defined in the user's configuration:
        - Use Kamajii::Conf::Site::open with the site name and the user name to open the site's configuration file for reading.
        - Make sure the user has defined all the parameters that are required for this site.
        - Go over the pages defined in the site's configuration file in order. For each page:
            - If there are run_foreach elements in the configuration, the page is handled once for each element of the list specified in run_foreach. All the lists are defined in advance (in previous pages), and the relevant parameters are configured for each iteration.
            - Replace any parameters that are referenced in the URL, header and content elements of the page.

- Build a request that asks for the URL with the given method, version, headers and content. Some headers are added automatically, unless a user-defined header overrides them:
    - Host: <site domain>
    - Connection: close
    - User-Agent: Kamajii
    - If a cookie was previously set that applies to this URL, the Cookie header is passed with the contents of the cookie.
- Send the request and read the reply from the web-server.
- Make sure the reply's status code matches the one defined in the return_code element.
- Keep the reply's cookie (for use in the next page).
- Go over the page_param elements. Each one adds a parameter to the list of parameters, as defined in the configuration file. Parameters that have the save_as modifier are also saved in another list (the save_as list), which contains their name in the database and their contents.
- Go over the diff elements, and check which of the differences applies. Diff is done using the module Algorithm::Diff. Accumulate all notifications that need to be sent to this user in a hash, which contains a list of notifications for each output destination.
    - Save save_as parameters to the database.
- Call Kamajii::Messenger to send all the notifications that were accumulated in the hash to the user.
- Remove unused sites and parameters for this user from the database.

- Remove non-existent users from the database.

4.5 Signal Handler

A signal handler will catch all signals that try to shut down the process. When such a signal arrives, the signal handler will print a message to the standard error ("Got signal <number>, quitting.") and will define a global variable called $quit.

When the Scheduler and Site Seer see this variable, they stop the process.

4.6 Error Management

All errors and warnings are printed to the standard error. Kamajii attempts to recover from errors and go on, but will quit when critical errors occur (for example, if the database is inconsistent).

5 Configuration: Detailed Design

Kamajii's configuration is divided into four types: The system configuration, which defines system-wide settings; the users list, which is the list of users of the system; the user configuration, which defines user-specific parameters; and the site configuration, which defines what to monitor on each site.

5.1 Security

Kamajii stores users' access codes for sites, so some of the files that it maintains can potentially contain sensitive information, such as ID numbers and bank accounts. A security mechanism must be implemented to protect this information.

5.1.1 System Password

Every time Kamajii starts it will open a popup window and ask for the system password. The system password will be used to encrypt and decrypt, using the TripleDES algorithm, files that may be used only by Kamajii. The following files will be encrypted with the system password:

- db – The database file.
- prk – Kamajii's private key.

5.1.2 System Public and Private Keys

Kamajii needs to be able to read user configurations while not allowing users to read each other's configurations. For this reason, the system will have an RSA private key (readable only by Kamajii) and a matching public-key (readable by all users).
The next section explains how these keys are used.

5.1.3 User Password

Every user who tries to open Kamajii's GUI or to use the CGIs will need to provide a password, which will protect the user's configuration file.

When a user needs to read/write his configuration file:

- Validate Password:
  - Encrypt the user's password via RSA with the system public key.
  - Open the user's file "user.key" and read its contents. If it is not the same as the encrypted password, the password is invalid.
- Use the (plain text) password to encrypt or decrypt the "user.xml" file with the TripleDES algorithm. The result is the xml user configuration.

When Kamajii needs to read a user's configuration file:

- Read password:
  - Open the user's file "user.key" and read its contents.
  - Decrypt these contents using Kamajii's private key. The result is the password.
- Use the password to encrypt or decrypt the "user.xml" file with the TripleDES algorithm. The result is the xml user configuration.

5.2 Configuration API

The configuration API will be a set of Perl modules (classes), which will be able to read and write to the configuration files. Kamajii::Conf::System manages the system configuration files, Kamajii::Conf::Users manages the list of users, Kamajii::Conf::User manages the configuration of a specific user and Kamajii::Conf::Site manages the configuration of a site.

All configuration files will be written in the xml format and parsed with the XML:: Simple package (a package that converts each configuration file into a tree that can be accessed directly by the modules that use the configuration).

5.2.1 Kamajii::Conf::System

This module manages Kamajii's system configuration. Changes to the system configuration file do not apply until the Kamajii process is restarted.
The system configuration can be edited only by the Kamajii's administrator: This should be enforced using the Operating System's file permissions.

5.2.1.1 Kamajii::Conf::System::open ["write"]

This function creates a Kamajii system configuration object. It reads the system configuration file ("system.xml", in the Kamajii root directory, which is read from the environment variable KAMAJII_ROOT). The system file is opened, and the object is returned to the caller.
The "write" keyword may be passed to the function. If it is passed, the file is opened with write permissions ("+<").
Usage example: $sysconf = Kamajii::Conf::System::open ("write");

5.2.1.2 close [<save>]

This function is run on an object that was created by Kamajii::Conf::System::open. It closes the handler to the system configuration file. Optionally, if the function gets 1 in its "save" argument, changes to the object are saved to the file before it is closed (will work only if the file was opened for writing).
Usage examples:
$sysconf->close(); # discard changes
$sysconf->close(1); # saves changes

5.2.1.3 save

This function saves changes to the Kamajii::SysConf object on the disk. It can work only if the file was opened for writing.
Usage example: $sysconf->save();

5.2.1.4 tree

This function returns the configuration as a reference tree (the format that the XML parser returns). Changing the tree actually changes the tree in the Kamajii::Conf::System object. These changes can be saved to the disk with the save() function.

5.2.2 Kamajii::Conf::Users

This module defines Kamajii's user-management APIs. These are a set of APIs that allow reading, adding, deleting or changing the details of users. They can be run by users who have permissions to access the users.xml file. Each API opens and closes the file internally.

5.2.2.1 Kamajii::Conf::Users::get <user-name>

If the user-name argument is passed, the function returns information about this user. Otherwise it returns information about all users.

The output depends on the context and on the input:

	Scalar Context	**List Context**
No user-name (all users)	Number of users.	List of user-info lists.
One user-name	Path of the given user.	User-info list of this user.

A user-info list contains the name of the user in $list[0] and the path in $list[1].

Usage examples:

```
$path = Kamajii::Conf::Users::get("chip");
print "Chip's path is $path\n";
@chip_info = Kamajii::Conf::Users::get("chip");
print "Path for $chip_info[0] is $chip_info[1]\n";
$n_users = Kamajii::Conf::Users::get();
print "Kamajii has $n_users users\n";
@all_users = Kamajii::Conf::Users::get();
foreach $user (@all_users) {
        print "Path for $$user[0] is $$user[1]\n";
}
```

5.2.2.2 Kamajii::Conf::Users::set <username> <path> …

This function gets usernames and paths, and sets the path for each one of the users. It can add new users or update existing ones. The input is a hash with the usernames as the keys and the paths as the data, or a list of usernames and paths.

Usage examples:

```
# Add the new user "greg" with the path "~greg/.kamajii" and update the path of
# the existing user "daisy" with the path "~daisy/info/.kamajii".
Kamajii::Conf::Users::set ("greg", "~greg/.kamajii", "daisy", "~daisy/info/.kamajii");
# Equivalent way to do this:
%users = (      "greg" => "~greg/.kamajii",
                "daisy" => "~daisy/info/.kamajii");
Kamajii::Conf::Users::set (%users);
```

5.2.2.3 Kamajii::Conf::Users::delete <username>

This function gets a username and deletes it from Kamajii.

Usage example: Kamajii::Conf::Users::delete("greg");

5.2.3 Kamajii::Conf::User

This is the user configuration module. It manages the configuration of a single user. This API reads and writes the XML user configuration to a file, taking care of encryption and decryption of the sensitive data.

5.2.3.1 Kamajii::Conf::User::open <user> <user-pass | sys-private-key> [write]

This function gets the name of a user and opens his user.xml configuration file. It finds the path to this file using Kamajii::Users::get. It returns an object that can be used to run the rest of the APIs in this section.

The "write" keyword may be passed to the function. If it is passed, the file is opened with write permissions ("+<").

The function uses the user password and the system private key for decryption of the file, as defined in 45.1.

Usage Example:
Chip wants to read and write his configuration.
$chip_conf = Kamajii::Conf::User::open ("chip", "secret", "write");
The site seer wants to read Chip's configuration.
$chip_conf = Kamajii::Conf::User::open ("chip", $sys_prk);

5.2.3.2 close [<save> <user-pass | sys-private-key> <sys-public-key>]

This function is run on an object that was created by Kamajii::Conf::User::open. It closes the handler to the user's configuration file. Optionally, if the function gets 1 in its "save" argument, changes to the object are saved to the file before it is closed (will work only if the file was opened for writing).
Usage examples:
$chip_conf->close(); # discard changes
$chip_conf->close(1, "secret", $sys_pubkey); # saves changes

5.2.3.3 save <user-pass | sys-private-key> <sys-public-key>

This function saves all changes to the .user configuration file associated with the object it is run on. It can work only if the file was opened for writing.
Usage example: $chip_conf->save($sys_privkey, $sys_pubkey);

5.2.3.4 tree

This function returns the configuration as a reference tree (the format that the XML parser returns). Changing the tree actually changes the tree in the Kamajii::Conf::User object. These changes can be saved to the disk with the save() function.

5.2.4 Kamajii::Conf::Site

The site's configuration is a written in XML, but it is essentially a simple script. It defines, or requires the user to define, several parameters, which are used to go over the required pages and compare the pages that are read from the internet to the data saved in the database.

5.2.4.1 Kamajii::Conf::Site::open <site name> [<user name>] [write]

This function reads the configuration of a site and returns a new object that contains it. The function first tries to read the specific configuration for the given user, and if it does not exist, it reads the general configuration of the site.
The "write" keyword may be passed to the function. If it is passed, the file is opened with write permissions ("+<").
Usage example:
$tau_grades = Kamajii::Conf::Site::open ("TAU-Grades", "chip");
$bank = Kamajii::Conf::Site::open ("Bank", "write");
$news = Kamajii::Conf::Site::open ("News-Site", "daisy", "write");

5.2.4.2 close [<save>]

This function is run on an object that was created by Kamajii::Conf::Site::open. It closes the handler to the site's configuration file. Optionally, if the function gets 1 in its "save" argument, changes to the object are saved to the file before it is closed (will work only if the file was opened for writing).
Usage examples:
$site->close(); # discard changes
$site->close(1); # saves changes

5.2.4.3 save <site name> [<user name>]

Writes the configuration into the configuration file of a site (the file is created or overwritten). If the user-name argument is passed, the configuration is written for this user. Otherwise it is written in the general sites directory. This API can work only if the configuration file was opened for writing.
Usage example: $tau_grades->save ("TAU-Grades");

5.2.4.4 tree

This function returns the configuration as a reference tree (the format that the XML parser returns). Changing the tree actually changes the tree in the Kamajii::Conf::Site object. These changes can be saved to the disk with the save() function.

5.3 Configuration GUI

This section describes the configuration GUI. The pictures should show the general idea, but the actual view may be different, depending on the widgets available in TK, which is the package we use to build GUI over perl.

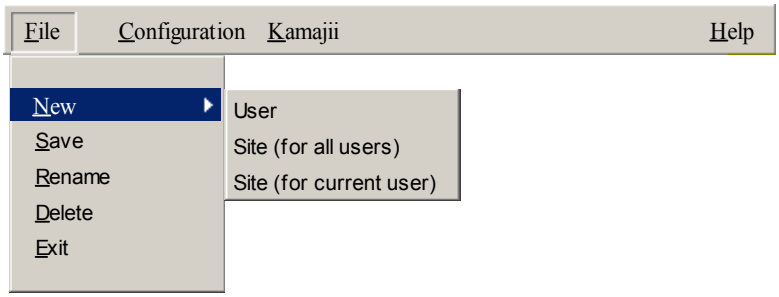
5.3.1 Executable

The GUI executable will be called kam_conf.pl.

5.3.2 Menus

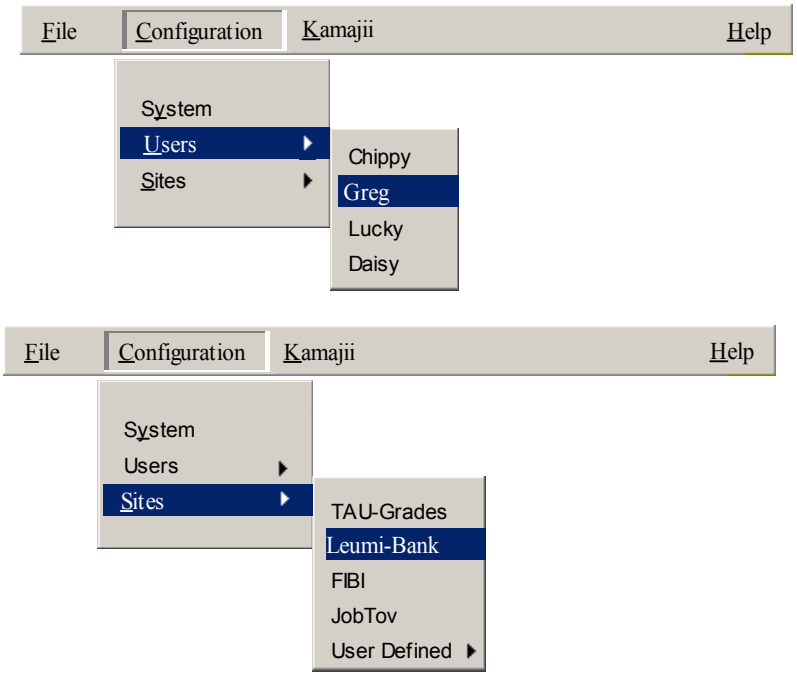
The GUI will have three menus: File, Configuration and Help.

5.3.2.1 File Menu



- New user – This option will be enabled if the user has permissions to write to the users.xml file. It gives the user the option to add a new user to Kamajii. When a new user is added, Kamajii updates the "users.xml" file with the new user data, and creates the user directories with some default configuration
- New site (for all users) – This option will be enabled if the user has permissions to write to the site_dir configured in system.xml. It gives the user the option to add a new site that all users have the option to monitor, either from an existing configuration file (to import downloaded or recorded files) or manually.
- New site (for current user) – This gives the user an option to add a site that is specific to him and only he can monitor, even without system permissions.
- Save – Saves the configuration file that is currently being edited. This option is disabled if the user running the GUI has no write permissions on the file he is viewing.
- Rename
    - If a user configuration file is being edited, renames the user. This option is disabled if the user running the GUI has no write permissions on users.xml.
    - If a site configuration file is being edited, renames the site configuration file. This option is disabled if the user running the GUI has no write permissions on the site he is viewing.
    - If the system configuration file is being edited, this option is disabled.
- Delete
    - If a user configuration file is being edited, deletes the user. This option is disabled if the user running the GUI has no write permissions on users.xml.
    - If a site configuration file is being edited, deletes the site configuration file. This option is disabled if the user running the GUI has no write permissions on the site he is viewing.
    - If the system configuration file is being edited, this option is disabled.
- Exit – Exit the GUI.

5.3.2.2 Configuration Menu

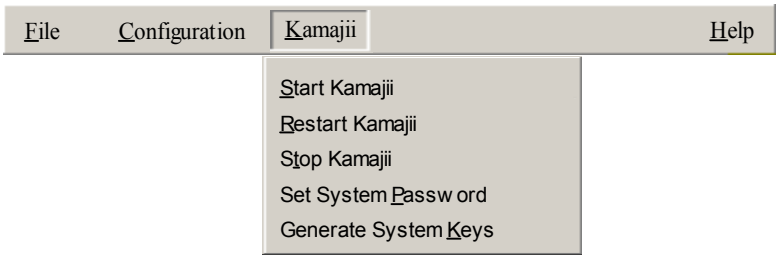


The configuration menu is used to select the configuration that is viewed and edited in the main window.

- System – view/edit the system configuration. If the user does not have permissions to read system.xml, this option is disabled.
- Users – Shows the list of users (retrieved using Kamajii::Conf::Users) and allows the user to pick one to edit. Only users that the user of the GUI has write permissions to their directories are shown in the list, so that normally each user will only see his own name. If the KAMAJII_USER environment variable is set, the user defined there will automatically be set to as the only one on the list (without checking all users).
  To view a user's configuration, the GUI requests the user is required to enter a password.
- Sites – Shows the names of all sites from the folder site_dir. Choosing "User-Defined" shows the same list of users as above, and for each user the sites that are configured in his user_site_dir folder.

5.3.3 Kamajii Menu

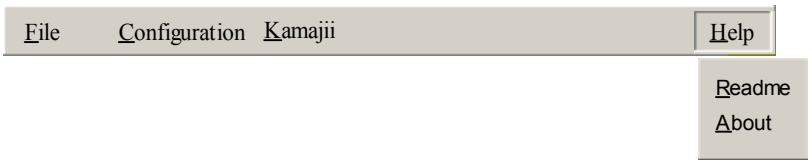
The Kamajii menu is used to perform administrative actions on Kamajii.



The Kamajii menu offers the following options:

- Start Kamajii – Starts kamajii.pl (if it is not running).
- Restart Kamajii – Stops and starts kamajii.pl (if it is running).
- Stop Kamajii – Stops kamajii.pl (if it is running).
- Set System Password – Asks for the current system password and for a new one, and converts the database and private key to use the new password. This option requires write permissions on the database and on the private keys.
- Generate System Keys – Asks for the current system password and creates new private and public keys. This function also creates a new user.key file for each user of the system and writes it over the existing user.key (if the permissions allow) or in a specified directory, in which case the person who ran this function is responsible for distributing the files to the users.

5.3.4 Help Menu

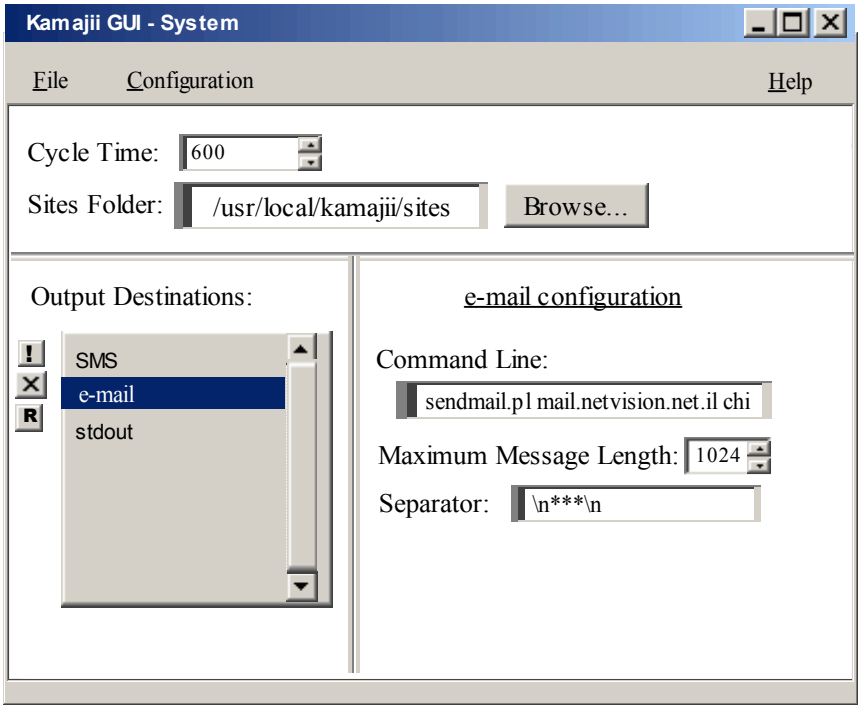


The help menu offers the following options:

- Readme – show documentation.
- About – show information about kamajii, including a link to the Kamajii website.

5.4 System Configuration

The system configuration is shown when it is chosen in the configuration menu. It allows the user with the right permissions to view or configure system.xml graphically.

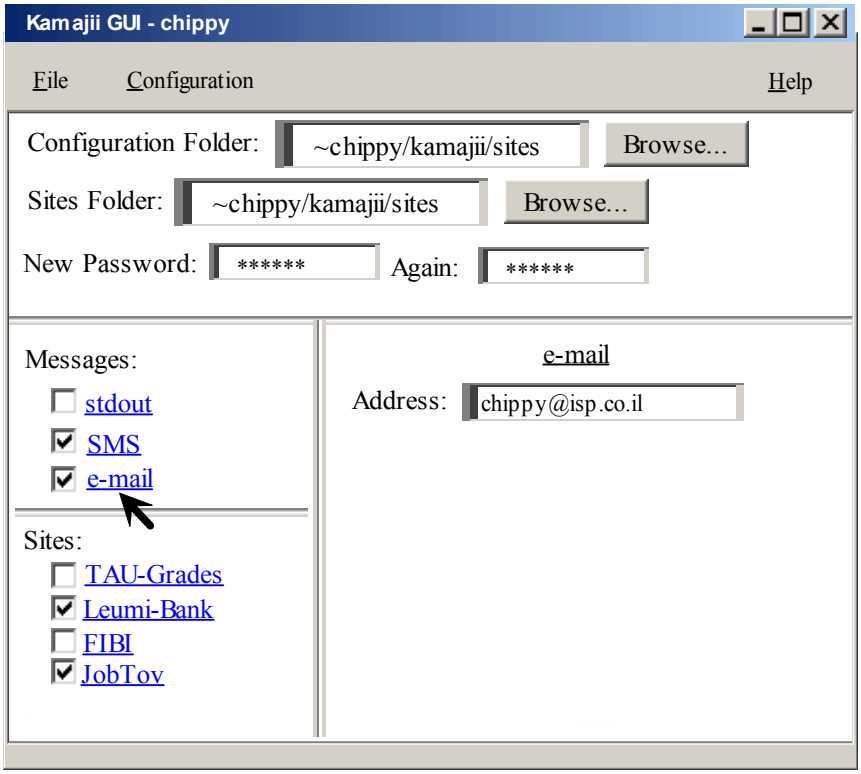


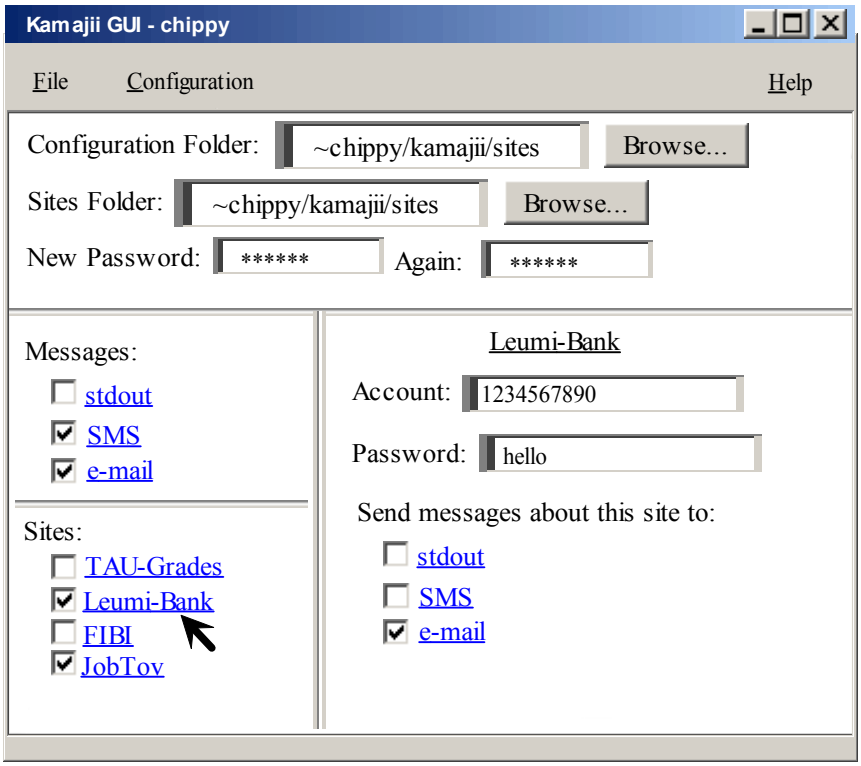
The screen is divided to three sections.
The upper section configures general system parameters.
The left section allows configuration of custom output destinations. New destinations can be added and existing ones can be deleted or renamed. Double-clicking on an output destination opens its configuration in the right section.
The right section shows the configuration of a specific output destination. The parameters that can be configured are the command line and the maximum length of a message.

The system configuration screen uses Kamajii::Conf::System to read and write the configuration.

5.5 User Configuration

The user configuration is shown when a user is chosen in the Configuration menu. It allows the user to configure his user.xml file graphically.





The screen is divided to four parts.

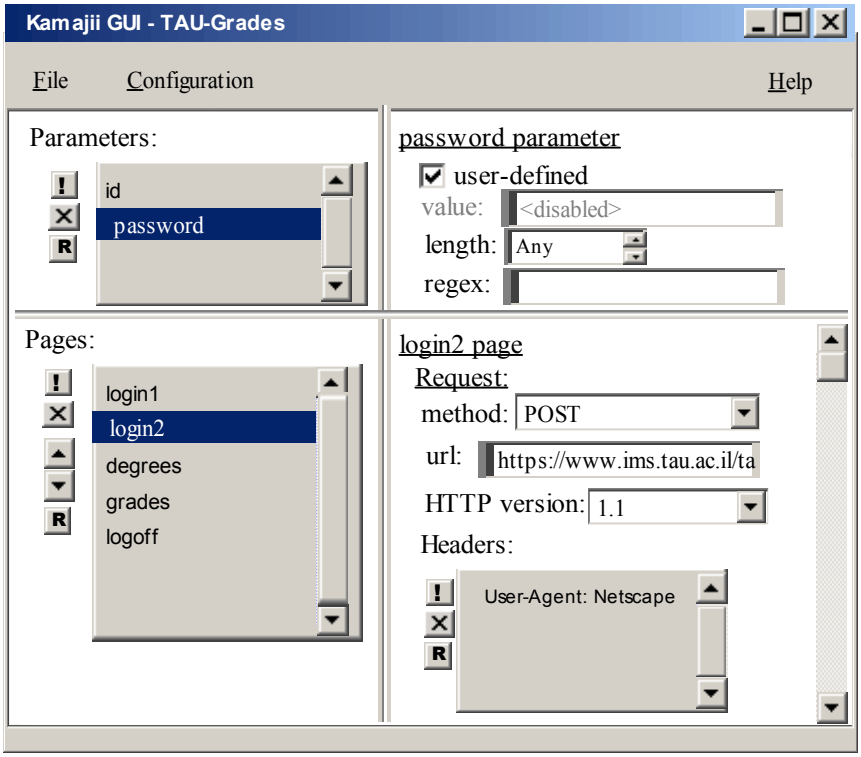
- The top section shows general configuration parameters for the user: The configuration folder is editable only if the user can edit users.xml (it is read and written with the Kamajii::Conf::Users API). The sites folder is the folder specified

in user_site_conf, and is read and written with the Kamajii::Conf::User API). The user can change his password from this section: changing the password re-encrypts user.xml with the new password and changes user.key.

- The Messages section shows a list of output destinations that the messenger supports (read from system.xml using Kamajii::Conf::System), and allows the user to decide which of these destinations is applicable to him. Clicking an output destination opens its configuration in the right part of the screen and allows the user to configure it. Each output destination needs different parameters, which are read from its command line in system.xml. The values of the parameters are read from (and written to) user.xml.
- The Sites section shows a list of sites from the user's sites directory and from the system sites directory. Clicking a site allows the user to configure the user-defined parameters of this site in the right part of the screen. The list of parameters is read from the site's configuration file, and their values are read from user.xml.

5.6 Site Configuration

The user configuration is shown when a site is chosen in the Configuration menu. It provides a graphical interface to the configuration of the sites.



The screen is divided to four parts:

- Upper left: Definition of the names of this site's parameters. New parameters can be added, and existing ones can be deleted or renamed. Double-clicking a parameter opens its configuration in the upper right corner.

- Upper right: Definition of a specific parameter. Either a value (if the parameter is not user defined) or a length and a regular expression (if it is).
- Bottom left: Definition of the pages that need to be requested in this site. A user can add, delete or rename a page, and can change the order of pages. Double-clicking a page opens its configuration in the bottom right corner.
- Bottom right: Definition of a specific page. The configuration shown here is just a GUI display of the page configuration described in 5.8.5.2.

5.7 Configuration CGIs

The configuration CGI is used by users who connect Kamajii through the Apache web-server. Since they cannot simply change their configuration using the GUI mentioned above, (it is not available on their computer), they need to pass every change they would like to make through Apache.

The set of operations possible through the CGI is only a sub-set of the operations which are enabled from the GUI. The reason for this is that whereas the GUI can be used by the system administrator as well, for needs such as adding a new site configuration (available to all users), or deleting a user from the "users.xml" file, the CGI is supposed to only be used by regular users, who we wouldn't want to allow to perform some malicious actions such as deleting other users from users.xml.

All of the actions described below will be implemented using the CGI.pm perl module.

The supported actions are:

5.7.1 New User Registration

- The user should first type the Kamajii entry URL on the Apache server (e.g. http://[hostname]/kamajii/).
- He then receives a page containing two links:
  - Login
  - Create a new kamajii account
- He chooses "create a new kamajii account".
- He now receives a form including two mandatory fields to fill:
  - Username
  - password
- After filling these fields, the kamajii CGIs check whether a user of that name is already defined, and if so returns the user to the previous screen, informing him that such a user name already exists in the system, so he should use a different user name.
- If kamajii finds that the user name is unique, it updates the "users.xml" file with the new user data, creates the user directories with some default configuration and returns the user to the Kamajii entry URL on the Apache, where he can choose the "login" option and change his settings.

5.7.2 User Configuration

- The user should first type the Kamajii entry URL on the Apache server (e.g. http://[hostname]/kamajii/).
- He then receives a page containing two links:
  - Login
  - Create a new kamajii account
- He chooses "login" and receives a form with two fields (as in the "create login" action).
- After the form is submitted, the CGI checks in the configuration files whether, such a user exists, and whether the password is correct.
- If the password is wrong, the user gets an appropriate error message, and can try login once again
- If the password is correct, kamajii sends a cookie to the user. The cookie is needed for authentication for the next stages, and to avoid a situation where a malicious user directly enters a configuration form without log in first.
- The next HTML contains two links:
  - User Configuration
  - User-defined site configuration **(This is just an optional low-priority feature, so it is not detailed below).**
- The user chooses the first link, so that the next HTML page which is sent to the user is similar to what is described in sub-section 5.5 dealing with the GUI interface. The user can fill in his user settings and submit the form. All of his changes are saved in the configuration files.

5.8 Configuration Files

The configuration files are written in xml. They are all surrounded by the tags <config></config>.

5.8.1 Directories

5.8.1.1 System Directory

The path to the system directory is defined in the environment variable KAMAJII_ROOT. The system directory will include the following files and directories:

File-Name	Permissions	Explanation	Generation
system.xml	rw-r--r--	System configuration	The default system file is copied upon installation. Can be changed from the GUI.
db	rw-------	Kamajii Database	Created automatically when it is first needed.
prk	rw-------	Kamajii Private Key	Can be generated from the GUI.
pubkey	rw-r--r--	Kamajii Public Key	Can be generated from the GUI (together with the prk).

sites	rwxr-xr-x	Directory with site configurations.	The default sites are copied upon installation. More sites can be added from the GUI.

5.8.1.2 User Directory

The path to the user directory is defined in the users.xml for each user (the variable KAMAJII_USER specifies the current username). The directory of every user will include the following files and directories:

File Name	Permissions	Explanation	Generation
user.xml	rw-r--r--	User-defined configuration	Default file is created when the user is defined. Can be changed from the GUI and from CGIs.
user.key	rw-r--r--	RSA encrypted password	Created with user.xml
sites	rwxr-xr-x	Directory with user-specific site configurations	Empty directory is created when the user is defined. Sites can be added from the GUI.

5.8.2 system.xml

The system file is configured by the "Kamajii Admin", i.e. the person who installs Kamajii (on a server or on a PC). Other users should have permission to read it but not to edit it.

The following elements are supported:

5.8.2.1 cycle_time

Default: <cycle_time>600</cycle_time>

Defines the number of seconds that the scheduler should wait between the beginning of the Site Seer's work and the next time it should be woken. For example, if the cycle time is set to 60 and the Site Seer was first woken up at 10:42:00, the next time the Scheduler will wake the Site Seer is 10:43:00.

The cycle may turn out to be longer than the specified time: If the Site Seer did not finish running when the Scheduler needs to wake it up, the Scheduler waits until it completes the previous run and then wakes it again.

5.8.2.2 output_dest

Default:

<output_dest name=”SMS” cmd=”sendsms.pl $phone$ Kamajii $message$” –a $company$ $account$ $password$ maxlen=130 seperator=”:” />
Mail server name and source address need to be defined for every installation.
<output_dest name=”e-mail” cmd=”sendmail.pl post.tau.ac.il kamajii@post.tau.ac.il $address$ \”Message From Kamajii\” $message$” maxlen=1024 seperator=”\n****\n”/>
<output_dest name=”stdout” cmd=”echo $message$” seperator=”:” />

Defines a list of output destinations that the system supports. Each destination has the following properties:

- name: Name that is used throughout the system for of the output destination.
- cmd: Command line to send a message to the destination. The command line can contain the parameter $message$, which is where the message to be sent is entered. It can also contain user-defined parameters surrounded by dollar signs. To write a dollar, use $$.
- maxlen: Length of the longest message that may be sent to this destination.
- seperator: A string that separates messages in case several messages are sent in a single command line.

5.8.2.3 sites_dir

Default: <sites_dir>sites</sites_dir>

The sites_dir is the directory in which all site configuration files are saved.
The value of this parameter can be one of:

- An absolute path.
- A relative path (from the directory in which system.xml is located).
- A path for a specific user (with a tilde).

5.8.3 users.xml

The users.xml file lists all users of kamajii. On a PC it will normally have one entry, but on a server it will have multiple entries. The permissions set for this file depend on the type of installation: Any user who can write to this file can register himself to get notifications from Kamajii.

The file itself is actually a list of user-names and passwords. The only XML element (other than <config>) is <user>, which can get a name (username) and a path (directory where this user’s “.user” file is located).

The path can be one of:

- An absolute path.
- A relative path (from the directory in which .system is located).
- A path for a specific user (with a tilde).

For example:
<user name=”chip” path=”~chippy/.kamajii” />
<user name=”greg” path=”/home/users/greg/config/.kamajii” />

5.8.4 user.xml and user.key

user.xml keeps the configuration for a specific user. This configuration may contain sensitive information such as passwords, so it is encrypted with a password, as defined in 5.1. user.key is used for decryption.

The following elements are supported:

5.8.4.1 user_sites_dir

Default: <sites_dir>sites</sites_dir>

Site configurations in the directory defined by user_sites_dir are used instead of the ones in the system sites_dir. This enables a user to create his own site configuration, for example by recording it.

The value of this parameter can be one of:

- An absolute path.
- A relative path (from the directory in which .user is located).
- A path for a specific user (with a tilde).

5.8.4.2 output

An "output" element is defined for each output destination, and contains user-defined variables that are used by the messenger when sending messages to the user through this output destination.

The name of the output destination is taken from the system configuration. It is indicated inside the tag, as the name. For example: <output name="SMS">…</output>
The parameters inside this block are specific to the destination and are taken from the output destination's command line (see 5.8.2.2).

For example:

```
<output name="SMS">
        <company>orange</company>
        <account>chip</account>
        <password>secret</password>
        <phone>054-123456</phone>
</output>
```

5.8.4.3 site

A "site" element is defined for each site that the user wants to be notified on. It contains the parameters that the site's configuration requires, as defined in each site's configuration (see 5.8.5.1).

The name of the site is the name of the site's configuration file, which is searched for first in the directory written in user_sites_dir and, if it is not found there, in the directory

written in sites_dir (in system.xml). The name of the site is indicated inside the tag, for example: <site name="TAU-Grades">…</site>.

In addition to the site-specific variables, the <output_dest> tag can be written in each site to define a list of output destinations for this site's messages. If this tag does not appear, all destinations for which the user has defined all necessary variables are used.

A sample site block:

```
<site name="TAU-Grades">
        <id>123456789</id>
        <password>1234</password>
        <output_dest>SMS</output_dest>
        <output_dest>e-mail</output_dest>
</site>
```

5.8.5 site/<site name>.xml

This file defines a site's configuration, which is used by the Site Seer to monitor the site.

The following elements are supported:

5.8.5.1 parameter

The <parameter> tag defines a parameter that can be referenced in some of the other elements.
The following may be defined in the tag:

- name: parameter's name (must be provided).
- value: Value of the parameter. If the value is not provided, it must be readable from the user configuration file.
- length (optional): Used for validating the parameter: The length of the value.
- regex (optional): Used for validating the parameter: A regular expression that the parameter must match.

For example:

```
<parameter name="id" length=9 regex="^[0-9]+$" />
<parameter name="password" />
<parameter name="pi" value=3.14 />
```

5.8.5.2 page

Most of the site's configuration consists of a list of pages. Each page defines a request to a URL and what to do with it. There are many elements that can appear under the page block:

5.8.5.2.1 Request Elements

The elements in this section determine how the request to the web-server is built.

method

The HTTP method that needs to be used to request the current page. The method can be either GET or POST, and defaults to GET.
For example:
<method>GET</method>

url
The HTTP URL that represents this page. The URL includes the protocol, domain name, path and query string.
The url may reference parameters that were defined in the <parameter> tags and parameters that were defined dynamically. To reference a parameter, its name is written between dollar signs.

For example:
<url>http://www.site.com/login?username=$username$</url>

http_version
The HTTP version that is used in the request: Defaults to "1.1".
For example:
<http_version>1.0</http_version>

header
This element may appear multiple times. Each instance is a header that overrides or is appended to the default headers that Kamajii sends with the HTTP request.
The header may reference parameters that were defined in the <parameter> tags and parameters that were defined dynamically. To reference a parameter, its name is written between dollar signs.
For example:
<header name="User-Agent">Netscape</header>

content
The content of the request that is sent to the page.
The content may reference parameters that were defined in the <parameter> tags and parameters that were defined dynamically. to reference a parameter, its name is written between dollar signs.
For example:
<content>id=id&password=$pass$</content>

5.8.5.2.2 Reply Elements
The elements in this section validate the reply to make sure it matches the expectations from it.

return_code
Default value: <return_code>200</return_code>

This is the status code that the request to the page must return. The return code can be an HTTP return code (which for our purposes is defined as any number between 100 and 999) or the number 0, which means that this page may return any code.

5.8.5.2.3 Error Elements

The elements in this section define what we about the site in case an error occurs.

error_page

The name of the page that is accessed when an error occurs in the current page. Can be any of the site's other pages, as long as a loop does not occur. If the error page is not defined, the flow does not continue to another page when an error occurs but quits checking the site.
For example: <error_page>logoff</error_page>

error_alert

If error_alert is defined, when an error occurs on this page, the message is sent to the user using the messenger. By default error_alerts are turned off, and the message is only written to the standard error.
For example: <error_alert />

5.8.5.2.4 Parameter Related Elements

The elements in this section define new parameters.

run_foreach

If there are one or more "run_foreach" elements in the configuration, the current page needs to be requested once for each element of the list it specifies. For every request for the page, the variable defined in "name" gets a different value from the list.
The list is the name of a parameter that was defined in the <parameter> tags or dynamically in previous pages.
For example: <run_foreach name="options" list="all_options">

page_param

This element contains a definition of a parameter out of something that is known in this page.

The name of the parameter is part of the tag, and may reference other parameters that were defined in the <parameter> tags or dynamically. To reference a parameter, its name is written between dollar signs.

The tag may contain the "save_as" modifier. If it is defined, when the block finishes processing, this parameter needs to be written to the database. save_as specifies is the name of the parameter in the database.

Another part of the tag is the type of the parameter, which defines how the variable is created. The type can be one of: "regex", "table", "db" or "modify".

The rest of the tag depends on the type.

The regex type: The parameter's value is calculated from the reply by searching it for a regular expression and returning the string that matches.

- regex: The regular expression to search (in perl regex format). May reference other parameters.
- appearance: Contains either a number or one of "list" or "uniq_list".
    - list: The parameter will be a list that contains all appearances of the regular expression in order.
    - uniq_list: The parameter will be a list that contains all the unique appearances of the regular expression, in an undefined order.
    - n (number): If the value is a number, the parameter will be a scalar that contains appearance number n of the regular expression.
- format: Each match of the regex with the reply that needs to be part of the variable is formatted according to this given format. The format can contain characters, variables that were defined previously and the variables 1, 2 etc, which are the strings that matched bracket #1, bracket #2 etc.

The table type: The parameter's value is calculated from the reply by searching it for HTML tables and returning the specified columns and rows of a specific table.

- number: The number of the table in the HTML (for example, 2 would return the second table).
- columns: Ranges of columns to put in the parameter. For example: "2,3,7-8,10-" means that the 2nd, 3rd, 7th, and 8th columns of the table will be read, and also all columns from the 10th to the last one. Column number 1 is the first column on the left, even if the reply is in Hebrew.
- rows: Ranges of rows to be put in the variable.

The db type: The parameter's value is read from the database.

- db_var: The name of the parameter in the database. May reference other parameters.

The modify type: The parameter's value is created from the value of another parameter by running a modifier on it.

- from_var: The name of the original variable. The name string may reference other parameters.
- modifier: Currently, the only supported modifier is strip_html, which removes all the HTML tags from the original variable and decodes the entities (characters like & that escape other characters in HTML). Other modifiers may be supported in the future.

Examples:

```
<page_param name="title">
        <type>regex</type>
        <regex>&lt;title&gt;(.*)&lt;/title&gt;</regex>
        <appearance>1</appearance>
```

```
        <format>$1$</format>
</page_param>
<page_param name="grades" >
        <type>table</type>
        <number>2</number>
        <columns>2,5</columns>
        <rows>2-<rows>
        <save_as>grades</save_as>
</page_param>
<page_param name="db_news">
        <type>db</type>
        <db_var>news</db_var>
</page_param>
<page_param name="clean_text">
        <type>modify</type>
        <from_var>html_text</from_var>
        <modifier>strip_html</modifier>
</page_param>
```

diff

Diffs check for differences between variables. A diff can contain the following elements:

- dataA: The data to compare with dataB. Can be a list/table parameter, a scalar variable, or a string that may contain scalar parameter.
- dataB: The data to compare with dataA. Can be a list/table parameter, a scalar variable, or a string that may contain scalar parameter.
- action: The action to perform if a difference is found. The action's type is either "error", which means that if a difference is found there is no use in continuing, or "notify", which means the messenger should send a notification string to the user. If the type is notify, the element contains the string. The string can contain variables, and values of the differing row in the data: An for the value of the n'th column of the differing row in A, and Bn for the value of the n'th column of the differing row in B. (Of course, when the difference is the addition of a row in one of the lists, there is no point in showing the value of this row on the other list). A notify tag can contain the modifier "urgency", which gets a number that is sent to the messenger to prioritize the message (0 – highest priority, 1 – lower priority, etc).
- check_for: Can be one of:
    - newA: A and B are lists or tables, and a new row was added to A that does not exist in B.
    - newB: A and B are lists or tables, and a new row was added to B that does not exist in A.
    - change: A row in A was replaced with a different row in B (if they are lists or tables), or A is different than B (if they are scalars).
    - AgtB: The number in a row in A is greater than the number in B (if it is a scalar) or the number in the corresponding row in B (if it is a table/list).

- BgtA: The number in a row in A is greater than the number in B (if it is a scalar) or the number in the corresponding row in B (if it is a table/list).
- AeqB: The number in a row in A is equal to the number in B (if it is a scalar) or to the number in the corresponding row in B (if it is a table/list).

For example:
Check for new grades:

```
<diff>
        <dataA>$grades$</dataA>
        <dataB>$grades_from_db</dataB>
        <action type=”notify” urgency=5>New grade in test $A1$: You got
        $A2$.</action>
        <check_for>newA</check_for>
</diff>
```

Alert when the bank account goes into overdraft:

```
<diff>
        <dataA>$bank_balance$</dataA>
        <dataB>0</dataB>
        <action type=”notify” urgency=0>Your bank account is in overdraft. Balance:
        $A1$.</action>
        <check_for>BgtA</check_for>
</diff>
```

6 Development Tools

6.1 Editors

Regarding the issue of choosing Kamajii's development tools, we tried to follow four major notions:

1. Perl is an interpreted language, and therefore the use of complicated development frameworks (such as Microsoft visual studio), responsible for compiling, linking, debugging, etc. which is more comfortable for compiler oriented languages such as C, C++ and Java (byte code), is not mandatory in our case.
2. Nevertheless, even in the case of interpreted languages, some basic debug options as well as basic file editing GUI, syntax coloring and more can be useful. If possible, we were interested in an additional tool which is a bit more complex than a basic editor.
3. Since Kamajii is planned to run on both operation systems - Linux and Windows – we will use different development tools for each.
4. Only development tools which are available for free (as in "free beer") are an option.

Based on the aforementioned we came to the following decisions:

- When working with Linux, we will use the traditional common UNIX editors: xemacs and vi (or vim). Unfortunately, more sophisticated tools such as ActiveState Komodo seem to cost money, so plain editors will have to do.

- When working with windows, and instead of editing our modules in notepad, we plan to use two products:
    1. Windows version of vim text editor – This tool supplies the perl programmer with a few handy features such as text wrapping, indentation control, line numbering, syntax highlighting, text completion, windows splitting, editing differences between files, text folding - all of these can be reached with some configuration changes.

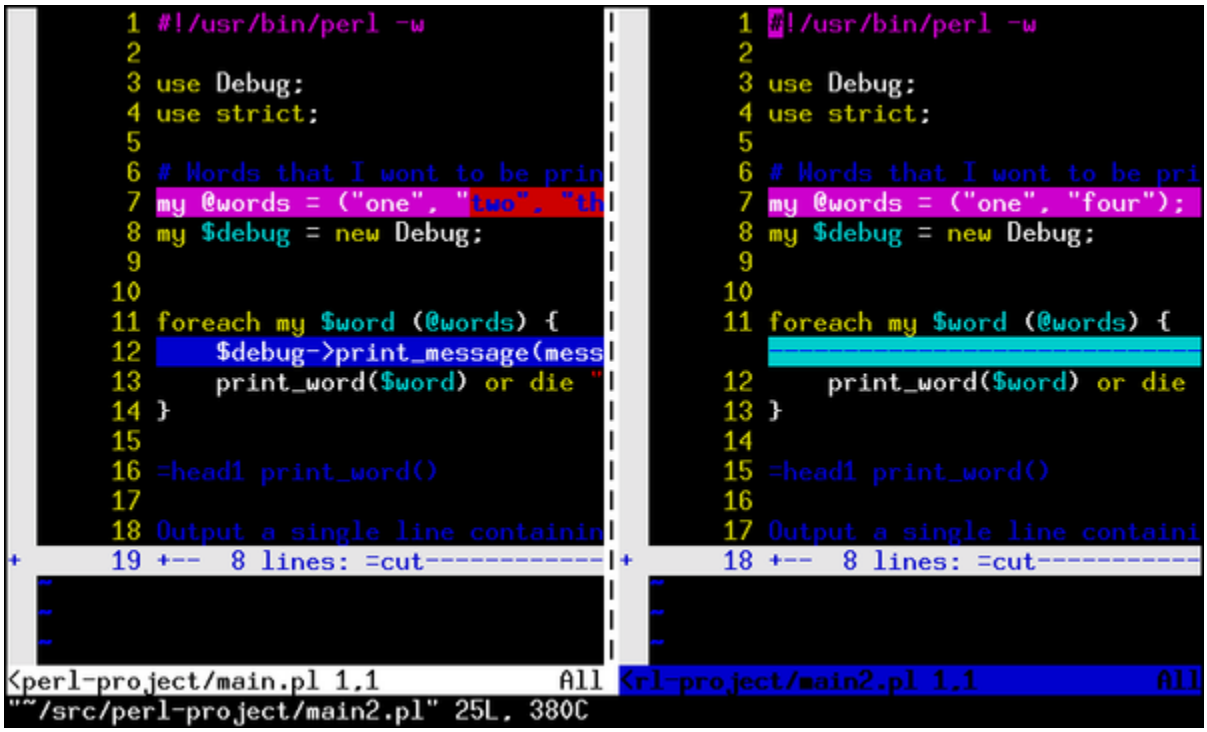


Figure:Vim's diff mode

    2. Open Perl IDE – this is an open source integrated development environment for writing and debugging Perl scripts with any standard Perl distribution under Windows 95/98/NT/2000. It differs from the relatively simple tools covered above since it handles debug, on-line help, multi-file view, and more. Among its features, some worth mentioning are:

        - Customizable Perl Syntax Coloring.
        - Erroneous script lines are listed on run.
        - Insertion and deletion of conditional breakpoints.
        - View and edit of variables in a watch tree.
        - Hint-Evaluation: Variables under the mouse-cursor will be evaluated and shown as hint.
        - Callstack info, List of loaded modules.
        - Flexible environment with dockable windows. Each configuration can be saved as an IDE-Desktop.
        - It is possible to use different desktop-settings for Edit, Run and Debug modes.

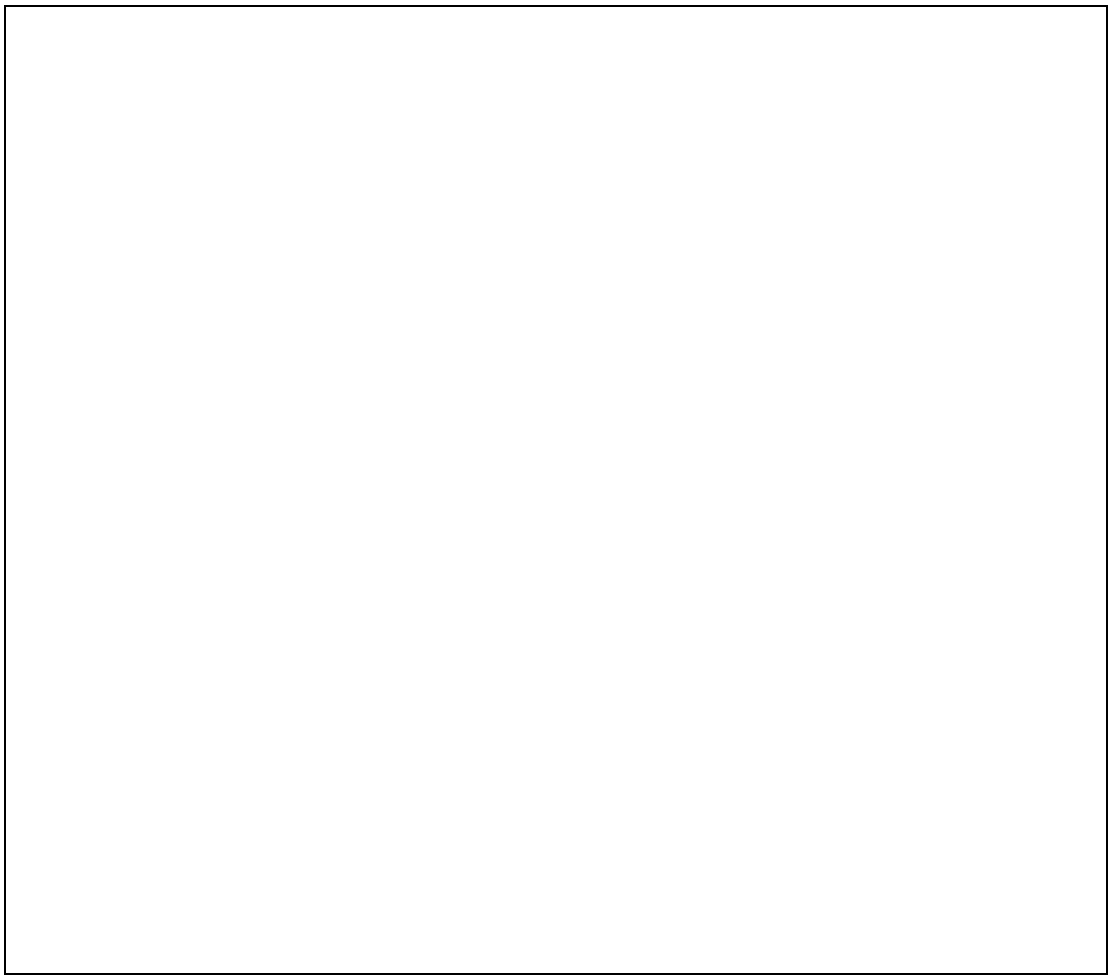


Figure: Open Perl IDE

6.2 Website

We will use sourceforge.net to host Kamajii's web-site.

6.3 Version Control

We decided to use CVS (Concurrent Versions System) for the following reasons:

- CVS is part of the GNU Project and is freely distributed in the open source community. (Unlike products such as Microsoft Visual Source Safe which cost money).
- It is widely used among open-source developers as can be observed in central open-source sites such as sourceforge.
- The major advantage of using CVS over the simpler tools like RCS or SCCS is that it allows multiple developers to work on the same sources at the same time.
- Besides simple version control check-in, it supports some handy features such as:
    - It provides change histories.
    - It supports rollbacks.
    - It provides release tagging.

- It backs up previous versions of source code in a compact format.
- It provides merging capabilities.
- It provides branching development.
    - There are three kinds of "branches in development" that CVS can support:
        - A developer working directory can be treated as a private branch.
        - A Development branch can be shared by one or more developers.
        - At release time, a branch is usually created for bug fixes.

7 Time Table

This section defines our time-table for completing Kamajii by the deadline on October 31.

Task	Responsible	Complete By
Web-Site		
Registering for and building the project web page.	Netta	13/09/2003
Updating the web page	Netta and Dror	Ongoing Task
Development		
Stage 1: Configuration classes		
Kamajii::Conf::System	Netta	20/09/2003
Kamajii::Conf::User	Dror	20/09/2003
Kamajii::Conf::Users	Netta	27/09/2003
Kamajii::Conf::Site	Dror	27/09/2003
Stage 2: Core Kamajii modules		
kamajii.pl Kamajii::SiteSeer Making changes to sendsms	Netta	11/10/2003
Kamajii::Database Kamajii::Messenger sendmail.pl	Dror	11/10/2003
Stage 3: Configuration		
Configuration GUI	Netta	18/10/2003
Configuration CGIs	Dror	18/10/2003
Recording Mechanism	Netta or Dror	Depending on available time.
QA and Conclusions		
QA	Dror and Netta	31/10/2003
Documentation	Dror and Netta	31/10/2003