Jarvis Web Gateway

Working with Jarvis
(User & Technical Documentation)

Jonathan Couper-Smartt

jarvis@nsquared.co.nz

Abstract: The Jarvis Web Gateway is a lightweight web-service designed to give Rich Internet Applications a rapid, powerful and secure mechanism to access and update server-side databases.

Jarvis supports fetch, create, update and delete with additional support for custom plug-in features. Jarvis is based on the RESTful approach to http web services. It provides JSON, XML interfaces, with additional support for CSV downloads.

Jarvis works excellently with JavaScript web apps using ExtJS, Dojo and many other JS toolkits, and is also ideal for richer client applications such as those written in Adobe Flex.

Jarvis supports any database for which a Perl DBI driver exists. This include Oracle, PostgreSQL, MySQL, SQL Server, SQLite and more.

Table of Contents

1 Introduction....4
2 Licence....5
2.1 GNU Lesser General Public License....5
2.2 Clarification of Intentions....5
3 Installation....6
3.1 Unpacking....6
3.2 Install Additional Perl Modules....6
3.3 Web Server Configuration....6
3.4 Relocating Jarvis....7
3.5 Test....7
4 Application Configuration File....8
4.1 Introduction....8
4.2 <jarvis><app>....9
4.3 Configuration: format....10
4.4 Configuration: debug....10
4.5 Configuration: dump....11
4.6 Configuration: log_format....11
4.7 Configuration: page_limit_param, page_start_param....11
4.8 Configuration: sort_field_param, sort_dir_param....12
4.9 Configuration: dataset_dir....12
4.10 Configuration: default_libs....13
4.11 Configuration: method_param....13
4.12 Configuration: login....14
4.13 Configuration: database....14
4.14 Configuration: sessiondb....14
4.15 Configuration: exec & plugin....15
4.16 Configuration: habitat....15
5 Login Module Configuration....16
5.1 Standard Modules....16
5.2 Active Directory Login....16
5.3 Basic Auth Login....17
5.4 Database Login....18
5.5 None Login....18
5.6 Single Login....19
6 Special Datasets & Jarvis Login....21
6.1 Jarvis URLs....21
6.2 The __status Dataset....21
6.3 The __habitat Dataset....22
6.4 The __logout Dataset....23
6.5 Logging-In to Jarvis....23
6.6 Jarvis Error Responses....23
7 Datasets....25
7.1 Data Fetch Response Format....25
7.2 Dataset Definition....26

7.3 Dataset Parameters....27
7.4 Secure Dataset Parameters....29
7.5 Modifying Datasets – Single Modification....30
7.6 Modifying Datasets – Array of Modifications....32
7.7 Modifying Datasets – Array of Mixed Modifications....34
7.8 Before & After Statements....34
7.9 Failed Modifications....35
7.10 Dataset Transformations....35
8 Exec & Plugin....37
8.1 Exec....37
8.2 Plugin....39

Version Tracking:

Version	Date	By	Comment
1.0.0	23 June 2009	JXC	Completed content.
1.0.1	25 June 2009	JXC	Proof-reading changes. Also root → response, rows → row.
1.0.2	25 June 2009	JXC	Added “method_param” documentation.
1.0.3	25 June 2009	JXC	Added “Single” login authentication module.
1.0.4	29 June 2009	JXC	Removed all references to “use_placeholders”.
1.0.5	29 June 2009	JXC	Removed references to “rollback_on_error” and “stop_on_error”. Jarvis will now always rollback and stop on error. In return structure removed “state”. Removed “modified” and “rows” from modification error return structure.
1.0.6	7 July 2009	JXC	Added “log_format”, plus “before” and “after”.
1.0.7	8 July 2009	JXC	Allow comma-separated IP list on Login::Single and BasicAuth.
1.0.8	21 July 2009	JXC	Added & documented support for “MIXED” requests.
1.0.9	2 Sep 2009	JL	Added new configuration 'default_libs' to define global library paths for plugins.
1.1.0	8 Sep 2009	JXC	Added support for “.” sub-directory names within dataset names.
1.1.1	10 Sep 2009	JXC	Added store/fetch transformations.
1.1.2	16 Sep 2009	JXC	Added separate “dump” option.
1.1.3	13 Oct 2009	JXC	Added per-dataset debug/dump flags.
1.1.4	11 Dec 2009	JXC	Updated notes on JARVIS_ROOT configuration.

1 Introduction

Jarvis is “helper glue”. It is designed to bridge the gap between your web-apps and your back-end database. The three standard components in a solution using Jarvis are:

1. Rich Internet Application. Ajax (XML or JSON) requests and responses.
2. JARVIS
3. Database. Accessible via SQL.

Front-end RIAs are often written using technologies such as Adobe Flex, or JavaScript using libraries including Dojo or ExtJS. These are often simple CRUD (CReate, Update, Delete) applications which simply wish to perform basic operations on a back end database.

This requires some server script to handle data requests over http and perform the corresponding back-end database transactions in a manner which is secure, extensible, standards-based and reasonably efficient.

Jarvis is that server script, and meets those three key requirements.

- **Secure:**
  - Jarvis can run over https.
  - Jarvis uses single-login and CGI cookies to maintain authenticated sessions.
  - Jarvis uses parameterized statements to prevent against SQL injection.
  - Jarvis provides group-based access control.
- **Extensible:**
  - Jarvis provides independent “datasets”, defined as simple XML files containing SQL.
  - Jarvis provides configurable “exec” extensions (e.g. running Jasper reports).
  - Jarvis provides a “plugin” mechanism for adding custom Perl modules.
- **Standards-Based:**
  - Jarvis is based on XML, JSON, http/s and REST.
- **Reasonably Efficient:**
  - Jarvis is “plenty fast enough”.
  - Jarvis is written in a scripted language as many back end services are.
  - Jarvis doesn't cache database connections. This may be a future enhancement.

However, in practical use, the load added by Jarvis appears to be very minimal – and Jarvis performance isn't likely to become a major concern unless you are running a very intensive application.

For the majority of “everyday” web-apps, I hope you'll find that Jarvis does everything you need to do, in a simple and helpful fashion.

2 Licence

2.1 GNU Lesser General Public License

This documentation is part of the Jarvis WebApp/Database gateway utility.

2.2 Clarification of Intentions

The intention of the releasing under the LGPL (rather than the GPL) is to provide additional freedom to allow you to develop Exec and Plugin modules.

Such Exec and Plugin modules are considered to be part of your “Application” and not part of Jarvis, and are thus excluded from the “Minimal Corresponding Source” under the terms of the agreement.

3 Installation

3.1 Unpacking

Unpack the tar.gz file into a directory. For the purposes of this documentation, I assume that you are running under Linux/Unix and have unpacked into /opt/jarvis. The following directory tree should be created.

Path	Notes
/opt/jarvis/cgi-bin	This contains jarvis.pl, which is the CGI script which accepts requests over http/https. The jarvis.pl file is relatively simple. Most of the functionality is in the lib/Jarvis/*.pm files.
/opt/jarvis/demo	Contains demo files which comprise a simple application showing how to use Jarvis with ExtJS and PostgreSql.
/opt/jarvis/docs	Documentation including this file.
/opt/jarvis/etc	Your application top-level configuration XML file will go here. Under Linux/Unix you will typically soft link to your application directory.
/opt/jarvis/htdocs	This contains a helper JavaScript include file for use with ExtJS.
/opt/jarvis/lib	This contains the majority of the Jarvis functionality as a set of Perl modules.

3.2 Install Additional Perl Modules

Jarvis is written in Perl, and relies on some additional Perl modules being installed. You can install these via CPAN, or under Ubunto the following apt-get commands.

REQUIRED

```
sudo apt-get install libcgi-session-perl
sudo apt-get install libxml-smart-perl
sudo apt-get install libjson-perl
sudu apt-get install libtext-csv-perl
sudu apt-get install libio-string-perl
sudu apt-get install libmime-types-perl
```

OPTIONAL

```
sudo apt-get install libdbd-sybase-perl     (DB = SQL Server)
sudo apt-get install libdbd-pg-perl         (DB = PostgreSQL)
sudo apt-get install libnet-ldap-perl       (Login = LDAP or ActiveDirectory)
```

3.3 Web Server Configuration

The Jarvis library needs to be installed into the webserver. Under apache2 copy the following lines into your apache configuration.

Note that the "/jarvis/script/" definition is required only if you are using the ExtJS helper script (required by the demo).

Note also that the "/jarvis/script/" definition is a more specific match and hence must precede the more general "/jarvis/" alias in the configuration.

```
ScriptAlias /jarvis/jarvis.pl "/opt/jarvis/cgi-bin/jarvis.pl"
Alias /jarvis/ "/opt/jarvis/htdocs/"
```

These lines are contained in /opt/jarvis/docs/jarvis-apache. With modern Linux distributions, the simplest approach is to copy this file into Apache's "conf.d" directory.

3.4 Relocating Jarvis

This example shows Jarvis installed into /opt/jarvis. If you install jarvis into another location, then change the Alias and ScriptAlias lines accordingly. In addition, you must either:

- Change the default value JARVIS_ROOT value hard-coded in jarvis.pl, **or**:
- Set the environment variable JARVIS_ROOT before running the script.

If your webserver is Apache, then you can simply set environment variables with a directive like:

```
SetEnv JARVIS_ROOT "C:/Jarvis"
```

Note also that the demo.xml application configuration file contains references to /opt/jarvis/demo. This is of course only relevant if you wish to run the demo.

3.5 Test

To test that Jarvis is correctly installed, point your browser at the installed machine (e.g. localhost) with the following URL.

http://localhost/jarvis/jarvis.pl

You should see the error response.

```
Missing path info.  Send http://.../jarvis.pl/<app-name>/<dataset-name> in URI!
```

This indicates that Jarvis is installed, and is ready to be configured for an application.

4 Application Configuration File

4.1 Introduction

From Jarvis's point of view, an "application" is a collection of datasets which share some basic attributes in common. Most importantly they share:

- The database connection.
- The username/group list.
- The directory containing dataset definition files.

These parameters and many others are defined by the application global configuration file. The following file demonstrates all of the current application config parameters supported by Jarvis.

```
<?xml version="1.0" encoding="utf-8"?>
<jarvis>
    <app format="json" debug="yes" dump="yes">
        <habitat>
            <install_type>test</install_type>
        </habitat>

        <page_limit_param>limit</page_limit_param>
        <page_start_param>start</page_start_param>
        <sort_field_param>sort</sort_field_param>
        <sort_dir_param>dir</sort_dir_param>
        <method_param>_method</method_param>

        <dataset_dir>/home/spiderfan/edit/etc/dataset</dataset_dir>

        <default_libs>
            <lib path="/home/spiderfan/edit/plugin"/>
        </default_libs>

        <default_parameters>
            <parameter name="max_rows" value="500"/>
        </default_parameters>

        <login module="Jarvis::Login::Database">
            <parameter name="user_table" value="staff"/>
            <parameter name="user_username_column" value="name"/>
            <parameter name="user_password_column" value="password"/>
            <parameter name="group_table" value="staff_group"/>
            <parameter name="group_username_column" value="name"/>
            <parameter name="group_group_column" value="group_name"/>
        </login>
        <database connect="dbi:Pg:dbname=test"
                username="" password=""/>

        <sessiondb store="driver:file;serializer:default;id:md5"
                expiry="+3M" cookie="APP_CGISESSID">
            <parameter name="Directory" value="/home/spiderfan/tmp/sessions"/>
        </sessiondb>

        <exec dataset="jasper" access="*"
            command="java -jar /opt/jasper/jasper.jar"
            add_headers="yes" filename_parameter="filename"/>
```

```
        <plugin dataset="csvexport" access="admin"
            lib="/var/www/jarvis/plugin"
            module="Test::ExportToCsv" add_headers="yes"/>

        <plugin dataset="doSomethingUnspecified" access="admin"
            module="Test::DoSomethingUnspecified" add_headers="yes"/>
    </app>
</jarvis>
```

Each of these will be discussed in detail.

4.2 <jarvis><app>

All configuration is currently contained within <jarvis><app>. Only one application is defined per configuration file. The name of the application is defined by the name of the application configuration file. For example, an application named "test" must be defined by a configuration file named:

```
/opt/jarvis/etc/test.xml
```

The <app> tag may have the following attributes. Note that throughout this configuration file all "boolean" attributes interpret "yes", "true", "on" or "1" to indicate true. All other values mean false.

Attribute	Default	Notes
format	json	"json" or "xml", specifies the format for returned content. Note that content in the request may in some cases allow either XML or JSON. In these cases, Jarvis will examine the first non-blank character and if '<' will treat it as XML, otherwise JSON.
debug	no	This will enable additional debug output to STDERR, which will appear in your apache error log.
dump	no	Identical to debug, the dump flag enables "extra" debug including a full dump of all requests and returned content.
log_format	See Notes	Specifies the format template to use for debug, dump and log output. Default is "[%P/%A/%U/%D] %M". See following log_format section for description of available fields.
page_limit_param	page_start	Name of the CGI parameter which will be used as the page size for server-side paging (performed after fetching).
page_start_param	page_limit	Name of the CGI parameter which will be used as the start row number for server-side paging (performed after fetching).
sort_field_param	sort_field	Name of the CGI parameter which will be used column name for server-side sorting (performed after fetching). CGI parameter with this name should match a returned column name.
sort_dir_param	sort_dir	Name of the CGI parameter which will be used column name for server-side sorting (performed after fetching). CGI parameter with this name should have value 'ASC' or 'DESC'.

Attribute	Default	Notes
method_param	_method	Name of the CGI parameter which may over-ride the request method for RESTful updates. Required for Flex which restricts requests to GET and POST only.
dataset_dir	<none>	This parameter is mandatory. It must specify the directory containing the application's datasets.
default_libs	<none>	This element allows a list of zero or more directories to be defined which will then be added to the @INC path when loading plugin modules to execute.
default_parameters	<none>	Defines values which are supplied as default user parameters on all requests.
login	<n/a>	This parameter is mandatory. It is a sub-element which defines and configures the login mechanism to be used for this application. Only one single instance of this should be defined.
database	<n/a>	This parameter is mandatory. It is a sub-element which defines the database connection parameters. Only one single instance of this should be defined.
sessiondb	<n/a>	This is a sub-element which defines the configuration for the cookie store. It may be omitted, since defaults should work in most cases.
exec	<n/a>	One or more optional sub-elements of this name may define an exec action (spawn a sub-process running a separate program).
plugin	<n/a>	One or more optional sub-elements of this name may define a plugin action (dynamically load a Perl module and execute the "do" method on that module).
habitat	<n/a>	Optional single sub-element containing configuration which will be passed to the application on request.

Table 1: <jarvis><app> Attributes

4.3 Configuration: format

This determines the default returned format. Specify "json", "xml" or "csv".

- This may be over-ridden a per-request basis with the CGI parameter "format".
- This does not affect the content format for submitting changes. That is specified using the content-type header.

4.4 Configuration: debug

This will cause extra debug to be written to standard error, which will appear in the apache error log. In a high-traffic environment, you should carefully consider the potential performance impact of enabling debug.

4.5 Configuration: dump

The dump flag enables a second level of debug which also prints output showing:

- All client-submitted request body.
- All returned XML/JSON/CSV content.
- All SQL statements being executed.

Enabling “dump” will also enable “debug”. Again, “dump” output should not generally be used in a high-traffic environment.

4.6 Configuration: log_format

The Jarvis script will generate log output in certain situations. If enabled, it will also generate a significant amount of detailed debug and/or dump output which is potentially useful for identifying problems where client requests are not being processed as expected.

The log format string specifies the format which is used for log, debug and dump output. This is a string which may contain the following placeholders.

Attribute	Notes
%T	Timestamp of log event, e.g. “Mon Jul 06 09:41:33 2009”. This is not generally required since the Apache error log output already contains a timestamp.
%L	Level of output, either “log” or “debug”.
%U	Current logged-in username.
%D	Requested dataset name.
%A	Requested application name.
%P	Process ID of slave process.
%M	Message text to be logged.

Table 2: Log Format Placeholders

4.7 Configuration: page_limit_param, page_start_param

This defines the name of the CGI parameters which should be interpreted as providing server-side page requests. Different RIA frameworks use different defaults. For example, the ExtJS PagingBar widget uses “limit” and “start”.

Whatever parameters are configured for these values (including defaults) you should make sure that your queries do not attempt to use CGI parameters with the same name in their fetch queries, otherwise you will find data being unexpectedly page-sliced.

If CGI parameters of the specified name are found in the CGI fetch request, then Jarvis will do the following:

- Extract ALL records from the database query.
- Slice out and return only those rows specified by the page parameters. E.g start = 50, limit = 25 means return rows with zero-based indexes 50-74.
- Return “fetched” as the total number of all rows read from the database query.
- Return only the actual data rows from the sliced page.

Fetching all rows on the server does incur overhead between Jarvis and the database. However, it has the advantage that we can tell the client exactly how many rows it is paging among.

4.8 Configuration: sort_field_param, sort_dir_param

This defines the name of the CGI parameters which should be interpreted as providing server-side page requests. Different RIA frameworks use different defaults. For example, the ExtJS PagingBar widget uses “sort” and “dir”.

Whatever parameters are configured for these values (including defaults) you should make sure that your queries do not attempt to use CGI parameters with the same name in their fetch queries, otherwise you will find data being unexpectedly server-side sorted.

When the CGI parameter named by sort_field_param is present, Jarvis will perform the following.

- Extract ALL records from the database query in the default query sort order.
- Use Perl “cmp” alphabetical sorting either ascending or descending.
- Return the rows in the Perl-sorted order.

Note that the column named in the sort field parameter is case-sensitive and must match the exact column name returned by the <select> query in the dataset definition.

Note that the Perl “cmp” sorting may very well be different from the databases ORDER BY sequencing.

Note that if the sort_field_param is present but the sort_dir_param is not, then the sort order will be ascending.

Note that the sort_dir_param is based on the first letter of the value. A direction starting with “a” or “A” means ascending. A direction starting with “d” or “D” means descending.

4.9 Configuration: dataset_dir

This is the location of the XML datafiles which define the application's datasets.

This must be defined, but if your application really has no datasets (perhaps is it consists entirely of exec and plugin actions) then you may define dataset_dir to be /dev/null or some non-existent location.

- All dataset definitions must be contained within XML files contained in or below the single directory path specified by the “dataset_dir” parameter.
- All dataset files must have the suffix “.xml”.
- The “.xml” suffix must not be included in the dataset name as provided by the client, it will always be added by Jarvis.
- The dataset name specified by the client must consist only of characters from the following list: “a-z”, “A-Z”, “0-9”, “_” (underscore), “-” (hyphen) and “.” (dot).
- A “.” (dot) character in the dataset name is interpreted as a directory separator by Jarvis. On Unix-based systems this is a forward-slash. On Windows-based systems, Perl will translate the forward slash into a backslash.
- A dataset name specified by the client may not start or end with a dot.

Consider the following examples for <dataset_dir>/opt/app/datasets</dataset_dir>. A unix-based slash has been used for these examples.

Client Requests	Resulting XML File
my-set	/opt/app/datasets/my-set.xml
.my-set	[Error, may not start with dot]
my-set.	[Error, may not end with dot]
folder.myset	/opt/app/datasets/folder/myset.xml
myset.xml	/opt/app/datasets/myset/xml.xml

Table 3: Examples of Resolving Dataset Names

4.10 Configuration: default_libs

A list of library paths to @INC into the process before attempting to load and execute a plugin module. For example:

```
<default_libs>
    <lib path="/home/spiderfan/edit/plugin"/>
    <lib path="/var/www/jarvis/plugin"/>
</default_libs>
```

When a plugin is to be executed, any default library paths defined will be added to the Perl include path, and then if the plugin includes the 'lib' attribute, the plugin's specific library path will be added. This allows the plugin to override the default paths if necessary.

4.11 Configuration: method_param

There is currently a limitation with Adobe Flex which appears to restrict non-proxied requests to using only the GET and POST methods. This naturally causes a problem when using RESTful web-services.

To work around this problem, Jarvis uses an approach similar to other web-services which allows a GET or POST parameter to override the supplied http request method when determining the nature of the action (insert, update, delete) to be performed.

By default, the "_method" CGI parameter is used for this purpose.

4.12 Configuration: login

Every application must have a login module defined. Currently Jarvis offers four standard login modules:

- None – user is always logged in.
- Database – the username and passwords are stored in the database.
- ActiveDirectory – the usernames and passwords are checked against an Microsoft ActiveDirectory server. It is likely that one day this will be copied and modified to add support for standard LDAP, which is very similar.
- BasicAuth – Apache basic authentication is configured, and we will trust it. This may include FakeBasicAuth which allows Apache to verify client SSL certificates on our behalf.

Login configuration is the subject of a separate chapter.

4.13 Configuration: database

This is a sub-element which must contain the following attributes:

Attribute	Default	Notes
connect	dbi:Pg:<app-name>	This is a DBI connection string.
username	<empty-string>	This is the username Jarvis uses to connect to the database for all requests.
password	<empty-string>	This is the password Jarvis uses to connect to the database for all requests.

Table 4: Database Configuration Attributes

See the DBI man pages for information on the connection string. But common examples are:

- dbi:Pg:<database-handle> (Postgres)
- dbi:Sybase:server=<ip-address-or-name> (SQL Server)

4.14 Configuration: sessiondb

This defines the session database. Sessions are created and managed with the Perl CGI::Session module. Configuration as follows.

Attribute	Default	Notes
store	driver:file;serializer:default;id:md5	This tells CGI::Session to use file-based cookies. See CGI::Session man page for further information.
cookie	CGISESSID	CGI::Session will use its default value of CGISESSID if this is not specified. If you are running multiple RIA apps from Jarvis, then configure each one to use a separate cookie name.
expiry	+1h	Specifies the extension period for the cookie after every successful Jarvis interaction.

Table 5: Session DB Configuration Parameters

In addition, the sessiondb configuration supports additional contained “parameter” tags, according to the CGI::Session driver type selected.

```
        <sessiondb store="driver:file;serializer:default;id:md5"
                expiry="+3M" cookie="APP_CGISESSID">
            <parameter name="Directory" value="/home/spiderfan/tmp/sessions"/>
        </sessiondb>
```

Each parameter sub-element has a “name” and “value” attribute. These parameters are passed directly to CGI::Session. The only documented parameter is “Directory” for the “file” driver. For other parameters, see the CGI::Session man page.

Name	Default	Notes
Directory	(System TMP Dir)	This specifies the location for storing CGI sessions on local disk. The default directory is operating system dependent.

Table 6: Session DB File Driver Parameters

4.15 Configuration: exec & plugin

An “exec” configuration defines a special dataset which is fetched by spawning a custom sub-process. A “plugin” configuration also defines a special dataset which is fetched by loading a custom Perl module written as a “.pm” file.

The dataset names configured for “exec” and “plugin” commands should each have a unique dataset name which does not conflict with any other “plugin” or “exec” dataset, does not conflict with any regular dataset, and which does not conflict with any of the predefined built-in special dataset names (e.g. '__status', '__habitat' and '__logout').

See the separate chapter on “exec” and “plugin” requests for further details.

4.16 Configuration: habitat

This is a static piece of content to be returned to any caller who invokes the '__habitat' special dataset. Note that no login is required in order to access the habitat string. It is entirely insecure.

Standard use of a habitat is to allow an application to run in separate environments (e.g. production, testing, etc.) and to have different configuration in those environments. E.g. the application may examine the habitat and decide to change its visual appearance to show users that it is a test environment.

See the separate section on “Habitat” requests for further details.

5 Login Module Configuration

5.1 Standard Modules

Jarvis uses a pluggable login module mechanism. The following modules are provided:

- ActiveDirectory.pm
- BasicAuth.pm
- Database.pm
- None.pm
- Single.pm

You may copy and modify these to create custom login modules in /opt/jarvis/lib/Jarvis/Login.

The XML configuration for all these modules uses an identical syntax with <parameter name='...' value='...'/> elements. The difference is in the names of the parameters for each module.

5.2 Active Directory Login

This Login module will query a Microsoft ActiveDirectory server. Example configuration is:

```
<jarvis>
    <app>
        <login module="Jarvis::Login::ActiveDirectory">
            <parameter name="server" value="company-pdc"/>
            <parameter name="bind_username" value="bind user"/>
            <parameter name="bind_password" value="bindpass"/>
            <parameter name="base_object"
                value="OU=OFFICE,DC=COMPANY,DC=LOCAL"/>
        </login>
```

The parameters are.

Attribute	Default	Notes
server	<none>	IP address or resolvable DNS name locating the primary domain controller. Currently this module does not support secondary domain controllers. Mandatory.
port	389	IP port number.
bind_username	"	The username to be specified at the bind attempt.
bind_password	"	The password to be specified at the bind attempt.
base_object	<none>	The base object to be specified in the search request. Mandatory.

Table 7: Active Directory Login Module Parameters

The module will bind to the AD server with the bind username and password. It will request a search of the full tree below the base object, with full dereferencing. The filter is for "samaccountname" equal to the username offered to Jarvis for this login attempt. We ask the search to tell us of all "memberOf" attributes for this group.

If the user exists, Jarvis then assembles the grouplist from the memberOf parameters returned. Then Jarvis unbinds, and attempts to rebind with the user-supplied username and password, instead of the plugin-defined values. If the rebind succeeds, then the user is validated.

5.3 Basic Auth Login

This login module expects Apache's BasicAuth mechanism to perform the password checking.

Example configuration (not using client certificates) is:

```
<jarvis>
    <app>
        <login module="Jarvis::Login::BasicAuth">
          <parameter name="group_list" value="staff"/>
        </login>
```

Example configuration (using client certificates) is:

```
<jarvis>
    <app>
          <parameter name="require_https" value="yes"/>
          <parameter name="remote_ip" value="10.42.2.100"/>
          <parameter name="remote_user"
      value="/C=NZ/ST=State/O=Company/CN=User Name Here"/>
          <parameter name="username" value="admin"/>
          <parameter name="group_list" value="admin,staff"/>
        </login>
```

The parameters are.

Attribute	Default	Notes
require_https	no	If “yes” then we require the connection to be HTTPS not HTTP.
remote_ip	<none>	If specified, this is a comma-separated list if IP addresses, one of which must be an exact match on the user's IP address.
remote_user	<none>	This specifies the username exactly as it is specified in the Apache HTTP password file granting BasicAuth access to the user. If this is not specified then any BasicAuth user will be allowed access.
username	<none>	Jarvis username which the user will be granted. If not specified, the name passed from Apache BasicAuth will be used.
group_list	<none>	This is the comma-separated Jarvis group list to grant to this user. If not specified, then a single group with name identical to the assigned Jarvis username will be used.

Table 8: BasicAuth Login Module Parameters

In the first example, we are using standard Basic Auth. Any client IP address is permitted. HTTPS is not required. Any configured basic auth username is allowed, and their Jarvis username will be the same as the username provided to Apache. All users will belong to a single group named 'staff'.

In the second example we are using Apache FakeBasicAuth and Client certificates. HTTPS is required. Only access from the specified source IP address is permitted. Only the specified certificate Distinguished Name is permitted, exactly as given. Note that Apache's FakeBasicAuth uses slashes instead of commas when deriving the DN. This matches what is configured for the client certificate in the Apache password file. The Jarvis username will appear as “admin” and the user will belong to two groups: “admin” and “staff”.

5.4 Database Login

This module performs username and password lookup in the configured database. Example:

```
<jarvis>
    <app>
        <login module="Jarvis::Login::Database">
            <parameter name="user_table" value="staff"/>
            <parameter name="user_username_column" value="name"/>
            <parameter name="user_password_column" value="password"/>
            <parameter name="group_table" value="staff_group"/>
            <parameter name="group_username_column" value="name"/>
            <parameter name="group_group_column" value="group_name"/>
        </login>
```

The parameters are.

Attribute	Default	Notes
user_table	<none>	Name of the database table containing username and password columns. Mandatory.
user_username	<none>	Name of the username column. Mandatory.
user_password	<none>	Name of the password column. Mandatory.
group_table	<none>	Optional group table containing username/group pairs.
group_username_column	<none>	Name of the username column in the group table.
group_group_column	<none>	Name of the groupname column in the group table.

Table 9: Database Login Module Parameters

The three user parameters are mandatory. In order to perform group lookup, all three "group" parameters must be configured. If not, all users will be placed in a single group named "default".

5.5 None Login

This module automatically performs login and allocates hard-coded username and group values. It is useful for quick testing of applications. Example configuration is:

```
<jarvis>
    <app>
        <login module="Jarvis::Login::None">
            <parameter name="username" value="admin"/>
            <parameter name="group_list" value="admin"/>
        </login>
```

The parameters are.

Attribute	Default	Notes
username	<none>	User name to assign to all logins.
group_list	<none>	Comma-separated group list to assign to all logins.

Table 10: None Login Module Parameters

5.6 Single Login

This module is slightly more secure than the None module. It is very limited in that it allows for a single username only, but it does provide at least some security checking:

- Remote (client) IP address match, and/or
- Client must supply username and password, and/or
- HTTPS protocol is required.

An example configuration is:

```
<jarvis>
    <app>
        <login module="Jarvis::Login::Single">
            <parameter name="require_https" value="no"/>
            <parameter name="remote_ip" value="127.0.0.1"/>
            <parameter name="username" value="bob"/>
            <parameter name="password" value="test"/>
            <parameter name="group_list" value="default"/>
        </login>
```

The parameters are.

Attribute	Default	Notes
username	<none>	User name to assign to anybody who successfully logs in using this module.
group_list	<none>	Comma-separated group list to assign to all logins using this module. If not specified, a single group identical to the username will be assigned.
password	<none>	If configured then the client must supply both “username” and “password” as CGI parameters when logging in to obtain a session cookie, and both supplied values must match the configured values. If not-configured (or configured empty) then any “username” and “password” supplied by the client will be ignored. The configured “username” parameter will be used as the username.
remote_ip	<none>	If configured then the client's remote IP address must exactly match one of the addresses. This may be a single address or a comma-separated list.
require_https	no	If configured “yes”, then the client connection must be HTTPS.

Table 11: Single Login Module Parameters

Note that at least one of “remote_ip” or “password” must be configured.

6 Special Datasets & Jarvis Login

6.1 Jarvis URLs

Now let's consider how to access Jarvis. The following examples are based on the demo configuration shipped with Jarvis. It assumes that:

- Jarvis is installed into /opt/jarvis as per default.
- The jarvis ScriptAlias is configured.
- The Apache webserver is accessible on http://localhost/.
- The demo.xml file has been soft linked or copied into /.
- The "None" login module is configured for the demo application.
- The default response format is configured as "json" for the demo application.

At this stage, we will consider only the "special datasets", which do not require the demo database to be actually installed.

6.2 The __status Dataset

The __status dataset (with two leading underscores) is a special dataset which allows your application to determine the user's login status without actually performing a data request.

To access the __status dataset in your web browser, send a GET request to the following URL.

```
http://localhost/jarvis/jarvis.pl/demo/__status
```

This is a specific example of the general Jarvis URL format which is.

```
http://localhost/jarvis/jarvis.pl/<app-name>/<dataset>[/<p1>[/<p2>...]]
```

This is a RESTful URL which specifies an application name "demo", then a dataset within that application. If Jarvis and the demo app are correctly configured, then the response should be JavaScript Object Notation (JSON) response with a Content-Type of "plain/text".

```
{
   "error_string" : "",
   "logged_in" : "1",
   "group_list" : "admin",
   "username" : "admin"
}
```

This is a response object with four attributes.

When logged_in = 0, username and group_list will always be empty and the error_string will be a non-empty description of why the login failed.

Conversely if logged_in = 1 then the error_string will always be empty, username will always be non-empty, and group_list contains a possibly zero-length comma-separated list of groups.

Note that it is not always necessary to be logged in to access a dataset. The special datasets are available to non-logged-in users. Also the dataset creator may offer user-defined datasets to non-logged-in users by specifying "**" as the access string in the dataset XML definition.

To receive responses in XML format, either configure the default format as "xml" in the demo.xml

file, or pass “format” as a CGI parameter. E.g.

```
http://localhost/jarvis/jarvis.pl/demo/__status?format=xml
```

The response is also Content-Type “plain/text” but with XML content. The same object is returned as in the JSON case, but in XML.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<?meta name="GENERATOR" content="XML::Smart/1.6.9 Perl/5.010000 [linux]" ?>
<response error_string="" group_list="admin" logged_in="1" username="admin"/>
```

6.3 The __habitat Dataset

The __habitat dataset provides a simple method for an application configuration file to pass some of its configuration directly back to the client application instance. BE WARNED. Your HABITAT is PUBLIC. Login is not required in order to view the habitat.

The habitat is configured in the applications configuration file. The habitat is returned essentially verbatim, however there a difference between JSON and XML habitat returned handling.

6.3.1 JSON Habitat

This is how you might define a JSON habitat. If your requested format is non-XML (e.g. JSON), we will strip the outer <habitat> tags for you.

```
<jarvis>
    <app>
        <habitat><![CDATA[hargs: {
    install_type: 'production'
}]]></habitat>
```

The return from the query:

```
http://localhost/jarvis/jarvis.pl/demo/__habitat?format=json
```

will be:

```
hargs: {
    install_type: 'production'
}
```

6.3.2 XML Habitat

In the XML case, the habitat is returned as an XML object, parsed and then re-encoded. This means that the habitat returned will be logically equivalent, but may not necessarily be byte-for-byte equivalent to what is defined in the application configuration file.

```
<jarvis>
    <app>
        <habitat>
            <install_type>production</install_type>
            <parameter name="pname" value="some_value"/>
            <parameter name="another" value="a_different_value"/>
        </habitat>
```

With the above habitat, requesting in XML:

```
http://localhost/jarvis/jarvis.pl/demo/__habitat?format=xml
```

The result returned to the client will be:

```
  <install_type>production</install_type>
  <parameter name="pname" value="some_value"/>
  <parameter name="another" value="a_different_value"/>
```

6.4 The __logout Dataset

Invoking the "__logout" dataset will cause the server-side session ID to be erased from the session store.

Note that when using the "None" Login module, erasing the cookie is all well and good, but the very next request will always automatically re-login and create a new cookie and session ID.

6.5 Logging-In to Jarvis

Using the "__status" and "__logout" datasets, you can login and logout of Jarvis.

To login to Jarvis, make any request to Jarvis, and include the following parameters:

- "username"
- "password"

Jarvis will perform the configured login process, and will always create a server-side session. The session ID will be returned in the response headers as a cookie. The default cookie name is CGISESSID. This can be configured to avoid conflict if multiple applications are using session cookies on the same server.

Even if the login fails, a session cookie will still be returned. For maximum efficiency and security, your application should supply "username" and "password" only on the very first request, and should pass the session cookie subsequently. The session cookie lifetime will be extended on every successful request. The exception to this is "exec" and "plugin" requests which create their own response headers.

Any request may be used to perform session login. However, the convention is to only perform login as part of a "__status" request.

Note that the "__status" response parameters (error_string, logged_in, username, group_list) are also included in data fetch results returned by Jarvis. However, they are not included in the responses to dataset modification requests.

6.6 Jarvis Error Responses

Exception handing is an important part of any application, especially so when dealing with client/server web-services.

Compilation Failure

When the Jarvis scripts fail to compile, you will receive a "500 Internal Server Error" generated by Apache.

Authorization Required

When access is requested to a resource which requires login, or which requires membership of a group not in the current user's group list, then a "401 Unauthorized" is returned with a "text/plain" content body which describes the failure reason. It may also include the script line number.

The calling application should invoke the "__status" dataset to ensure that the user is logged-in, and log them in if required.

No Such Dataset

When access is requested to a dataset which does not match a plugin, exec, special dataset, or regular dataset, then a "404 Not Found" is returned with a "text/plain" content body which names the not-found dataset. It may also include the script line number.

Request/Configuration Error

Otherwise, if Jarvis compiles but identifies a fatal problem not related to dataset access permissions or unknown dataset, it will return a Jarvis-generated "500 Internal Server Error" request.

- Internal configuration problem – bad server-side XML configuration (main or dataset).
- Database connection problem.
- Request body is malformed JSON or XML.
- Missing mandatory parameter in request.
- Any other fatal problem.
- Configured SQL statement in dataset definition is not valid.

Soft Errors

Soft errors are errors related to the actual user-supplied data values being rejected by the database. Examples:

- Cannot insert data due to primary key, foreign key, unique constraints.
- No data found.
- Data parameter is invalid type/value for column.

In all these cases, the returned content will be JSON or XML or CSV in a "200 Successful" response.

7 Datasets

7.1 Data Fetch Response Format

Now let's consider a real dataset examples, defined as a set of SQL statements in the datasets configuration XML file. For the purposes of this discussion, I assume that the demo database has been created by using the postgres.sql file supplied in /opt/jarvis/demo/sql. Let's proceed directly to an example.

```
http://localhost/jarvis/jarvis.pl/demo/boat_class
```

The JSON format response for a data fetch is as follows:

```
{
   "data" : [
      {
         "active" : "Y",
         "class" : "Makkleson",
         "id" : "6",
         "description" : "Suitable for infants and those of timid heart."
      },
      {
         "active" : "N",
         "class" : "X Class",
         "id" : "4",
         "description" : "Product of a deranged mind."
      }
   ],
   "fetched" : 2,
   "error_string" : "",
   "logged_in" : "1",
   "group_list" : "admin",
   "username" : "admin"
}
```

For XML, change the default format, or specify the format per-request.

```
http://localhost/jarvis/jarvis.pl/demo/boat_class?format=xml
```

The returned content in XML is:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<?meta name="GENERATOR" content="XML::Smart/1.6.9 Perl/5.010000 [linux]" ?>
<response logged_in="1" username="admin" error_string=""
    group_list="admin" fetched="2">
  <data>
    <row active="Y" class="Makkleson"
        description="Suitable for infants and those of timid heart." id="6"/>
    <row active="N" class="X Class"
        description="Product of a deranged mind." id="4"/>
  </data>
</response>
```

In JSON or XML, the top level attributes are the same for all data fetch responses:

Attribute	Notes
data	An array of objects returned from the SELECT statement.
fetched	The number of rows fetched from the SELECT, before any server-side paging.
error_string	Refer to the "__status" dataset notes.
logged_in	Refer to the "__status" dataset notes.
group_list	Refer to the "__status" dataset notes.
username	Refer to the "__status" dataset notes.

Table 12: Dataset Fetch Top-Level Attribute

7.2 Dataset Definition

The dataset is defined in a .XML file containing up to four SQL statements. The "boat_class.xml" file is as follows.

```
<dataset read="*" write="*" debug="no" dump="no">
    <select>
SELECT id, class, active, COALESCE (description, '') AS description
FROM boat_class
ORDER BY class
    </select>
    <update>
UPDATE boat_class
SET class = BTRIM ({{class}}), active = {{active}},
    description = NULLIF (BTRIM ({{description}}), ''),
    change_user = {{__username}}, change_date = now()
WHERE id = {{id}};
    </update>
    <insert returning="yes">
INSERT INTO boat_class (class, active, description, change_user, change_date)
VALUES (BTRIM ({{class}}), {{active}}, NULLIF (BTRIM ({{description}}), ''),
{{__username}}, now())
RETURNING {{_record_id}}::integer as _record_id, id;
    </insert>
    <delete>DELETE FROM boat_class WHERE id = {{id}};</delete>
</dataset>
```

Note the use of COALESCE, NULLIF and BTRIM to ensure that a NULL in the database is presented as " to the client. Similarly a " provided from the client is stored as NULL in the database. The <transform> tag described later provides a way to simplify this process.

The top-level attributes of all datasets definitions are as follows.

Attribute	Notes
read	Defines which groups may execute the SQL defined in the “select” element. This is an access identifer as per the following table.
write	Defines which groups may execute the SQL defined in the “update”, “insert” and “delete” elements. This is an access identifer as per the following table.
debug	Enables debug on a per-dataset basis. Debug starts from the point that the dataset is successfully loaded. Debug uses the globally configured debug format and is identical in all regards to the global debug output.
dump	Enables dump on a per-dataset basis.

Table 13: Attributes of the <dataset> Element

The elements within the <dataset> definition are as follows:

Element	Notes
transform	Optional transformations for Jarvis to perform when storing or fetching data.
select	SQL to execute when dataset is invoked with the http GET request method.
before	SQL to execute once before performing update/insert/delete changes.
update	SQL to execute when dataset is invoked with the http PUT request method.
insert	SQL to execute when dataset is invoked with the http POST request method.
delete	SQL to execute when dataset is invoked with the http DELETE request method.
after	SQL to execute once after performing update/insert/delete changes.

Table 14: Dataset Sub-Elements

The “insert” statement supports one sub-parameter, the “returning” parameter. This is discussed later under the “insert” statement section.

The access identifiers are.

Access ID	Notes
<empty>	Nobody may access these statements.
<g1>[,<g2>...]	A comma separated group list. Membership in any one of these groups will grant access.
“*”	Any logged-in group may access this feature.
“**”	Any user, even if not logged-in, may access this feature.

Table 15: Access Control Specifiers

This is the same list of options as is used for the access control of “exec” and “plugin” datasets.

7.3 Dataset Parameters

The SQL statements in the datasets definition contain substitution parameters defined in curly braces. Either double or single curly braces may be used. The parameter may optionally have a preceding '$' dollar symbol. I.e. '{name}', '{{name}}', '{$name}' and '{{$name}}' are all equivalent.

The values for these parameters may come from the following sources:

User Parameters:

- CGI GET parameters in the URL suffix "/<app>/<ds>?param=value&param2=another".
- XML or JSON parameters in the "application/json" or "application/xml" request body.
- RESTful parameters in the URL suffix "/<app>/<ds>/p1/p2".
- Jarvis-Supplied default user parameters (from "default_parameters" configuration).

Secure Parameters:

- Jarvis-Supplied secure parameters.

Also, any parameter may be defined as a pipe-separated sequence of parameters. The listed parameters will be checked in series until one with a defined value is encountered. An empty string value counts as defined for this purpose. If no client-supplied value is found then the server-side "default_parameters" configuration is checked. Finally, a NULL value will be used.

Consider the following query.

```
<dataset read="*" write="*">
    <select>
SELECT id, class, active FROM boat_class
WHERE ({{1|class_name}}::text IS NULL) OR class ~ {{1|class_name}}
ORDER BY class
    </select>
</dataset>
```

The following URLs are identical

```
http://localhost/jarvis/jarvis.pl/demo/boat_filter/a
http://localhost/jarvis/jarvis.pl/demo/boat_filter?class_name=a
```

The first is a RESTful parameter. It is the first RESTful parameter and is assigned to parameter "1". The second case is a CGI GET supplied user parameter named "class_name". Note that when supplying RESTful parameters, it is not possible to supply e.g. parameter "3" without also defining "2" and "1" as at least empty strings. e.g.

```
http://localhost/jarvis/jarvis.pl/demo/boat_filter/a//c
```

This will supply RESTful parameter "1" → "a", "2" → "", and "3" → "c". If you wished to supply the second parameter as NULL, you would need to use the CGI GET syntax with "?" and "&".

To supply default values for user parameters, use the application configuration file:

```
<jarvis>
    <app>
        <default_parameters>
            <parameter name="max_rows" value="500"/>
        </default_parameters>
```

The above configuration would ensure that "{{max_rows}}" always evaluated to a default value of 500 if it was not specified by the client in a request. However, in this example, be aware of the limitations of SQL placeholders.

Jarvis uses SQL placeholders in all queries for maximum security and efficiency. This means that each variable is replaced by a prepared statement placeholder '?' in the prepared SQL.

However, there are limits in the power of prepared statement placeholders. E.g. the following are invalid when “use_placeholders” = “yes”.

```
<dataset><select>SELECT * FROM t WHERE c LIKE '%{{filter}}%';
```

Instead, use Postgres's POSIX RE matching operator '~', or else use:

```
<dataset><select>SELECT * FROM t WHERE c LIKE '%' || {{filter}} || '%';
```

Other limitations also apply, e.g. the number of lines in SQL Server SELECT TOP my not be a placeholder parameter.

Finally, note that by default, the prepared statement compiler will not know the data type of these substituted variables. In Postgres and other databases you may often need to provide it hints by using ::<type>, e.g.:

```
WHERE ({{1|class_name}}::text IS NULL) OR class ~ {{1|class_name}}
```

7.4 Secure Dataset Parameters

When supplying user variables via CGI GET and via JSON/XML request body, the parameter names must be limited to alphanumeric, plus underscore and hyphen. They may contain a single leading hyphen, but not two. The first non-hyphen character must be [a-zA-Z]. Parameter names are always case-sensitive.

E.g. the following are invalid as user-supplied parameter in a Jarvis dataset request:

- my(param)
- __my_param
- _1param
- 1

The reason for the limitation on numeric parameters is to avoid conflict with the RESTful parameters which are named simply “1”, “2”, etc. The reason for the limitation of the leading double-underscore is because all parameters beginning with double-underscore are secure parameters, supplied by Jarvis. The secure parameters substituted into SQL statements are:

Attribute	Notes
__username	The logged-in username. Never the empty string.
__group_list	The comma separated group list. May be empty string.
__group:<g1>	Evaluates to “1” if the user is in group <g1>. NULL otherwise.
__group:<g2>	Evaluates to “2” if the user is in group <g2>. NULL otherwise.

Table 16: Jarvis Secure Variables

The following pattern is common in SQL for Jarvis datasets. Imagine that table t contains records owned by the username defined in column “t.owner”. Users in group “admin” may read and write all records. Other uses may see (read-only) just their own records.

```
<dataset read="*" write="admin">
    <select>
SELECT * FROM t WHERE (owner = {{__username}}) OR ({{__group:admin IS NOT NULL)
    </select>
    <update>...</update>
```

```
</dataset>
```

7.5 Modifying Datasets – Single Modification

Now we are ready to modify some data. To do this, we submit an http request with request method PUT (update), POST (insert), or DELETE (delete). These requests may specify either a single record, or an array of one or more records.

> *Note for clients written in Adobe Flex, the framework you use may force you to use only GET or POST. In this case you may specify an override method (POST, POST, DELETE) to Jarvis by passing the "_method" parameter as a CGI parameter in the GET or POST request.*

In return you will receive a response body containing a matching single record, or a matching array of one or more records. Let us consider a practical case. Here is a POST request body to insert a record into the "boat" table in the demo database in the single record (non-array) case.

The Content-Type must be "application/json" or "text/json" for JSON, and "application/xml" or "text/xml" for XML.

In JSON the request is as follows:

```
(Request-Line) POST /jarvis/jarvis.pl/demo/boat HTTP/1.1
Content-Type      application/json; charset=UTF-8
{
    "id":0,
    "name":"New Boat Name",
    "class":"Makkleson",
    "registration_num":0,
    "_record_id":1007
}
```

Corresponding request in XML:

```
(Request-Line) POST /jarvis/jarvis.pl/demo/boat HTTP/1.1
Content-Type      application/xml; charset=UTF-8
<request>
    <id>0</id>
    <name>New Boat Name</name>
    <class>Makkleson</class>
    <registration_num>0</registration_num>
    <_record_id>12</_record_id>
</request>
```

The SQL in the <insert> statement is:

```
    <insert returning="yes">
INSERT INTO boat (name, registration_num, class, owner, description,
change_user, change_date)
VALUES ({{name}},
        NULLIF ({{registration_num}}, 0),
        NULLIF (BTRIM ({{class}}), ''),
        NULLIF (BTRIM ({{owner}}), ''),
        NULLIF (BTRIM ({{description}}), ''),
        {{__username}}, now())
RETURNING {{_record_id}}::integer as _record_id, id;
    </insert>
```

The user parameters in the JSON record are substituted into the SQL, along with any server side "default_parameters" values, and any secure parameters such as {{__username}}. The statement is prepared and executed. Because the attribute "returning" is defined as "yes", then the insert statement results are fetched and returned in a "returning" parameter of the resulting JSON.

The response is correspondingly a single record, e.g response in JSON:

```
(Status-Line)     HTTP/1.1 200 OK
Content-Type      text/plain; charset=ISO-8859-1
{
   "success" : 1,
   "returning" : [
      {
         "_record_id" : "1009",
         "id" : "16"
      }
   ],
   "modified" : 1
}
```

Response in XML:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<?meta name="GENERATOR" content="XML::Smart/1.6.9 Perl/5.010000 [linux]" ?>
<response success="1" modified="1">
  <returning _record_id="1206" id="639"/>
</response>
```

In either case, the top-level attributes of the returned object are as follows.

Attribute	Notes
success	Overall success of the modification request. Success = 1 if the update succeeded or 0 if the update failed. For array modifications, success = 1 only if all array modifications succeeded. Otherwise success = 0 indicates one or more rows failed.
modified	The total of records modified. For an array modification, the top level "modified" value is the sum of all of the "modified" values in all of the "row" attributes. This attribute is present only if success = 1.
message	This is present only if one of the updates failed. For a non-array modification this is the error message for the failed update. For an array modification, this is a copy of the error message for the first failed update in the "row" sub-array. i.e. for a failed array modification, the first error message is always present twice in the returned result.

Attribute	Notes
returning	This is present at the top level only for a non-array modification where the "returning" clause is specified in the dataset definition, and where the SQL operation did in fact return one or more rows. The attributes of the returning entries are those returned columns specified in the SQL definition. The most common use of the returning clause is returning the auto-generated serial ID value for an inserted row. For an array modification, the "returning" elements are found within each of the "row" update return results. Note that returning is an array (JSON) or repeatable element (XML), because it is possible for a single insert/update statement to return more than one row.
row	For an array modification request, the one or more "row" elements in the result returns the results of each individual modification request in the array. This attribute is present only if success = 1.

Table 17: Top-Level Attributes of Array Modification Request

7.6 Modifying Datasets – Array of Modifications

The array update case allows multiple transactions of the same type to be performed at once, for better throughput. Only transactions of the same type (i.e. either UPDATE or INSERT or DELETE) may be performed within a single Jarvis http request. If you require a mix of transaction types, you must send separate requests for each type.

If your database supports the MERGE statement (e.g. recent Oracle versions, or PostgreSQL, or SQL Server 2008) then you could write your <update> SQL statement in your dataset definition file to be a combined insert/update statement. Alternatively you could write a stored procedure to do the same thing.

For SQL Server, see: http://weblogs.sqlteam.com/mladenp/archive/2007/07/30/60273.aspx

Request Format

The format of the array modification request is simply an array of single modifications. Note that an array with one element (an array modification) will generate a different response from a single modification (not in an array).

Here is a request updating one toggle to true and another toggle to false. Request in JSON:

```
[
    {"key":"music","name":"Reprints","toggle":true},
    {"key":"music","name":"Second Hand","toggle":false}
]
```

In XML an parallel structure with <request> and <row> tags. Note that the parameters may be given as attributes OR elements. E.g. the following two XML requests will give identical results.

```
<request>
  <row key="music" name="Reprints" toggle="true"/>
  <row key="music" name="Second Hand" toggle="false"/>
</request>
```

Or equally:

```
<request>
  <row><key>music</key><name>Reprints</name><toggle>true</toggle></row>
  <row><key>music</key><name>Second Hand</name><toggle>false</toggle></row>
</request>
```

The response in JSON format might be:

```
{
   "success" : 1,
   "row" : [
      {
         "success" : 1,
         "modified" : 1
      },
      {
         "success" : 1,
         "modified" : 1
      }
   ],
   "modified" : 2
}
```

Or in XML the response might be:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<?meta name="GENERATOR" content="XML::Smart/1.6.9 Perl/5.010000 [linux]" ?>
<response success="1" modified="2">
  <results>
    <row modified="1" success="1"/>
    <row modified="1" success="1"/>
  </results>
</response>
```

Note that the preceding example does not show the optional returning clause

The attributes of each returned "row" entry are as follows.

Attribute	Notes
success	Success result of this individual modification. Value 0 (success) or 1 (failed).
modified	Number of rows affected by this individual modification.
message	If not success, this is the error message for this individual modification.
returning	Returned information. Only present if "returning" = "yes" was specified for this modification type in the dataset definition, and if the modification actually returned one or more rows. Elements of the returning clause are as specified in the SQL.

Table 18: Attributes of the "row" Sub-Elements in Array Modification Response

7.7 Modifying Datasets – Array of Mixed Modifications

In the previous section describing array modifications, all of the changes in the transaction needed to be of a single type – e.g. all "update", all "insert", all "delete", etc. This is according to the RESTful standard, which requires the "method" to be specified at the top level of the request.

However, in practical terms, it is often desirable to perform a single transaction which consists of a mixture of insert/update/delete on the same dataset. Jarvis does support that, with the "MIXED" method request.

Simply specify "MIXED" as the request method. Then, for each row, specify the "_ttype" parameter as one of the per-row attributes or elements.. E.g.

```
<request>
  <row><_ttype="update" key="music" name="Reprints" toggle="true"/>
  <row><_ttype="insert" key="music" name="Second Hand" toggle="false"/>
  <row><_ttype="delete" key="music" name="Underfunded"/>
</request>
```

All of these modifications will be performed within a single begin/end transaction, including (if defined) any before and/or after statements as described subsequently.

7.8 Before & After Statements

The "before" and "after" elements in a dataset allow you to specify optional SQL to be executed at the beginning and end of a modification transaction (either array or single). This allows you to perform the equivalent of "Pre-Commit" triggers in databases which do not otherwise support them.

When performing array modifications, all changes are performed within a bounding begin/end transaction. The actual sequence for single http/s modification request is as follows.

- Begin transaction.
- Perform "before" SQL (if configured for this dataset).
- Perform the one single or one-or-more array modifications in the request body.
- Perform "after" SQL (if configured for this dataset).
- Commit transaction.

An error at any stage will cause the entire transaction to be rolled back.

Note that any "before" or "after" statements may contain references to Secure Dataset Parameters (such as "{__username}" and "{__group_list}". They may also access RESTful arguments as specified in the URL e.g. "{1}" or "{2}".

However, "before" and "after" statements naturally do not have access to any of the per-row parameters in the request body. Note also that the "before" and "after" statements are not invoked for "select" operations, only for modification operations.

7.9 Failed Modifications

Here is an example of a case where two modifications were requests, but one failed. All changes are rolled back:

```
{
   "success" : 0,
   "message" : "ERROR:  duplicate key violates unique constraint \"boat_pkey\""
}
```

Note here:

- Even though the first modification succeeded, it was rolled back.
- The first failing error message is shown.

- The “row” elements and “modified” keys are not present.

7.10 Dataset Transformations

Refer back to the original “boat_class.xml” example given at the start of this chapter. When fetching data, the SQL in the <select> used COALESCE to convert NULL into empty strings to present to the client.

Similarly when storing data, the <update> and <insert> SQL used BTRIM to trim leading/trailing space from strings supplied from the client. Then it used NULLIF to convert empty strings into NULL when writing to the database.

If your database schema and client uses the same conventions, then the <transform> element can be specified to perform these tasks, greatly simplifying the SQL that must be written. The following dataset definition is equivalent.

```
<dataset read="*" write="*">
    <transform store=”trim,null” fetch=”notnull” />
    <select>
SELECT id, class, active, description FROM boat_class ORDER BY class
    </select>
    <update>
UPDATE boat_class
SET class = {{class}}, active = {{active}},
    description = {{description}},
    change_user = {{__username}}, change_date = now()
WHERE id = {{id}};
    </update>
    <insert returning="yes">
INSERT INTO boat_class (class, active, description, change_user, change_date)
VALUES ({{class}}, {{active}}, {{description}}, {{__username}}, now())
RETURNING {{_record_id}}::integer as _record_id, id;
    </insert>
    <delete>DELETE FROM boat_class WHERE id = {{id}};</delete>
</dataset>
```

The “store” and “fetch” attributes specify the transformations to perform when fetching (select) and when storing (insert, update, delete). The attribute value is a comma-separated list of transformations. The following are currently supported and are applied in this order:

Transformation	Notes
trim	Leading and trailing spaces will be removed.
null	Empty strings will be converted to NULL. Occurs after trimming.
notnull	NULL values will be converted to empty string.

Table 19: Transformation keywords for “store” and “fetch”

8 Exec & Plugin

8.1 Exec

The “exec” configuration entry allows you to define commands which should be executed on the server side, with the output results returned to the client via the http response. The return is synchronous, i.e. the client is expected to wait for the response. Common exec commands may be e.g. to run a reporting program such as Jasper Reports.

None or more “exec” configurations may be defined. The dataset name must not conflict with any other “exec” dataset, any “plugin” dataset, any regular dataset, or any special dataset.

When invoking an “exec”, use the GET or POST methods. Specify the “exec” dataset name in the URL, e.g. if the “dataset” attribute of the “exec” element is “echo” then access the dataset with the following URL.

```
http://localhost/jarvis/jarvis.pl/demo/echo/
```

The configuration for an “exec” element is as follows:

Attribute	Default	Notes
dataset	<none>	Dataset name which is to be interpreted as an “exec” rather than a regular dataset. Mandatory parameter.
access	<none>	This lists the group names that the logged-in user must belong to one of before they can access this “exec”. Specify a group name, comma-separated list, “*” or “**”. See the documentation for “read” and “write” groups on regular datasets for further details.
command	<none>	This is the command line to be executed. Mandatory parameter.
add_headers	no	If yes, Jarvis will add Cookie, Content-Type and Content-Disposition headers to the response. Otherwise the exec command is entirely responsible for printing ALL headers. Note that if add_headers is no, be sure to add the header 'Cache-Control' and set the value to 'no-cache' yourself in the outgoing response if you want to avoid responses being cached by IE.
filename_parameter	<none>	Only used if add_headers is “yes”. We expect a CGI parameter of this name to be present and to contain the returned filename.
default_filename	<none>	Only used if add_headers is “yes”. Default filename to use if no filename_parameter is defined or if the CGI parameter named by filename_parameter is not present.

use_tmpfile	depends	Whether or not the command should write its output to a temporary file or not. The default is no under non-MS Windows operating systems, and yes for MS Windows. This is required under MS Windows with Apache otherwise output will be corrupted. If this is set to "yes", or the system is running on MS Windows, the parameter "__use_tmpfile" is passed through on the command line and references the filename of the file into which the command's output should be placed.

Table 20: Exec Configuration Attributes

A sample "exec" is defined in the demo application, as follows:

```
<jarvis>
    <app>
        <exec use_tmpfile="no" dataset="echo" access="*" command="echo"
            add_headers="yes" filename_parameter="filename"/>
```

This uses the Unix "echo" program to simply echo its parameters back to standard output. We will use this to demonstrate the parameters available to "exec" datasets.

Environment Variables

The environment variables to the exec are the same as to the Jarvis script itself. Refer to the Apache documentation for details.

In addition, if the "debug" attribute is "yes" for this application's main configuration, then the environment variable "DEBUG" will be set to "DEBUG=1" when calling the exec process.

Command Line Parameters

The exec parameters to the command line are essentially those passed into SQL dataset statements. They will include:

- Any GET or POST parameters supplied in the request, subject to the standard permitted syntax as described in the regular dataset section. i.e. user-supplied parameters may not start with a double-underscore, may not include special characters, etc.
- Any RESTful parameters specified in the URI. These are passed as "p1", "p2", etc.
- The Jarvis secure parameters beginning with double-underscore. These include __username, __group_list, etc.
- Any default parameters configured in the <default_parameters> section.

E.g with the supplied demo.xml file, consider the following request.

```
http://localhost/jarvis/jarvis.pl/demo/echo/v1//v2?he=th'is&she=that
```

The command line executed will be:

```
echo __group:admin='1' __group_list='admin' __username='admin' he='th'\''is'
max_rows='500' p1='v1' p2='' p3='v2' she='that'
```

The parameters are quoted to make them safe for the command shell.

Output & Headers

The exec process should write its output to standard output. If "add_headers" is "no" then the program is also responsible for writing HTTP headers and a blank line before beginning output.

If "add_headers" is "yes" then Jarvis will add the content type headers. In addition, Jarvis can add a Content-Disposition header if you supply the "default_filename" and "filename_parameter" parameters.

The "filename_parameter" configuration parameter specifies the name of the client-given CGI parameter to evaluate to determine the returned filename. If not present, the value of the "default_filename" parameter will be used. This will be used as the filename in the "Content-Disposition" header when returning the output to the client. The filename suffix will be examined to determine the appropriate "Content-Type" header value to use.

Exec output may include binary data. For a practical example, see the "png" exec in demo.xml.

8.2 Plugin

The "plugin" configuration entry allows you to define a dataset which is fetched by executing a Perl module on the server side. The Perl module will have access to the Jarvis database connection, and will have the user authentication tasks already performed. This makes is simple to add new server-side functionality which cannot be easily represented in a SQL statement.

Jarvis plugin datasets should be accessed by the GET or POST methods. None or more "plugin" elements may be configured.

The configuration attributes for a "plugin" element are as follows.

Attribute	Default	Notes
dataset	<none>	Dataset name which is to be interpreted as a "plugin" rather than a regular dataset. Mandatory parameter.
access	<none>	This lists the group names that the logged-in user must belong to one of before they can access this "exec". Specify a group name, comma-separated list, "*" or "**". See the documentation for "read" and "write" groups on regular datasets for further details.
lib	<none>	This is an additional directory to be optionally added to the @INC path when loading this module. If the necessary path for the module is defined using the global <default_lib> element in the configuration, then this attribute is not necessary.
module	<none>	This is the Perl module name with subdirectories separated by '::' and ending in '.pm'.
add_headers	no	If yes, Jarvis will add Cookie, Content-Type and Content-Disposition headers to the response. Otherwise the plugin module is entirely responsible for printing ALL headers. Note that if add_headers is no, be sure to add the header 'Cache-Control' and set the value to 'no-cache' yourself in the outgoing response if you want to avoid responses being cached by IE.

Attribute	Default	Notes
filename_parameter	<none>	Only used if add_headers is “yes”. We expect a CGI parameter of this name to be present and to contain the returned filename.
default_filename	<none>	Only used if add_headers is “yes”. Default filename to use if no filename_parameter is defined or if the CGI parameter named by filename_parameter is not present.
parameter	<n/a>	One or more sub-elements defining static parameters to be passed to the plugin. Each “parameter” element has a “name” and “value” attribute. This allows site-specific plugin configuration to be stored in a common, accessible location. This also allows one Perl module to be used by different “plugin” instances within the same application, each with their own “dataset” name and each with slightly different behavior.

Table 21: Plugin Configuration Attributes

A sample plugin is included with the demo module. It is configured as follows:

```
<jarvis>
    <app>
        <plugin dataset="plug" access="*" lib="/opt/jarvis/demo"
                module="plugin::Demo" add_headers="yes"
                filename_parameter="filename">

            <parameter name="smtp_server" value="192.168.70.100"/>
            <parameter name="category" value="kiwisaver"/>
        </plugin>
```

The source of the sample “Demo.pm” file is as follows:

```
use strict;
use warnings;

sub plugin::Demo::do {
  my ($jconfig, %args) = @_;

  my $output = "";

  &Jarvis::Error::debug ($jconfig, "Running the Demo plugin.");

  # This demonstrates that the plugin can access Jarvis::Config info.
  $output .= "APP: " . $jconfig->{'app_name'} . "\n";
  $output .= "Username: " . $jconfig->{'username'} . "\n";
  $output .= "CGI myval: " . ($jconfig->{'cgi'}->param('myval') || '') . "\n";

  # This demonstrates XML-configured parameters to the plugin.
  $output .= "CAT: " . $args{'category'} . "\n";

  my $dbh = &Jarvis::DB::Handle ($jconfig);
  $output .= "Boats: " . $dbh->do("SELECT COUNT(*) FROM boat");

  return $output;
}
```

```
1;
```

This demonstrates:

- Using the Jarvis debug methods.
- Accessing the Jarvis Config "jconfig" object.
- Accessing the static configured plugin parameters.
- Accessing the Jarvis-supplied database connection.

Accessing this plugin via the following URL:

```
http://localhost/jarvis/jarvis.pl/demo/plug?myval=test
```

The output is:

```
APP: demo
Username: admin
CGI myval: test
CAT: kiwisaver
Boats: 1
```

Output & Headers

Jarvis offers support for "add_headers", "filename_parameter" and "default_parameter" handling for "plugin" datasets identical to "exec" datasets.

Exception Handling

To handle an exception in your plugin, simply call "die". You may optionally set an explicit HTTP status to return. Otherwise Jarvis will return a "500 Internal Server Error".

```
$jconfig->{'status'} = '404 Not Found';
die "Plugin cannot continue, moon not found in expected phase.";
```