

Prasi: Protocol for Remote Authentication and Shared Information

How to interface with the Prasi system using Perl
by Aurélien Botman

for version 1.2, September 2006

This document gives information about how developers should interface with Prasi from Perl CGI scripts.

Prasi is open-source free software and may be obtained from <http://prasi.sourceforge.net>.

Contents

1	Determining the scope	2
2	Configuration	2
2.1	Local services	2
2.2	Remote services	3
3	The interface	4
3.1	Authentication	4
3.2	Data manipulation	5
3.3	List of users	5
3.4	Direct linking	6
3.4.1	Authentication tasks	6
3.4.2	Administrative tasks	6
4	Simple examples	6
4.1	Local service	6
4.2	Remote service	7
5	Contact details	8
A	Note of major changes between 0.6.x and 1.0	8

1 Determining the scope

Firstly it should be determined whether the target Perl CGI script (the ‘service’) is local or remote. If there is a direct filesystem connection from one to the other, it is local (this includes connections over local networks). If there is no ‘local’ connection and the only method is to communicate via Internet protocols, the service is remote.

There is a strong preference to having a local service over a remote one. Firstly because connections over the Internet rather than LAN are always less secure and reliable. Secondly because some functionality available to local services are not available to remote services¹.

2 Configuration

2.1 Local services

Prior to interfacing with Prasi, the service needs to provide a configuration file inside the Prasi data-tree.

We shall take a service called ‘myservice’ as our example. An example configuration file would be as follows:

```
service:name:myservice
service:version:1.0
service:contact:myservice.admincontact@mydomain.com
service:url:/subpath/myserviceURL
default:myservice_pathdata
default:myservice_otherdata
suggest:myservice_otherdata:brown_hair
require:myservice_displaystyle
```

It can be seen that there are 4 types of configuration commands. Those starting with ‘service’ describe basic details about the service, such as its name, version, contact point and URL. All these entries are mandatory. Those starting with ‘default’, ‘suggest’, and ‘require’ are relevant to user profile data and are not necessary for basic usage of Prasi.

The ‘default’ keyword is followed by the name of a variable. This named data is set by the Prasi administrator at the time of configuring a user for a service. In the example given, the data ‘myservice_otherdata’ is set by the administrator when adding the new user to the service. The service can then read and also modify that data (see Section 3.2).

The ‘require’ keyword is also followed by the name of a variable. This named data, if it doesn’t already exist when the user accesses the service, is asked for. So in the example given, the data ‘myservice_displaystyle’ is such a data: when the user logs into that service, if the data is not already set they will be presented with a link to a form asking them to complete the information. Access to the service will be denied until the user has filled in that form. Once the form is filled in, the user is redirected back to the service automatically as if nothing had happened.

Any fields in the ‘suggest’ commands are default values for variables defined in the ‘default’ and ‘require’ commands, with which to pre-fill the web page form. So, in the example given, the data ‘myservice_otherdata’ will have the value ‘brown_hair’ pre-filled in the part of the form which asks the user/admin for that value. This can help admins and users quickly fill in the form using values regarded by the service as ‘reasonable’ (since it defined them).

¹At least, not in the beginning.

This all means that the service can rely on the information in the ‘require’ and ‘default’ blocks as being completed when the user accesses it, and there is some provision in place for admin-prefilled values and user-decided values.

The configuration file should be placed inside the Prasi data-tree, under the directory ‘services’, and named ‘myservice.cfg’.

Next, the service needs to know about Prasi. In the service script, simply ‘require’ the file ‘prasi.pl’. For example,

```
require "/var/www/prasi/prasi.pl";
```

where the path reflects the location where Prasi was installed into.

2.2 Remote services

Prior to interfacing with Prasi, the service needs to provide two configuration files: one inside the Prasi data-tree, and one with the prasi-client.

We shall take a service called ‘myservice’ as our example. An example configuration file (prasi-tree) would be as follows:

```
service:name:myservice
service:version:1.0
service:contact:myservice.admincontact@mydomain.com
service:url:http://www.myremotedomain.com/myservice/myservice.cgi
service:gateway:http://www.myremotedomain.com/myservice/prasi-client.cgi
service:domain:www.myremotedomain.com
service:code:MYSECRET
default:myservice_pathdata
default:myservice_otherdata
suggest:myservice_otherdata:brown_hair
```

It can be seen that there are 3 types of configuration commands. Those starting with ‘service’ describe basic details about the service, such as its name, version, contact point, URL, gateway, domain and secret code. All these entries are mandatory. The difference between gateway and URL are that gateway points to prasi-client.cgi on the remote server, always, whereas URL points to the remote service. The ‘code’ is a secret password known only to the administrator. It must be common in both configuration files. It should never be told to anyone. If it is suspected of being known, just change the code (in both configuration files), users will have to re-login but the communication will be secure again. See the Prasi project website for a page giving details on how this mechanism works.

The keywords starting with ‘default’ and ‘suggest’ are relevant to user profile data and are not necessary for basic usage of Prasi. Note that contrary to local services, remote services do not implement the ‘require’ keyword².

The ‘default’ keyword is followed by the name of a variable. This named data is set by the Prasi administrator at the time of configuring a user for a service. In the example given, the data ‘myservice_otherdata’ is set by the administrator when adding the new user to the service. The service can then read and also modify that data (see Section 3.2).

Any fields in the ‘suggest’ commands are default values for variables defined in the ‘default’ commands, with which to pre-fill the web page form. So, in the example given, the data ‘myservice_otherdata’ will have the value ‘brown_hair’ pre-filled in the part of the form which asks the

²Yet.

user/admin for that value. This can help admins and users quickly fill in the form using values regarded by the service as ‘reasonable’ (since it defined them).

This all means that the service can rely on the information in the ‘require’ and ‘default’ blocks as being completed when the user accesses it, and there is some provision in place for admin-prefilled values and user-decided values.

This configuration file should be placed inside the Prasi data-tree, under the directory ‘remote-services’, and named ‘myservice.cfg’.

The other configuration file example (the prasi-client one) would be:

```
#!/usr/bin/perl
package PRASI;
$pcook = "prasi_remoteauth";
$purl = "http://www.mydomain.com/prasi/";
$clientcode = "SECRET";
package main;
return 1;
```

where pcook is the remote cookie name, purl the URL of the prasi SERVER, and clientcode the secret password code mentionned earlier in this subsection. This file should be saved as ‘prasi-client-config.pl’ in the same directory as the remote service, on the remote server. ‘prasi-client.pl’ and ‘prasi-client.cgi’ should then be copied to this same directory. Finally, a language choice should be made and the appropriate language file copied as ‘prasi-client-lang.pl’.

Finally, the remote service needs to know about Prasi. In the remote service script, simply ‘require’ the file ‘prasi-client.pl’. For example,

```
require "./prasi-client.pl";
```

Your remote service should, in theory, now be configured to be used with Prasi.

3 The interface

3.1 Authentication

The function to call to check if a user is logged in is ‘prasi_login’. It must be called after printing the Content-Type and before doing any user-specific processing.

```
$user = prasi_login("myservice",0);
```

The first argument is the service name. The second argument is either 1 or zero. The function checks whether the browsing user is logged in yet or not. If the user is logged in, \$user will be a string containing the username of the person.

If the user is not logged in yet, if called with 0 as second argument then nothing happens and the function returns an empty string “”. Otherwise if called with 1, the function will re-route the user’s browser to a login screen. After logging in the user is redirected back to the script (so the script is re-loaded with the same parameters).

3.2 Data manipulation

To save information about the current user in that user's global Prasi profile, issue the following call:

```
prasi_save($username,$data_name,$data_content,$service_name);
```

where the user's username is held in `$username`. `$data_name` should be the name to save the information as for later retrieval. `$service_name` is the name of the service writing the data. To prevent different services from overwriting each other's data, the one restriction on the `data_name` is that it contain the name of the service within it. For example, a sensible scheme would be to prefix the name with the name of your service, for example 'myservice_number_of_visits'. `$data_content` is then obviously what to save. The string may contain newlines³.

It is strongly recommended to issue a call to `prasi_data` (see below) beforehand, to check that you are not about to overwrite information which already exists. `prasi_save` does absolutely no checking of previous data and overwrites information without asking. The overwritten data cannot be recovered and the only solution might be to reconfigure a user for the service adversely affected.

A call to `prasi_save` with an empty string for the data content will delete the profile file of that name. Limited checking is performed at present - only trusted services should ever be installed anyway.

To retrieve information about the current user from that user's global Prasi profile, issue the following call:

```
($content) = prasi_data($username,$data_name);
```

where the user's username is held in `$username`. `$data_name` should be the name the information was stored as in a call to `prasi_save()`. The data is returned in `$content` but note the brackets: the function actually returns an array pointer (if you know how to use perl arrays you can harness the additional power)⁴. The variable will be an empty string if an error occurred (for example, if no data was stored under that name). A 'chomp' operation is recommended before using the resulting data as there may be errant newlines.

You are not limited to data stored by your service; you can read in any data stored by any service provided you know the name the data was saved as.

Note that some non-service specific data is already stored by Prasi by default and may be accessed by any service, for example 'email', 'fullname', 'language' and 'ip'. 'password' is also accessible although that is a hash so is of limited interest. In the future the session key and expiry may also be available via this method, allowing you to re-write the prasi manager itself for example (that is already implemented as a service! You could roll your own if you don't like the default one...).

3.3 List of users

To obtain a list of all users on the prasi system, you can use the following call:

```
$list = prasi_userlist($service);
```

with `$service` being null. If a string is specified for `$service`, the call will return only those users who have access to the service of that name. (This call can be useful to find the list of prasi administrators, for example.) The returned list is newline-separated.

³Currently only local services - remote services will only be able to read/write the first line of the sent data.

⁴Again, remote services will only receive the first line of the data.

3.4 Direct linking

3.4.1 Authentication tasks

It is not necessary to redirect the user each time to the Prasi forms to log them in or out – the appropriate forms can easily be printed from the service.

To produce a login form, 4 fields are needed: username ('user'), password ('pw'), return url ('return') and the action 'login':

```
<form action="/prasi/login.cgi" method="post">
<input type=submit value="login">
<input type=text name="user">
<input type=password name="pw">
<input type=hidden name="return" value="/subpath/myserviceURL">
<input type=hidden name="action" value="login">
</form>
```

To produce a logout form, 2 fields are needed: the return url ('return') and the action 'logout':

```
<form action="/prasi/login.cgi" method="post">
<input type=submit value="logout">
<input type=hidden name="return" value="/subpath/myserviceURL">
<input type=hidden name="action" value="logout">
</form>
```

3.4.2 Administrative tasks

Services who read the prasi.admin status flag are able to call manage.cgi directly. One hidden option (added in version 1.2) is that appending closewindow=1 to the query string will result in the extra window being closed immediately once the action is finished (ie. the user is automatically returned to the calling service).

4 Simple examples

4.1 Local service

Configuration file 'services/localservice.cfg':

```
service:name:localservice
service:version:1.0
service:contact:localservice.admincontact@mydomain.com
service:url:/subpath/my-example-service.cgi
default:email
```

Service 'my-example-service.cgi':

```
#!/usr/bin/perl --
use CGI::Carp "fatalsToBrowser";
print "Content-type: text/html\n\n";
require "/var/www/prasi/prasi.pl";
```

```

my $user = prasi_login("localservice",1);
if(!$user) {print "No user is logged in.";exit;}
print "You are user '$user'. (so, correctly logged in)<br>";
my $email = prasi_data($user,"email");
print "Your email address is '$email'. (so, data fetching works)<br>";
my $count = prasi_data($user,"localservice-counter");
$count++;
my $status = prasi_save($user,"localservice-counter",$count,"localservice");
print "Counter is '$count' and status is '$status'.
      (so, data saving works, try reloading!)<br>";
exit;

```

4.2 Remote service

Configuration file 'remote-services/remoteservice.cfg':

```

service:name:remoteservice
service:version:1.0
service:contact:remoteservice.admincontact@mydomain.com
service:url:http://www.myremotedomain.com/remoteservice/my-example-service.cgi
service:gateway:http://www.myremotedomain.com/remoteservice/prasi-client.cgi
service:domain:www.myremotedomain.com
service:code:MYSECRET
default:email

```

Configuration file 'prasi-client-config.pl' (where 'prasi-client.pl' and 'prasi-client.cgi' and 'prasi-client-lang.pl' are located along with 'my-example-service.cgi'):

```

#!/usr/bin/perl
package PRASI;
$pcook = "prasi_remoteauth";
$purl = "http://www.mydomain.com/prasi/";
$clientcode = "SECRET";
package main;
return 1;

```

Service 'my-example-service.cgi':

```

#!/usr/bin/perl --
use CGI::Carp "fatalsToBrowser";
print "Content-type: text/html\n\n";
require "./prasi-client.pl";
my $user = prasi_login("remoteservice",1);
if(!$user) {print "No user is logged in.";exit;}
print "You are user '$user'. (so, correctly logged in)<br>";
my $email = prasi_data($user,"email");
print "Your email address is '$email'. (so, data fetching works)<br>";
my $count = prasi_data($user,"remoteservice-counter");
$count++;
my $status = prasi_save($user,"remoteservice-counter",$count,"remoteservice");
print "Counter is '$count' and status is '$status'.
      (so, data saving works, try reloading!)<br>";
exit;

```

5 Contact details

If you would like to contact us regarding this document for any reason, please do contact us via the facilities provided on our website (<http://www.sourceforge.net/projects/prasi>). Please do also contact us if you require further clarification concerning any issue of integrating Prasi into your Perl CGI scripts.

A Note of major changes between 0.6.x and 1.0

The user with username “demo” is now hard-coded in Prasi as the demonstration user. All Prasi applications should expect to be accessed in demo-mode by this user. They must however ensure themselves that no unnecessary priviledges are given to this user.

A further change is that new users are automatically added to the “prasi” service. Previously this step required a separate manual action from the Prasi administrator; now this action is no longer neccessary. However it is the responsibility of the Prasi administrator to manually remove access from this service if it is undesirable.