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 can be configured to use Apache's "mod_perl" for better performance than simple CGI.

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

Version Tracking (Since v2.0.0):

Version	Date	By	Comment
3.0.1	7-May-2010	JXC	Added support for hooks.
3.0.2	10-May-2010	JXC	Adempiere login module now creates session.
3.0.3	27-May-2010	JXC	Added Drupal6 login module.
3.0.4	2-Jun-2010	JXC	Allow Jarvis Drupal6 login to inherit Drupal sessions.
3.0.5	10-Jun-2010	JXC	Ability to disable <sessiondb>. Change Drupal6 Login config.
3.0.6	22-Jul-2010	JXC	Hook classes no longer need to implement all hook methods.
3.1.0	23-Jul-2010	JXC	Additional hook parameters, plus per-dataset hooks.
3.1.1	28-Jul-2010	JXC	Add user_id_column to Login::Database.
3.1.2	2-Aug-2010	JXC	Add after_login hook. Tweak finish hook calling sequence.
3.2.0	2-Aug-2010	JXC	Add return_status, return_fetch and return_store hooks.
3.2.1	2-Aug-2010	JXC	Added textual substitution parameters.
3.2.2	4-Aug-2010	JL	Added Eksblowfish encryption for passwords.
3.2.5	22-Aug-2010	JXC	Added "lib" and "default_libs" support to login modules.
3.3.0	18-Oct-2010	JXC	First Debian package. Changed default directory paths.
3.3.1	26-Oct-2019	JXC	Add "require_https" global application configuration option.
3.3.2	24-Nov-2010	JXC	Added more info on transaction behaviour.
4.0.0	21-Dec-2010	JXC	Added support for more than one dataset fetch per request.
4.0.4	3-Feb-2010	JXC	Allow override of mime type for plugins.
4.1.1	4-Feb-2010	JXC	Support named database connections.
4.1.2	18-Feb-2010	JXC	Document "dump" and "debug" for Exec and Plugin config.
4.1.4	7-Mar-2010	JXC	Add "word2html" transformation tag.
4.1.5	18-May-2010	JXC	Improved Drupal6 group determination.

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.

Jarvis (including documentation) is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

Jarvis is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with Jarvis. If not, see <http://www.gnu.org/licenses/>.

This software (including documentation) is Copyright 2008 by Jonathan Couper-Smartt.

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.

2.3 Jarvis Tracker Icons

The Jarvis Tracker icons, as can be found in the 'tracker/htdocs/style' directory are copyright FamFamFam (http://www.famfamfam.com/lab/icons/silk/), and are used under the Creative Commons Attribution License v3.0 (http://creativecommons.org/licenses/by/3.0/). We extend our thanks to the creator of these icons for their quality and availability.

3 Installation

3.1 Unpacking

Install the appropriate package for your system. The following directory tree should be created. Paths are given for Debian, and for "other" systems.

Path	Notes
Debian: **/usr/share/jarvis/cgi-bin** Other: **/opt/jarvis/cgi-bin**	This contains jarvis-agent, which is the CGI script which accepts requests over http/https when not using mod_perl. The jarvis-agent file is relatively simple. Most of the functionality is in the lib/Jarvis/*.pm files in the lib directory.
Debian: **/usr/share/jarvis/demo** Other: **/opt/jarvis/demo**	Contains demo files which comprise a simple application showing how to use Jarvis with ExtJS and PostgreSql.
Debian: **/usr/share/doc/jarvis** Other: **/opt/jarvis/docs**	Documentation including this file.
Debian: **/usr/share/jarvis/etc** Other: **/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. Apache configuration files are also placed here.
Debian: **/usr/share/jarvis/htdocs** Other: **/opt/jarvis/htdocs**	This contains a helper JavaScript include file for use with ExtJS.
Debian: **/usr/share/jarvis/lib** Other: **/opt/jarvis/lib**	This contains the majority of the Jarvis functionality as a set of Perl modules.
Debian-Only: **/var/lib/jarvis**	Working directory for SQLite database files for Tracker and Demo applications, also recommended location for CGI session files.

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 Ubuntu 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
sudo apt-get install libtext-csv-perl
sudo apt-get install libio-string-perl
sudo apt-get install libmime-types-perl
sudo apt-get install libdbi-perl
sudo apt-get install libdbd-sqlite3-perl
```

REQUIRED (if using mod_perl)

```
sudo apt-get install libapache2-mod-perl2
sudo apt-get install libapache-dbi-perl
sudo apt-get install libapache2-request-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)
sudo apt-get install libcrypt-eksblowfish-perl
                            (Login = Database, Authentication = 'eksblowfish')
```

3.3 Apache2 Configuration – Without mod_perl

If performance is not a significant factor, you can simply run Jarvis as a CGI script without using mod_perl. Add the following lines to your server configuration.

Note: When installing the Debian package, this setup is done automatically.

For Debian:

```
ScriptAlias /jarvis-agent "/usr/share/jarvis/cgi-bin/agent.pl"
Alias /jarvis/ "/usr/share/jarvis/htdocs/"
```

For Other Systems:

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

These lines are contained in the supplied "apache/jarvis.conf" file. Under Linux you can generally copy or soft-link this file into the /etc/apache2/conf.d directory.

3.4 Apache2 Configuration – With mod_perl

Jarvis supports the Apache mod_perl mechanism. Without mod_perl, the webserver must compile and execute the Perl code for each request. Equally, without mod_perl, a new database connection must be made for each request.

Using Apache's mod_perl, the script is compiled once, and re-executed for each request. Database connections can also be re-used. The overhead for each request is greatly reduced. Refer to the mod_perl home page (http://perl.apache.org/) for further details.

Once mod_perl is configured and available, add the following lines to your server configuration.

Note: When installing the Debian package, this setup is done automatically.

For Debian:

```
PerlRequire /etc/jarvis/apache/startup.pl
<Location ~ "/jarvis-agent/.*">
    SetHandler perl-script
    PerlResponseHandler Jarvis::Agent
</Location>

Alias /jarvis/ "/usr/share/jarvis/htdocs/"
```

For Other Systems:

```
PerlRequire /opt/jarvis/etc/apache/startup.pl
<Location ~ "/jarvis-agent/.*">
    SetHandler perl-script
    PerlResponseHandler Jarvis::Agent
</Location>

Alias /jarvis/ "/opt/jarvis/htdocs/"
```

These lines are contained in the supplied "apache/jarvis-mod-perl.conf" file. Under Linux you can generally copy or soft-link this file into the /etc/apache2/conf.d directory.

Note that if you are co-hosting Jarvis on a server running drupal from the root path, you will probably need to exclude jarvis-agent from the drupal rewrite rule. E.g. something like the following needs to be added to /usr/share/drupal6/.htaccess.

```
  RewriteCond %{REQUEST_URI} !^/jarvis-agent/
```

Otherwise you will find that Drupal attempts to handle your jarvis-agent requests. Note that this is only required when running with mod_perl. When running via a simple ScriptAlias (see below), Apache will give priority to the Alias match over the Drupal match, and the problem does not arise.

3.5 Relocating Jarvis

This example shows Jarvis installed into the default locations. If you install jarvis into another location, then you must change the Jarvis installation path in the following locations:

- **Modify the Apache configuration (as described above).**

- **Modify the Jarvis module path in etc/apache/startup.pl (only if using mod_perl).**

The startup.pl script is a Perl fragment which tells mod_perl where to find the module containing the mod_perl handler for the Jarvis web service agent. When relocating Jarvis, modify the path to specify the new location of the Jarvis lib directory.

By default it contains the following.

```
use Apache::DBI;

use lib qw(/usr/share/jarvis/lib);      # For Debian.
use lib qw(/opt/jarvis/lib);            # For other systems.

1;
```

3.6 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-agent/

You should see the error response.

```
Missing path info.  Send http://.../jarvis-agent/<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(s).

- 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" require_https="no">
        <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/myapp/edit/etc/dataset</dataset_dir>

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

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

        <tracker logins="yes" requests="yes" errors="yes">
            <dbfile>/var/lib/jarvis/tracker/tracker.db</dbfile>
        </tracker>

        <login module="Jarvis::Login::Database">
            <parameter name="user_table" value="staff"/>
            <parameter name="user_id_column" value="id"/>
            <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"/>
            <parameter name="encryption" value="md5"/>
            <parameter name="salt_prefix_len" value="2"/>
        </login>
        <database connect="dbi:Pg:dbname=test" username="" password=""/>
        <database name="secondary" 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/myapp/tmp/sessions"/>
        </sessiondb>

        <exec dataset="sample" access="*"
          command="/path/to/program"
          add_headers="yes" filename_parameter="filename"/>

        <plugin dataset="csvexport" access="admin"
          lib="/path/to/plugin"
          module="Test::ExportToCsv" add_headers="yes"/>

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

        <hook module="MyHooks::CustomAudit"/>
    </app>
</jarvis>
```

Each of these will be discussed in detail.

4.2 List of Parameters

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:

```
/etc/jarvis/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.
require_https	no	Require all connections for this application to use HTTPS.
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_limit	Name of the CGI parameter which will be used as the page size for server-side paging (performed after fetching).

Attribute	Default	Notes
page_start_param	page_start	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'.
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.
tracker	See Notes	This section enables the optional request/error tracking feature of Jarvis. See the subsequent section which describes this in detail.
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 is a sub-element which defines the database connection parameters for the database(s) you intend to access. If there is more than one, they should have unique "name" parameters. The default name is "default".
sessiondb	<n/a>	This defines the configuration for the cookie store. If this is omitted, then Jarvis will not maintain a login session cookie. Instead it will perform a full login check for every request. This can make sense in some situations.
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).
hook	<n/a>	One or more optional custom Perl modules on which we call specially named methods at key points in the Jarvis processing.
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 Parameter Details

4.3.1　　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.

4.3.2　　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.

4.3.3　　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.

4.3.4　　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

4.3.5 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 "returned" as the number of data rows from the sliced page.

- 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.

4.3.6 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.

4.3.7 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>/path/to/dataset</dataset_dir>. A unix-based slash has been used for these examples.

Client Requests	Resulting XML File
my-set	/path/to/dataset/my-set.xml
.my-set	[Error, may not start with dot]
my-set.	[Error, may not end with dot]
folder.myset	/path/to/dataset/folder/myset.xml
myset.xml	/path/to/dataset/myset/xml.xml

Table 3: Examples of Resolving Dataset Names

4.3.8   Configuration: default_libs

A list of library paths to @INC into the process before attempting to load and execute a plugin, login, or hook module.

For example:

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

When a plugin, library or hook is to be executed, any default library paths defined will be added to the Perl include path. After that, any "lib" parameter specifically set on the plugin, library or hook will be added, and will take precedence over the default_libs path.

4.3.9   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.

4.3.10   Configuration: login

Every application must have a login module defined. Jarvis offers half a dozen login modules to handle commonly used situations.   You can easily add your own by copying and modifying the supplied defaults.

Login configuration is the subject of a separate chapter. A complete list of Jarvis-supplied modules is given in that section, along with all associated configuration parameters.

4.3.11   Configuration: database

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

Attribute	Default	Notes
name	default	For a single database connection, the default name of "default" is fine. If your application wishes to use more than one connection, use this field to ensure the names are unique.
connect	dbi:Pg:<app-name>	This is a DBI connection string. See the DBI documentation.
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)

4.3.12   Configuration: sessiondb

This defines the session database. Sessions are created and managed with the Perl CGI::Session module. Refer to the documentation for that module for a more detailed description of configuration options available for the store and expiry parameters.

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	<APPNAME>_CGISESSID	Default session cookie name is <APPNAME>_CGISESSID. If you are serving multiple Jarvis applications on the same host, then ensure that each one uses a separate cookie name.

expiry	+1h	Specifies the extension period for the cookie after every successful Jarvis interaction.
sid_source	cookie	The places where Jarvis should look for the SID. This is a comma-separated string of the following options: • cookie - Look in the cookie (as per the 'cookie' attribute). • url - Look in the parameters passed in the URL. The URL parameter is expected to be the name provided in the "cookie" attribute. The order these are listed is relevant. For example "url,cookie" means that the URL parameters will be looked at first, then, if the relevant parameter cannot be found, the cookie list will be searched.

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" sid_source="cookie">
    <parameter name="Directory" value="/home/myapp/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

Note that if you omit the <sessiondb> tag entirely, then Jarvis will not maintain a session cookie. Instead, every Jarvis request will invoke the full Login check sequence as configured by your <login> settings.

The following considerations apply:

With Jarvis <sessiondb> and cookie.	Without Jarvis <sessiondb> or cookie.
More secure. No need to pass "username" and "password" in each request.	*This is a key point. If the login settings you are using requires a username and/or password then in general a <sessiondb> is recommended.*
Less secure. If the user is blocked in the underlying database, they can still continue to access Jarvis as long as the cookie is valid.	More secure. If the user is blocked in the user database, they are immediately locked out of Jarvis access.

More efficient. No need to re-perform the full login sequence for each request.	More secure. If login configuration is setup to check on Certificate or IP address, it would be better to perform this check on every request.
User Changes Ignored. If the Jarvis session is piggybacking on another session (e.g. a Drupal6) then re-logging to a new Drupal user will not be picked up by Jarvis. It will blindly and wrongly continue its session for the old user.	User Changes Detected: Any changes to the parent session will be immediately detected by Jarvis.

Table 7: Advantages/Disadvantages of Jarvis <sessiondb>

4.3.13   SID source: URLs vs Cookies

With the "sid_source" parameter available in the sessiondb configuration, client system can be written to provide the necessary session ID information via either the URL, cookies, or both. This section details the considerations when dealing with each.

1. Cookie-only based authentication. The default approach to session management is for Jarvis to store session ID's in a HTTP cookie. This cookie is returned to the client with every successful request to Jarvis. This approach:

    a) Requires no specific support by client applications. Web browsers will transfer the cookie automatically with every request.

    b) Allows web-browser applications to have the same session across all open windows. E.g. this allows users to open many browser tabs/windows into the application, and all of these will use the same session ID.

    c) Allows users to, within the session timeout, re-access web based systems without retyping their username/password.

2. URL based authentication.   This approach requires the session ID to be passed in a URL parameter (or HTTP POST parameter) on each request. Unlike cookie based systems, this approach requires explicit support by the client. This approach:

    a) Without explicit support in the application to manage and support it, each new window or tab opened for a web-based application will require the user to log back into the application. In many situations this is highly undesirable.

    b) Conversely, it does mean that web-based applications can be designed to allow a user to log in multiple times to the same system within the same browser – something the cookie based approach does not allow (as cookies are browser-wide, not per window or per tab).

3. URL based authentication, with a cookie-based secondary fallback. This approach, available when sid_source is given as "url,cookie", allows the best of both of the above approaches, but does require explicit support in the application. With this approach:

    a) Jarvis will look for the URL parameter holding the SID, but if not found, look for the cookie.

    b) The client system can by default use cookie based SIDs, and then, when necessary (e.g. when the user wants a secondary login to the application), switch to using URL based SIDs.

c) One example of when this is useful is when most of the time users require only a single session, but for some users it is important to provide the ability to log in as multiple people. With cookie based SIDs users would need to use different browsers (or browser profiles) to get this, and with URL based SIDs users would lose the ability to have multiple tabs open to the same application.

Note in the URL & cookie based authentication method above, Jarvis will ensure that cookies are only used when the SID is retrieved from the cookie. This avoids URL based sessions trampling cookie based sessions.

4.3.14   Configuration: exec & plugin & hook

In addition to any SQL procedures in your dataset, Jarvis also supports several different ways to integrate additional Perl or command line functionality into your application. These are configured in the main application XML file. The different mechanisms are:

- **exec** – An "exec" configuration defines a special dataset which is fetched by spawning a custom sub-process. Use this when you wish to use Jarvis security and pre-processing, but the actually response content is to be provided by an external program, e.g. running reports.

- **plugin** – A "plugin" configuration also defines a special dataset which is fetched by loading a custom Perl module written as a ".pm" file. Use this when you wish to use the Jarvis interface but the dataset content or insert/update is too complex to easily express in SQL, and you wish to implement it in Perl code.

- **hook** – A "hook" is a custom Perl module containing specially named methods which are invoked at key points in then processing of all datasets. Use this when you wish to perform additional security checking, custom logging or auditing for all datasets.

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 chapters on "Exec Dataset", "Plugin Datasets" requests and on "Hook Modules" for further details.

4.3.15   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 in the installed Jarvis "lib" directory:

- ActiveDirectory.pm
- BasicAuth.pm
- Database.pm
- Adempiere.pm
- Drupal6.pm
- None.pm
- Single.pm

You may copy/modify these to create custom login modules.

All modules are configured by a "login" element with the following parameters.

Attribute	Default	Notes
module	\<none\>	Full name of the module to load, e.g. Jarvis::Login::None.
lib	\<none\>	Optional base library directory in which to find the module. Note that the "default_libs" are also searched.
parameter	\<none\>	One or more sub-parameters with "name" and "value" attributes.

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. Mandatory. Secondary controllers are not supported.
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 8: 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 9: 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_id_column" value="id"/>
            <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
dbname	default	The name of the database connection to use. Default = "default".
user_table	<none>	Name of the database table containing username and password columns. Mandatory.
user_id_column	<none>	Name of username ID column. Optional.
user_username_column	<none>	Name of the username column. Mandatory.
user_password_column	<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 group name column in the group table.

Attribute	Default	Notes
encryption	none	Specifies the encryption method for the password. This can be one of: • none – no encryption. • md5 - the md5 hash algorithm is used, with optional salt. • eksblowfish – the Eksblowfish encryption algorithm is used. Salt is hard coded to 16. **Note:** it is strongly suggested that eksblowfish is used, for reasons outlined here: http://paulbuchheit.blogspot.com/2007/09/quick-read-this-if-you-ever-store.html.
salt_prefix_len	0	If configured to a value > 0, then encrypted passwords are encoded with a salt prefix to hamper dictionary attacks. Currently only necessary for the 'md5' encryption method. For MD5, the hex-encoded MD5 hash value is expected to be prefixed by exactly this number of ASCII salt characters. The salt characters are pre-pended to the user-supplied password before generating the MD5 hash.

Table 10: 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".

If the "user_id_column" parameter is defined, then it must specify the name of an additional database column in the user table. When configured, the value from this column matching the supplied username will be stored in a "safe" variable which can be accessed as {{__user_id}} in datasets.

5.5 Adempiere Login

This module performs username and password lookup in the configured Adempiere database. Currently this module supports login for only a single client and organization.

The following tables are referenced:

- ad_user
- ad_user_roles
- ad_role
- ad_window_access

- ad_process

- ad_process_access

Example:

```
<jarvis>
    <app>
        <login module="Jarvis::Login::Adempiere">
```

```
              <parameter name="client_name" value="MyClientName"/>
              <parameter name="org_name" value="MyOrgName"/>
       </login>
```

The parameters are.

Attribute	Default	Notes
dbname	default	The name of the database connection to use. Default = "default".
client_name	<none>	Name of the single Adempiere client (table ad_client) for whom we allow login via this method. Mandatory.
org_name	<none>	Name of the single Adempiere organization (table ad_org) for whom we allow login via this method. Mandatory.
reverse_dns	No	Should we perform reverse DNS to fill the remote_host field in the ad_session table. In some cases reverse DNS can be very slow.
relevant_groups		A comma-separated list of the groups relevant to the application. ADempiere provides a large number of groups, and this parameter can be used to alter the group-list for each user to a smaller subset of their groups.

Table 11: Adempiere Login Module Parameters

The determined group list is a long comma-separated string where each of the many elements is either:

- *role-<role_name>*   For each active user role.
- *read-<window_name>*   For each active window access (read access)
- *write-<window_name>*   For each active window access (write access), and any processes the user has access to.

All spaces and special characters are stripped from the role name and window names.

So for window "My Window", your corresponding dataset is likely to have access settings

```
    read="read-MyWindow,write-MyWindow" write="write-MyWindow"
```

Note that when this module is used to login, any dataset requests will also have access to the following additional variables:

    "__ad_user_id"
    "__ad_client_id"
    "__ad_org_id"
    "__ad_session_id"

These variables contain the corresponding numeric ID values found to match the configured client_name and org_name parameters. These variables can be accessed in a dataset e.g. by specifying {{__ad_client_id}}. These are "safe" variables in the same sense as e.g. the "__username" variable in that they are set purely by Jarvis and cannot be set or modified by the remote client.

5.6 Drupal6 Login

This module performs two functions:

1. Check to see if an existing, valid Drupal session cookie is defined.

2. Perform basic username and password lookup in a Drupal6 database.

The second step is performed only if there is no existing Drupal session cookie available and only if the "allow_login" parameter is enabled.

The following Drupal tables are accessed:

- users

- sessions

You will typically need login to the database as the database owner and grant the webserver user read access to these tables, e.g.:

```
postgres# GRANT SELECT ON users TO "www-data";
postgres# GRANT SELECT ON sessions TO "www-data";
```

Example configuration:

```
<jarvis>
    <app>
        <login module="Jarvis::Login::Drupal6">
            <parameter name="allow_login" value="yes"/>
            <parameter name="admin_only" value="yes"/>
            <parameter name="admin_group" value="admin"/>
        </login>
```

The parameters are.

Attribute	Default	Notes
dbname	default	The name of the database connection to use. Default = "default".
login_type	drupal	*drupal* = We insist on an existing Drupal session cookie. *jarvis* = Jarvis performs session login and management.
admin_only	no	Is the user required to be the primary administrator (uid = 1).
admin_group	admin	What group is assigned to the user with (uid = 1).

Table 12: Drupal6 Login Module Parameters

The determined group list is:

- *<admin_group>*   For the user with uid = 1.
- *<role1>,<role2>*   For all other users, taken from the users_roles table.

Note that when this module is used to login, any dataset requests will also have access to the following additional variables:

"__uid"

This variable contains the uid value from the Drupal6 "users" table. It can be accessed in a dataset e.g. by specifying {{__uid}}. This is a "safe" variables in the same sense as e.g. the "__username" variable in that it is set purely by Jarvis and cannot be set or modified by the remote client.

Login Type "drupal"

When using "login_type=drupal", you must have an existing Drupal session in order to perform a Jarvis login. With this setting, Jarvis will never accept a username and password. Only Drupal may perform the username and password validation.

Jarvis performs its validation by looking for a Drupal session cookie and comparing it against list of cookies in the "sessions" table of the Drupal database. Note that currently the Drupal database must be the same database which Jarvis uses for fetching data for dataset requests. In the future, the Drupal6 module may allow you to specify a different database connection path for user validation.

Note that when using Login Type "drupal" you most likely will want to disable the Jarvis session database, by removing the <sessiondb> tag from your application configuration file. This is so that if the user logs out of Drupal, or changes their Drupal username by re-logging, then Jarvis will immediately detect this change.

Login Type "jarvis"

When using "login_type=jarvis", then Jarvis will ignore any existing Drupal session. Instead, you must create a Jarvis session by providing the username and password of a valid Drupal user. The username and password are checked against the Drupal6 database, but Jarvis will never create a Drupal6 session.

Jarvis will create its own session, stored in the session database configured by the <sessiondb> tag. This session remains entirely independent of any Drupal session that may be created by logging in to Drupal. Logging out of Drupal will not affect your access to Jarvis so long as the Jarvis cookie remains valid.

5.7 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 13: None Login Module Parameters

5.8 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 14: 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 assume that:

- Jarvis has been installed/configured as above, and is available via the web-server.
- The demo.xml file has been soft linked or copied into the Jarvis "etc" directory.
- The directory paths in demo.xml have been correctly configured.

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-agent/demo/__status
```

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

```
http://localhost/jarvis-agent/<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-agent/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-agent/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-agent/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 Fetching 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. These examples are based on the Demo application provided with Jarvis. This uses the JSON encoding for request/response.

Consider the following Jarvis request.

```
http://localhost/jarvis-agent/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",
   "returned" : "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-agent/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" returned="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.
returned	The number of rows returned after 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 15: 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">
    <hook module="MyHooks::PerDatasetHook1"/>
    <before>SELECT pre_request_procedure()</before>
    <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>
```

```
    <after>SELECT pre_request_procedure()</after>
</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.

This example also defines a before-query, an after-query, and a dataset-specific hook module. These are shown for completeness only. In general you will not need to define <hook> definitions in your dataset, and you do not generally require <before> or <after> queries.

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

Attribute	Notes
dbname	Specify which database connection should be used for fetching and storing this dataset. By default, the "default" database descriptor will be used.
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 16: 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.
hook	One or more hook module definitions. Definition is identical to global hooks.
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 17: 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 18: 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 Bind Parameters*

The SQL statements in the dataset query definition contain bind 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 actual name of the bind parameter specified in the dataset query definition may contain only upper/lower-case a-z, digits 0-9, underscore, colon and hyphen. All other characters will be ignored.

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-agent/demo/boat_filter/a
http://localhost/jarvis-agent/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-agent/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 Bind Parameters

When supplying user variables via CGI GET and via JSON/XML request body, the parameter names must be limited to alphanumeric, plus underscore, colon 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 19: 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 Textual Substitution Dataset Parameters

Whenever possible, you should always use the {{parameter}} mechanism to substitute parameters. This is because that mechanism creates bind parameters which are substituted into the statement at execution time using ? placeholders.

However, there are situations where you want to use textual substitution. This is where a parameter is inserted as text into the statement at the time that the statement is prepared.

The most common reason for this is the ability to use the LIMIT or TOP syntax to restrict the number of rows which is fetched. Also this is useful to perform SORT requests inside the database engine itself.

Note that Jarvis does provide a post-processing paging and sorting function. In that case Jarvis fetches all the rows from the database and then uses Perl sorting and slicing to extract the specific subset of records required for the single page. The build-in Jarvis paging and sorting is fine for many situations, however it is not ideal when the full number of rows is very large, or if the database is remotely connected to Jarvis over a slow link. Also, the Jarvis built-in mechanism does not support ordering by multiple columns.

In this case, the dataset creator may use [[arg]] parameter to request textual substitution be done at statement preparation. The parameters to be substituted may include:

- Any RESTful parameters by position number, e.g. [[1]]

- Any query parameters given by the user for a fetch request, e.g. [[myfetcharg]]

- Any "safe" parameter generated by Jarvis or a hook, e.g. [[__username]]

Naturally with textual substitution there always the danger of client SQL injection to consider. Hence the following rules are used:

1. By default, if the parameter value is an integer or floating point, it is unquoted. Exponential

numbers will be recognized as numbers.

2. Otherwise the parameter is quoted using the DBI database handle $dbh->quote(...) method appropriate for this database. According to the DBI documentation, this is safe from SQL injection.

3. If the ":noquote" flag is given, e.g. [[sortorder:noquote]] then the value is not quoted. Instead, all characters except those in the set [0-9a-zA-Z ,_-] are removed.

4. If the ":quote" flag is given, e.g. [[sortorder:quote]] then the value is always quoted.

Note that in order to specify a client-supplied ORDER BY clause containing more than one column you will need to use the ":noquote" flag appended to the textual substitution variable name.

Note that as for bind parameters, the name of the textual substitution parameters specified in the dataset query definition may contain only upper/lower-case a-z, digits 0-9, underscore, colon and hyphen. All other characters will be ignored.

8 Storing Datasets

8.1 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-agent/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-agent/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. Note that under PostgreSQL (and very likely other drivers) an error will occur if you specify 'returning="yes"' but then do not add a RETURNING clause to the SQL. Note that SQLite does not support the RETURNING clause. However, under SQLite if you specify 'returning="yes"' then Jarvis will automatically use the last_insert_rowid() mechanism and will return it as a column named 'id'.
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 20: Top-Level Attributes of Array Modification Request

8.2 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 21: Attributes of the "row" Sub-Elements in Array Modification Response

8.3 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.

8.4 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.

8.5 Transaction and Rollback

It is important to note:

- For "fetch" statements in a dataset (i.e. the <select> attributes), no explicit transaction is performed.

- For "store" statements in a dataset (i.e. <insert>/<update>/<delete> attributes), an outer transaction is automatically applied.

Note that this is not based on the textual content of the SQL, but on which named attribute tag is being executed.

This means that if your <select> attribute SQL performs multiple data modifications as a side-effect, then a failure in a later step will not rollback any earlier changes. If you require a transaction around the modifications, then either define it as an <insert>/<update>/<delete> and use the appropriate REST action to invoke it, or else implement your own transaction within your SQL body.

Here is an example of a case where <insert> element SQL was used with an array of two new rows being requested. In this case, 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.

9 Dataset Transformations

9.1 Introduction

Refer back to the original "boat_class.xml" example given earlier in this document. 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.
word2html	Convert MS Word unicode and 8-bit character to HTML to ensure no binary content in the string. Will brute-force convert characters we don't recognize.

Table 22: Transformation keywords for "store" and "fetch"

10 Fetching Multiple Datasets in One Request

10.1 Introduction

As of Jarvis 4.0, the facility exists to request the results of more than one dataset in a single HTTP request. This is a standard technique for improving the response time of rich Internet applications.

No additional server-side configuration is required to access this feature. Simply perform a normal Jarvis "fetch" request, but specify a comma-separated list of dataset names in your request.

Note that:

- This functionality is available for "fetch" requests on standard SQL datasets only. It does not apply to Exec or Plugin datasets.

- This functionality is available only for JSON and XML format requests. It is not available for CSV format requests.

- You may use a "result_fetch" hook to enhance/modify the returned content of these requests.

- Do not put spaces with the comma separators.

10.2 JSON Multi-Dataset Result Structure

The format for the returned result of a multi-dataset fetch request is necessarily more complex than for a single dataset request. Each dataset is nested inside a structure to separate the data values from other dataset results.

```
{
    "dataset" : {
        "sport_list" : {
            "data" : [
                {
                    "name" : "Golf",
                    "description" : "An outdoor sport."
                },
                {
                    "name" : "Tennis",
                    "description" : "Another outdoor sport."
                },
            ],
            "fetched" : "2",
            "returned" : "2"
        },
        "player_details" : {
            "data" : [
                {
                    "name" : "Aaron Hoffman",
                    "sport" : "Golf",
                    "rank": 323
                },
                {

                    "name" : "William Uchtman",
                    "sport" : "Tennis",
                    "rank": 14002
                }
            ],
```

```
            "fetched" : "2",
            "returned" : "2"
        }
    },
    "logged_in" : "1",
    "error_string" : "",
    "username" : "Guest",
    "group_list" : "admin"
}
```

10.3   XML Multi-Dataset Result Structure

Equally, the XML results structure for a multi-dataset response encapsulates each dataset response in a substructure as the following example indicates:

```xml
<?xml version="1.0" encoding="iso-8859-1" ?>
<?meta name="GENERATOR" content="XML::Smart/1.6.9 Perl/5.010001 [linux]" ?>
<response logged_in="1" username="Guest" error_string="" group_list="admin">
  <dataset fetched="2" name="sport_list" returned="2">
    <data>
      <row description="An outdoor sport." name="Golf"/>
      <row description="Another outdoor sport." name="Tennis"/>
    </data>
  </dataset>
  <dataset fetched="2" name="player_details" returned="2">
    <data>
      <row name="Aaron Hoffman" sport="Golf" rank="323"/>
      <row name="William Uchtman" sport="Tennis" rank="14002"/>
    </data>
  </dataset>
</response>
```

11 Exec Datasets

11.1 Introduction

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-agent/demo/echo/
```

11.2 Configuration

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. The actual dataset given in the Jarvis request must match this, or a sub-dataset of this. E.g. if the exec configuration specifies "jasper" as its dataset name, then the dataset name supplied to Jarvis by the client may be "jasper" or "jasper.ReportName".
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.
mime_type	<none>	Only used if add_headers is "yes". Allow plugin to override the default MIME type by specifying a value to use. This takes precedence over any mime type which may be derived from a filename.

filename_parameter	'filename'	Only used if add_headers is "yes". We expect a CGI parameter of this name to be present and to contain the returned filename. If you wish to use "filename" as a normal parameter to your plugin, you will need to change this configuration.
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.

See following notes on output methods. |
| tmp_directory | depends | This allows you to overwrite the default TMP directory to be used when exec output is spooled to file instead of printed directly to the http client via standard output.

Note that specifying "tmp_directory" will force the use of temporary files, regardless of the value of the "use_tmpfile" parameter. |
| tmp_http_path | <none> | Setting the parameter tells Jarvis to redirect the client to the output temporary file via this URI which must be configured in the web server to point to the temporary directory.

E.g. if "tmp_directory" is "/tmp" and "tmp_http_path" is "/tmp-reports" then the webserver must contain an alias mapping "http://*host*/tmp-reports/" to "/tmp". Use of a dedicated report output directory is recommended for security reasons.

Note that specifying "tmp_http_path" will force the use of temporary files, regardless of the value of the "use_tmpfile" parameter.

Note that specifying "tmp_http_path" will mean that "add_headers" is ignored, as MIME headers are not relevant to a redirection response. |

cleanup_after	0	If set to a non-zero value, this parameter will enable cleanup of files in the temporary directory after the specified number of minutes. Set to zero (default) means "never cleanup". Jarvis will only remove files which are owned by the user running the server script (e.g. www-data). Even so, care should be taken when using a shared temporary directory for exec output.
debug	<n/a>	Enable "debug" output to standard error for this plugin only. This parameter can not be used to disable global debugging.
dump	<n/a>	Enable "dump" output to standard error for this plugin only. This parameter can not be used to disable global dump output.

Table 23: Exec Configuration Attributes

11.3 Output Return Mechanism

Given the above configuration, there are three different ways in which Jarvis might return the output from an exec dataset. These are as follows.

Mechanism	Notes
Redirection	This mechanism is used if and only if "tmp_http_path" is specified. Jarvis will choose a value for "__tmpfile" which is based on the "reportfile" name requested by the client. A random component will be added to the name and it is guaranteed not to exist at the time of exec. The "__tmpfile" parameter is passed to the executable command line. The executable command line write its output in that file. When the execution is complete, Jarvis will send back a "302 Found" response to the client to redirect it to the new output using the specified "tmp_http_path". *This is the recommended approach for cross-platform systems and/or large output and/or binary data.*
Streaming	If "use_tmpfile" and/or "tmp_directory" is specified without specifying any value for "tmp_http_path" then streaming will be used. Jarvis will choose a value for "__tmpfile" which is based on the "reportfile" name requested by the client. A random component will be added to the name and it is guaranteed not to exist at the time of exec. The "__tmpfile" parameter is passed to the executable command line. The executable command line write its output in that file. When the execution is complete, Jarvis will loop through this file and stream the contents back to the client. If configured, Jarvis will add headers derived from the filename parameter. *This is method appears to work fine under Windows under HTTP.* *Problems have been encountered with this method under Linux.* *Problems have been encountered with this method under Windows with Apache/HTTPS.*
Direct/STDOUT	If no tmpfile parameters are specified, then Jarvis will copy the stdout of the executed program and will echo it via stdout through the server and back to the client. *Problems have been encountered with this method under Windows with binary data.*

Table 24: Exec Return Mechanisms

11.4   A Simple Example

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.

11.5   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.

11.6   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 parameter __dataset is given as the client-supplied dataset given to Jarvis which invoked this exec request.

- The parameter __tmpfile is given if use_tmpfile is configured for this exec.

- 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-agent/demo/echo.test/v1//v2?he=th'is&she=that
```

The command line executed will be:

```
echo __group:admin='1' __dataset='echo.test' __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.

11.7   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.

12   Plugin Datasets

12.1   Introduction

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.

12.2   Configuration

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.
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.

Attribute	Default	Notes
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.
debug	\<n/a\>	Enable "debug" output to standard error for this plugin only. This parameter can not be used to disable global debugging.
dump	\<n/a\>	Enable "dump" output to standard error for this plugin only. This parameter can not be used to disable global dump output.

Table 25: Plugin Configuration Attributes

12.3   Example Plugin

A sample plugin is included with the demo module. By default it is configured as follows:

```
<jarvis>
    <app>
        <plugin dataset="plug" access="*" lib="/usr/share/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, $restArgs, %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";

    # This demonstrates accessing RESTful arguments that are
    # from the URL after the dataset name
```

```
    my $counter = 0;
    map {
        $output .= "RESTful argument $counter: " . $restArgs->[0] . "\n";
    } @{$restArgs};

    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-agent/demo/Plug/rest1/rest2/?myval=test
```

The output is:

```
APP: demo
Username: admin
CGI myval: test
CAT: kiwisaver
RESTful argument 0: rest1
RESTful argument 1: rest2
Boats: 1
```

12.4   Output & Headers

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

12.5   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.";
```

13   Hook Modules

13.1   Introduction

The "hook" configuration entry allows you to define an application-custom module containing methods which will be invoked at key points during the processing.

Hooks are invoked only for the following request types: *Special Datasets (start and finish hooks only), Regular Dataset Fetch (start and finish hooks only), Regular Dataset Insert/Update (all hooks)*. Hooks are never invoked for "plugin" or "exec" datasets.

Hooks can be defined globally and/or for individual datasets. All global hooks are invoked before all dataset-specific hooks.

There are ten supported hook points, which fit into the Jarvis processing sequence as follows:

Hook	Notes
start	The start hook is called when the hook module is first loaded. No dataset analysis has been performed yet. No database transaction is open.
	For global hooks, the start hook is invoked before any login check occurs.
	For per-dataset hooks, the start hook is invoked when the dataset configuration is loaded, before any dataset security checking occurs.
	Args are:
	• $jconfig, • $hook_parameters_href
	The $jconfig is the standard Jarvis configuration object. The hook may modify this object.
	The $hook_parameters_href is a hash containing the name and value from any <parameter> tags configured for this hook in the application XML file.
after_login	This hook is called only when Jarvis successfully creates a new, logged-in session. It allows the hook module to perform additional login processing, e.g. recording first/last login times.
	Also, the hook module may create additional "safe" variables which will be stored in the session and available to all subsequent dataset interactions on this session.
	Note that this hook is called only for global hooks. It is never triggered for per-dataset hooks.
	Args are:
	• $jconfig, • $hook_parameters_href • $additional_safe_href
	The $additional_safe_href is a hash of "safe" parameters which the hook module may extend. All "safe" parameter names must begin with a "__"

	double underscore prefix.

before_all
(insert/update/delete)
	This hook is called after Jarvis has loaded the dataset configuration and processed all REST args. It is called after the transaction begins.
	
	It is called before invoking the <before> SQL statement in the dataset XML file. It is called even if there is no <before> SQL statement.
	
	Args are:
	
	    • $jconfig
	    • $hook_parameters_href
	    • $dsxml
	    • $rest_args_href
	    • $fields_aref
	
	The $dsxml is an XML::Smart object for the dataset's XML config.
	
	The $rest_args_href is the hash of safe variables and REST args to be given to the <before> statement. The hook code may change this hash. Changes will affect only the <before> SQL.
	
	The $fields_aref is the array of user change objects submitted for this request.

before_one
(insert/update/delete)
	This hook is called after Jarvis has assembled all parameters in order to execute a single insert/update/delete statement.
	
	It is invoked before the actual execution of the row modification SQL.
	
	Args are:
	
	    • $jconfig
	    • $hook_parameters_href
	    • $dsxml
	    • $sql_params_href
	
	The $sql_params_href is the hash of safe variables, REST args and general SQL args to be given to the <insert>, <update> or <delete> statement. The hook code may change this hash. Changes will affect only the single SQL statement.

after_one
(insert/update/delete)
	This hook is called after Jarvis has executed a single insert/update/delete statement. It is called after any returning row result has been fetched.
	
	Args are:
	
	    • $jconfig
	    • $hook_parameters_href
	    • $dsxml
	    • $sql_params_href
	    • $row_result_href
	
	The $sql_params_href is the hash of variables previously bound to the single SQL row update. Changing this hash will have no effect.
	
	The $row_result_href is the contents of any returning row. The hook code

	may change this hash.

after_all *(insert/update/delete)*	This hook is called after invoking the <after> SQL statement in the dataset XML file. It is called even if there is no <after> SQL statement. The transaction is not yet committed.
	
	Args are:
	
	    • $jconfig
	    • $hook_parameters_href
	    • $dsxml
	    • $rest_args_href
	    • $fields_aref
	    • $results_aref
	
	The $results_aref is a reference to the @results array of return objects that will be encoded into JSON or XML. Changes to this objects in this array will be represented in the data returned to the client.

return_status	This hook is called only for "__status" requests. It is called just before the status result is encoded to JSON/XML. The hook may do one or more of the following:
	
	   1. Add some extra root level parameters (by modifying $extra_href)
	   2. Perform a custom encoding into text (by setting $return_text)
	
	Args are:
	
	    • $jconfig
	    • $hook_parameters_href
	    • $extra_href
	    • $return_text_ref
	
	The $extra_href is a reference to a hash of name/value mappings all of which will be added as root level parameters to the JSON object or XML response object that is encoded and returned to the client.
	
	The $return_text_ref is a reference that the plugin may set to a non-empty text string if it wishes to override the default JSON/XML encoding that would otherwise be performed by Jarvis. In this case the hook is entirely responsible for encoding the entire JSON/XML result.

dataset_fetched	This hook is called only for "fetch" (i.e. data select) requests. It is called after the SQL select is performed. If the request is for multiple comma-separated datasets then this hook is called once for each dataset in the list.
	
	The hook may do one or more of the following:
	   1. Add some extra scalar parameters (by modifying $extra_href)
	   2. Modify the returned content (by modifying $rows_aref)
	
	Args are:
	
	    • $jconfig
	    • $hook_parameters_href
	    • $dsxml

	• $sql_params_href • $rows_aref • $extra_href The $rows_aref is a reference to the array of objects fetched as a result of the executed SQL. This is the data that will be encoded into JSON or XML. Changes to this objects in this array will be represented in the data returned to the client. Any scalar parameters set by modifying $extra_href will be added to the dataset return structure.
return_fetch	This hook is called only for "fetch" (I.e. data select) requests. It is called at the end of the fetch sequence, just before the returned dataset results are encoded to JSON/XML. The hook may do one or more of the following: 1. Add some extra root level parameters (by modifying $extra_href) 2. Alter the the returned data structure (by modifying $return_object) 3. Perform a custom encoding into text (by setting $return_text_ref) Args are: • $jconfig • $hook_parameters_href • $sql_params_href • $return_object • $extra_href • $return_text_ref The $return_object is a Perl object containing Hash and Array references. The exact structure of this object is different for JSON and XML formats, and also differs according to if the request was for a single dataset or for multiple comma-separated datasets.
return_store	This hook is called only for "store" (I.e. data insert/update/delete) requests. It is called at the end of the stored sequence, just before the results of the store operation(s) are encoded to JSON/XML/CSV. The hook may do one or more of the following: 1. Add some extra root level parameters (by modifying $extra_href) 2. Modify the returned content (by modifying $results_aref) 3. Perform a custom encoding into text (by setting $return_text) Args are: • $jconfig • $hook_parameters_href • $dsxml • $rest_args_href • $fields_aref • $results_aref • $extra_href

	• $return_text_ref
finish	The finish hook is called after the special dataset, fetch or insert/update/delete processing is completed and the response has been sent to the client. Any database transaction associated with the dataset is now finished. The hook module should use this hook only for cleanup and/or auditing. Args are: • $jconfig • $hook_parameters_href

Table 26: List of Hook Points

Note that:

- You may implement none, some, or all hook methods.

- All implemented hook methods must all return "1".

- There is no way for a hook to stop processing, other than to call "die".

- If an SQL error occurs during dataset updates, any following before_one, after_one and after_all hook calls will be skipped. Only the finish hook call will be invoked.

- If multiple <hook> sections are defined in the application XML file, then at the hook point, the corresponding method from all defined hook classes will be called sequentially in the order that they are listed.

13.2   Configuration

The configuration for a "hook" element is as follows:

Attribute	Default	Notes
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'.

| 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 27: Hook Configuration Attributes

13.3   Standard Hook Template

The following empty module should help you get your first Hook module under way.

```
use strict;
use warnings;

# "start" hook.  Not used.
sub hook::Audit::start {
    my ($jconfig, $params_href) = @_;
    return 1;
}

# "finish" hook.  Not used.
sub hook::Audit::finish {
    my ($jconfig, $params_href, $return_text_ref) = @_;
    return 1;
}

# "before_all" hook.  Not used.
sub hook::Audit::before_all {
    my ($jconfig, $params_href, $dsxml, $restful_args_href) = @_;
    return 1;
}

# "after_all" hook.  Not used.
sub hook::Audit::after_all {
    my ($jconfig, $params_href, $dsxml, $restful_args_href) = @_;
    return 1;
}

# "before_one" hook.  Not used.
sub hook::Audit::before_one {
    my ($jconfig, $params_href, $dsxml, $sql_params_href) = @_;
    return 1;
}

# "after_one" hook.  This is the hook we hang our auditing from.
sub hook::Audit::after_one {
    my ($jconfig, $params_href, $dsxml, $sql_params_href, $row_result_href) =
@_;
    return 1;
}

1;
```

14   Performance Notes

14.1   *How Fast is Jarvis?*

The following performance results were measured under the following conditions:

- Dual-Core Intel @ 2.5 GHz, 4G of RAM. (CPU Blowfish rating 7.09).

- Kubuntu 09.10 (Karmic Koala).

- Apache 2 webserver using HTTP (not SSL).

- Apache jmeter client with five simultaneous threads.

- PostgreSQL database.

- Dataset content "<select>SELECT 1 AS result</select>".

Result summary is:

Setup Notes	Latency	Throughput
CGI	523 ms	~ 9 request/sec
mod_perl	54 ms	~ 90 request/sec
mod_perl + Apache::DBI (JSON)	41 ms	~ 120 request/sec
mod_perl + Apache::DBI (XML)	41 ms	~ 120 request/sec
mod_perl + Apache::DBI + Tracking	73 ms	~ 70 request/sec

This shows that the Jarvis framework is easily capable of supporting well over 100 requests per second on commodity hardware. This is suitable for many mid-range RIA applications.

Naturally, any heavy database processing will reduce this response rate accordingly.

15 Tracking Application

15.1 Tracking Requests & Errors

Included in the Jarvis distribution is a mechanism for tracking user requests and associated errors. This is configured in the configuration file e.g. as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<jarvis>
    <app format="json" debug="yes" dump="yes">
        <tracker logins="yes" requests="yes" errors="yes">
            <dbfile>/var/lib/jarvis/tracker/tracker.db</dbfile>
        </tracker>
```

The parameters are.

Attribute	Default	Notes
logins	No	Should Jarvis track logins in the SQLite tracking database?
requests	No	Should Jarvis track requests in the SQLite tracking database?
errors	No	Should Jarvis track errors in its SQLite tracking database?
dbfile	See Note	Location of the tracker SQLite database. This is required iff one or more of "requests" or "errors" is configured to be enabled. The default location is: /var/lib/jarvis/tracker/tracker.db

Table 28: Tracker Configuration Parameters

15.2 The Tracker Database File

The tracker database file must be manually placed into a location where the user executing the CGI script can access and write to the file. Also, note that the CGI user must also have write access to the file's parent directory in order to create locking files. Under Apache with Ubuntu, this is the www-data user.

Any empty tracker database file is provided for this purpose. The file is named "tracker-empty.db" and is installed into the "jarvis/tracker" folder of your installation.

To use this in production you must:

- Take a copy of "tracker-empty.db".
- Place this copy in a folder where it can be written by the web server.
- Update the "tracker.xml" file so that it correctly references the new location.
- Update the application configuration files of any Jarvis app wishing to record tracker data.

15.3 The Tracker Display Application

When Jarvis is used, each request and/or error will cause a record to be written into the database tables in tracker.db. You can access these tables for your own purposes. Use your favorite SQLite

utility to view the schema and content of the tables "request" and "error".

Alternatively, use the "tracker" application provided with Jarvis. This is a application written using ExtJS. It is installed by default, but for security purposes it must be configured before use.

15.3.1　　Enabling the Display Application

Enabling the tracker display application requires both Apache configuration to provide access to the HTTP documents, and Jarvis configuration for the application to access the tracker database.

The tracker display application requires a number of supporting projects to be installed:

- ExtJS 2.3. Please see below for further details on this.

- SHJS. Syntax Highlighting for Javascript. Please see below for further details on this.

- Protovis. For graphs. Please see below for further details on this.

Once these three pre-requisites are installed, the tracker display application can be configured and will be then available. To configure the tracker application itself, first, enable the display application in Apache: The following Apache configuration is sufficient:

For Debian:

```
Alias /jarvis-tracker/ "/usr/share/jarvis/tracker/htdocs/"
```

For Other Systems:

```
Alias /jarvis-tracker/ "/opt/jarvis/tracker/htdocs/"
```

Configure Jarvis to recognise the tracker. Copy the tracker's example Jarvis application configuration file from "tracker/etc/tracker.xml" to the Jarvis etc directory:

For Debian:

```
cp /usr/share/jarvis/tracker/etc/tracker.xml /etc/jarvis/
```

For Other Systems:

```
cp /opt/jarvis/tracker/etc/tracker.xml /opt/jarvis/etc/
```

Edit the configuration file to ensure all the paths and other configuration are correct. Note that this example file comes with configuration to log Jarvis requests and errors into the tracker database.

Reload Apache's configuration, then point your browser at the following:

```
http://localhost/jarvis-tracker/
```

15.3.2　　Installing Tracker Display Application Support Files

The front-end for the demo application is written in JavaScript using the ExtJS and also utilises a number of other libraries for various features of the display application.

ExtJS

Please download the 2.3 release of ExtJS from http://www.sencha.com/products/js/download.php

and install. The file "ext-all.js" should be accessible by the URL "/ext-2.3/ext-all.js". E.g the Apache configuration may be:

```
Alias /ext-2.3/ "/opt/extjs-2.3/"
```

PROTOVIS

To make the Protovis library available on your server, simply download the SVGWeb enabled Protovis 3.2 or later from http://gitorious.org/~jlove/protovis/jloves-protovis and unpack it on your machine. Then configure Apache to serve the protovis files through the alias "/protovis-3.2".

For example, if you have unpacked Protovis into "/opt/", then the path to protovis-d3.2.js (the release of Protovis at the time of writing – this may have changed) should be "/opt/protovis-3.2/protovis-d3.2.js". The following Apache configuration is sufficient.

```
Alias /protovis-3.2/ "/opt/protovis-3.2/"
```

Note that this 3.2 release of Protovis does support IE.

SHJS

Similarly for SHJS, download the latest release of SHJS from http://shjs.sourceforge.net/ and install. For example the Apache configuration may be:

```
Alias /shjs/ "/opt/shjs-0.6/"
```

SIMILE

Similarly for SIMILE, download the 2.3 SIMILE "timeline" library release from http://code.google.com/p/simile-widgets/downloads/list and install. For example the Apache configuration may be:

```
Alias /timeline-2.3/ "/opt/timeline_2.3.0/"
```

The home page for the SIMILE widget is http://www.simile-widgets.org/timeline/.

15.3.3   Rebuilding Support Tables

Included in the tracker database are a number of tables that are used by the display application for generating data for showing in the browser. These tables are the "calendar" and "day_interval" tables.

Both these tables must have their data generated. The "tracker-empty.db" file comes pre-loaded with these tables full of generated data. If you need to regenerate the data for these tables, the script "populate-generated-tables.pl" that can be found in the "tracker/db" directory can do this for you.

To use, you must first ensure that the Perl module Date::Calc is installed. Under some systems with apt-get, you can perform:

```
sudo apt-get install libdate-calc-perl
```
to do this for you.

Once this is installed, you can run the script directly:

```
perl populate-generated-tables.pl
```

Note that this script attempts to load the database file "tracker.db" in the directory the script is run from. If you need to update a different database, you will need to edit the script and alter the database the DBI handler loads.

If you have data that is not within the range Jan 1st 2008 to Jan 1st 2020, then you will need to edit the script further and alter the date range the 'calendar' table is generated for.

16 Demo Application

16.1 Introduction

A simple demonstration application is provided with Jarvis. This application uses an SQLite database. When installing the Debian packages, this demonstration is automatically configured and enabled and available under the address:

```
http://localhost/jarvis-demo/
```

The following instructions are required only when not using the Debian package of Jarvis.

16.2 Complete Basic Jarvis Configuration

First, ensure that Jarvis itself is available via the /jarvis-agent/ URL. Access the following URL:

```
http://localhost/jarvis-agent/
```

Verify that you receive the response...

```
Missing app name.  Send /jarvis-agent/<app>[/<dataset>[/<arg1>...]] in URI!
```

16.3 The Demo Database File

Take a copy of the database to a directory writable by the Apache user. For example:

```
# mkdir -p /var/lib/jarvis/demo/
# cp /opt/jarvis/demo/db/demo-empty .db /var/lib/jarvis/demo/demo.db
# chown www-data:www-data /var/lib/jarvis/demo/demo.db
```

If this is not done correctly, you will see the error "Attempt to write to a read-only database" when saving changes via the demo screen.

16.4 Configure Demo Web-Service

Install the demo.xml Jarvis application configuration file:

```
# cp /opt/jarvis/demo/etc/demo.xml /opt/jarvis/etc/demo.xml
```

In Apache configuration create an alias to the Jarvis demo htdocs:

```
Alias /jarvis-demo/ "/opt/jarvis/demo/htdocs/"
```

Change the default values configured in demo.xml to match your installation:

- Location of the demo database file. Default is /var/lib/jarvis/demo/demo.db.
- Location of the datasets configuration. Default is /usr/share/jarvis/demo/dataset.
- Location of the session files. Default is /var/lib/jarvis/demo/sessions.
- Location of the plugin file. Default is /usr/share/jarvis/demo.