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 and MS Excel 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 v4.0.0):

Version	Date	By	Comment
5.4.6	18-Mar-2014	JXC	Added “!out” flag to support DBD::Oracle RETURNING.
5.4.10	7-Apr-2014	JXC	Cleanup/correct documentation for Exec/Plug config parameters.
5.5.0	7-Apr-2014	JXC	Add “json.rest” format. Remove “xml.array”. Add “router”.
5.5.1	8-Apr-2014	JXC	Added support for presentation=“singleton” on route.
5.6.0	8-Apr-2014	JXC	Add “dataset_pre_fetch” hook. Plugin rest args is now HASH. Removed support for comma-separated datasets. Cleanup on hook parameters. Clear separation of global vs per-dataset hooks.
5.7.0	10-Apr-2014	JXC	Removed tracker. Added nested dataset fetching.
5.7.1	11-Apr-2014	JXC	Removed $extra_href from dataset_fetched hook. Added doc for Jarvis::Dataset::fetch_rows ( ). Added docs for nested fetch.
5.7.2	14-Apr-2014	JXC	Added “dataset_pre_store” and “dataset_stored” hooks. Changed parameters on “return_store” hook.
5.7.3	14-Apr-2014	JXC	Made “$extra_href” available to “dataset_fetched” and “dataset_stored” hooks.
5.7.4	15-Apr-2014	JXC	Added “$extra_href” to documentation for “fetch_rows” method.
6.0.0	15-Apr-2014	JXC	Added nested insert/update.
6.0.3	12-May-2014	JXC	Added “dataset_type” officially to $jconfig.
6.1.0	11-Nov-2014	JXC	Now “after” statement and “after_all” hook are inside transaction.
6.3.7	29-Mar-2018	JL	Documented new error logging features

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 can cache database connections under mod_perl.

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 applications, I hope you will find that Jarvis does everything you need to do, in a simple and helpful fashion.

2 Licence

2.1 GNU Lesser General Public License

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

2.2 Clarification of Intentions

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

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

3 Installation

3.1 Separate Documentation

Please see the separate “Jarvis Install” document. That covers:

- Installation on Linux, Solaris and Windows.
- Testing your Jarvis installation.
- Testing the “Demo” 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>/usr/share/jarvis/demo/dataset</dataset_dir>

        <router>
            <route path="/boat_class/:boat_class" dataset="boat_class"/>
            <route path="/boat/by-class/:boat_class" dataset="boat"/>
            <route path="/boat/:id" dataset="boat" restful="element"/>
        </router>

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

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

        <login module="Jarvis::Login::Database">
            <parameter name="user_table" value="staff"/>
            <parameter name="user_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	Default format for returned content. Valid configuration options are: “json”, “json.array”, “json.rest”, “xml”, “csv”, or “xlsx”. Note that the content of client POST requests is also either JSON or XML, but does not need to match the returned format.
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.

Attribute	Default	Notes
log_format	See Notes	Specifies the format template to use for debug, dump and log output. Default is “[%P/%A/%U/%D][%R] %M”. See following log_format section for description of available fields.
error_response_format	See Notes	Specifies the format template to use for error responses to the client. Default is "[%T][%R] %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).
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>	Defines a directory to contain dataset definition files. Multiple dataset_dir entries may be defined.
router	<none>	Router configuration allows the use of named RESTful parameters with the URL. It also gives explicit control over URL pattern mapping to datasets (including plugin and execs).
default_libs	<none>	This element allows a list of zero or more directories to be defined which will then be added to the @INC path when loading plugin modules to execute.
default_parameters	<none>	Defines values which are supplied as default user parameters on all requests.
login	<n/a>	This sub-element defines and configures the login mechanism to be used for this application. At most one instance of this should be defined. If none is defined, users will never be logged in, and datasets must use “**” as their access group parameter.
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”.

Attribute	Default	Notes
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

Default encoding format from response body returned by Jarvis to the client.

Specify “json”, “json.array”, “json.rest”, “xml”, “csv”, or “xlsx”.

- This may be over-ridden a per-request basis with the CGI parameter “format”.
- This parameter is entirely separate from the encoding of the submitted POST data in the request, which is specified via the content-type header in the HTTP request.

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.
%H	High Resolution Timestamp providing time up to microseconds, e.g. “Mon Jul 06 09:41:33.072831 2009”. Accuracy is restricted by system clock.
%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.
%S	The session ID of the current user’s session. All requests will have a session.
%R	The request ID of the HTTP request. This is a UUID, unique to the HTTP request.
%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 differ from the databases ORDER BY sequencing.

Note that if sort_field_param is present but sort_dir_param is not, then sort order is 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 defines the location of the XML datafiles which comprise the application's datasets.

More than one dataset directory may be specified, with a unique combination of:

- Dataset Type
- Dataset Prefix

At least one dataset directory must be defined, unless your application really has no datasets. I.e. unless your application consists entirely of exec and plugin actions.

If multiple “dataset_dir” entries are defined, they must each have a unique “prefix”.

Attribute	Default	Notes
prefix	<empty-string>	This is the prefix for incoming dataset names which determine which dataset_dir entry is used for locating the matching datasets. If omitted/empty then all dataset names will match. A trailing “.” is automatically added to the prefix. If an incoming dataset name matches multiple dataset_dirs the longest-matching prefix is used.

Attribute	Default	Notes
type	dbi	Specify the type of datasets contained within this directory. • "dbi" (SQL databases using default DBI driver). • "sdp" (SSAS datapump via SOAP requests). This is used to locate a suitable database definition for executing the dataset.
dbname	default	Specifies the default database name to use for datasets in this directory. Can be overridden on a per-dataset basis.
[content]	<n/a>	The content inside the <dataset_dir> element is the path to the directory within the server file system.

Table 3: Dataset Dir Attributes

Note the following:

- The client-supplied dataset name must not include the ".xml" suffix added by the server.
- The matched prefix is stripped from the dataset name before expansion.
- 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 configuration:

```
<dataset_dir>/path/to/dataset</dataset_dir>
<dataset_dir prefix="other">/another/path</dataset_dir>
```

The final dataset location would be as follows:

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
other.set.name	**/another/path**/set/name.xml

Table 4: Examples of Resolving Dataset Names

4.3.8 Configuration: router

The "router" section is optional. Many applications will not need one. However, it can be used to assign names to RESTful parameters (which otherwise would only be available via position index).

The "router" element contains zero or more "route" entries. Each "route" entry has the following

attributes.

Attribute	Default	Notes
path	<none>	This is the path pattern to match. It begins with a leading "/". The route path is compared against the supplied inbound request path. Note that the "app-name" and any script location is removed from the inbound request path before matching is applied. Each component in the path must be either: • An empty part (matches any inbound part). • An asterix "*" (matches any inbound party). • An exact string to match. • A colon-prefixed ":<var>" variable to extract.
dataset	<none>	Specify the name of the dataset to be executed if this path pattern is matched. This may be a standard XML-file defined dataset within "dataset_dir", or it may be the name of a virtual dataset implemented by a "plugin" or "exec" definition.
presentation	"array"	Specifies if the returned data will be given back as an "array" (the default), or as an "singleton". If "singleton" is specified then: • The query must return exactly 1 object. • The object is returned as top level data, with no wrapper. • Jarvis will return 404 if the query returns no rows. • Jarvis will "die" if the query returns > 1 row. Note: Singleton formatting applies only to "json" and "json.rest" return formats. Specifying "singleton" will not affect the return encoding for other formats, although the row count validation will still be performed.

Table 5: Route Attributes

For example, consider the configuration.

```
        <router>
            <route path="/boat_class/:boat_class" dataset="boat_class"/>
            <route path="/*/by-class/:boat_class" dataset="boat"/>
            <route path="/boat/:id" dataset="boat" presentation="singleton"/>
        </router>
```

Now consider the following request:

```
http://localhost/jarvis-agent/demo/boat/by-class/X%20Class
```

The processing is as follows:

1. The leading script location: "http://localhost/jarvis-agent" is removed.

2. The application name “/demo” is removed.
3. The supplied path “boat/by-class/X%20Class” is matched against the routes in order.

The first route compares supplied “boat” against an exact string “boat_class” and the match fails.

The second route compares as follows:

a) Part one compares “boat” against the wildcard “*” and passes.

b) Part two compares “by-class” against exact “by-class” and passes.

c) Part three assigns “X Class” to the variable “boat_class”.

d) The dataset “boat” is selected.

The final result is:

- The dataset is “boat”.
- The parameter {$1} in a dataset XML will expand to “boat”.
- The parameter {$2} in a dataset XML will expand to “by-class”.
- The parameter {$3} in a dataset XML will expand to “X Class”.
- The parameter {$boat_class} in a dataset XML will expand to “X Class”.

Note however that when performing “insert/update/delete” requests, any per-row value for {$boat_class} will override the URL-supplied value for {$boat_class}.

4.3.9 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.

4.3.10 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.

4.3.11 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.

4.3.12 Configuration: database

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

Attribute	Default	Notes
type	dbi	Either “dbi” or “sdp”.
name	default	You may have one default “dbi” database and one default “sdp” database. The correct one will be selected to match the type parameter on the “dataset_dir” definition. For a simple database connection, the default name of “default” is fine. If your application wishes to use more than one database instance of a type, use this field to distinguish them. Within each dataset you may specify the database name to be used for executing that dataset.
connect	dbi:Pg:<app-name>	For “dbi” databases, this is a DBI connection string. e.g. *dbi:Sybase:server=10.42.2.8;database=Demo* For “sdp” databases, this is a SOAP proxy address, e.g.: *http://10.42.2.8/ssas_datapump/msmdpump.dll*
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.
prepare	<empty-string>	Optional parameters passed on to DBI::prepare. Example: <database connect=“...” prepare=“pg_server_prepare => 0”/>
post_connect	<empty-string>	For “dbi” databases, an SQL command that is executed immediately after a connection has been established. E.g. <database connect=“...” post_connect=“SET ANSI_WARNINGS, ANSI_PADDING, ANSI_NULLS, ARITHABORT, QUOTED_IDENTIFIER, CONCAT_NULL_YIELDS_NULL ON;”/>

Table 6: Database Configuration Attributes

For SOAP, the username and password are provided as “basic authentication” credentials.

The configuration of an HTTP SOAP Data Pump service is outside the scope of this document.

In addition, the database configuration supports additional contained “dbh_attributes” tags which are passed to the DBI->connect() method when logging into the database. For example:

```
        <database connect="dbi:Sybase:server=192.168.1.5;database=mydb"
username="sa" password="sa">
            <dbh_attributes>
                <attribute name="syb_chained_txn" value="0"/>
            </dbh_attributes>

            <post_connect>
                SET ANSI_WARNINGS, ANSI_PADDING, ANSI_NULLS, ARITHABORT,
QUOTED_IDENTIFIER, CONCAT_NULL_YIELDS_NULL ON;
            </post_connect>
        </database>
```

Each dbh_attributes sub-element named “attribute” has a “name” and “value” attribute. These parameters are passed directly to DBH->connect() method, in the fourth parameter.

Note when connecting to MS SQL Server on a Linux platform via the DBD::Sybase driver, using DBD::Sybase versions later than v1.00, the use of the DBH attribute “syb_chained_txn” is required, lest the following error occur:

```
The COMMIT TRANSACTION request has no corresponding BEGIN TRANSACTION. Server
message number=3902 severity=16 state=1 line=2
```

4.3.13 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 7: 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 8: 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 9: Advantages/Disadvantages of Jarvis <sessiondb>

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

4.3.15 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.

4.3.16 Configuration: habitat

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

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

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

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.
require_post	no	If true, prevent usage of URL parameters username/password for login purposes.

Setting require_post to true is recommended to avoid logging of usename/password as part of the request URL.

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.
no_password	0	If set true, the module will skip the password checking.
allowed_groups	"	Comma-separated list of groups. If specified, restricts login to users with group in allowed groups. Wildcard * is supported.

Table 10: 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 11: 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”.

Attribute	Default	Notes
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.
encryption	none	Specifies the encyption 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.
password_cost_update_statement	<none>	SQL statement to update stored user passwords. Optional. This will be executed transparently upon login if the following conditions are met: • eksblowfish encryption method is used. • stored user password hash is weaker than the current strength level. e.g. “UPDATE users SET password = ? where username = ?”

Table 12: 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. Wildcard * is supported.
allowed_groups	"	Comma-separated list of groups. If specified, restricts login to users with group in allowed groups. Checked after filtering for relevant groups. Wildcard * is supported.

Table 13: 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 14: 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 15: 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.

Attribute	Default	Notes
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 16: Single Login Module Parameters

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

5.9 Executable Login

This module performs username and password authentication by using an executable file.
The executable file is expected to return successfully and output JSON to STDOUT.

Example:

```
<jarvis>
    <app>
        <login module="Jarvis::Login::Executable">
            <parameter name="executable" value="/executable_file_path"/>
        </login>
```

The parameters are.

Attribute	Default	Notes
executable	<none>	Absolute path to an executable file for authentication. It will be executed with the username and password as parameters.
allowed_groups	'*'	Comma-separated list of groups. If specified, restricts login to users with group in allowed groups. Checked after filtering for relevant groups. Wildcard * is supported.

Table 17: Executable Login Module Parameters

All spaces and special characters are stripped from the allowed group name.

5.9.1 Executable result.

The result from the executable is expected to provide the following:

Attribute	Default	Notes
success	<none>	A 1 if successful login 0 or not specified if failed.
groups	<none>	Comma-separated list of groups. Or an Array of groups that this login/session belongs to.
message	<none>	A string to explain the reason why login failed.
additional	<none>	An object defining any amount of key=value to assign to the session. Key names for safe parameters must start with __ and will be put into $jconfig->{additional_safe}.
cookies	<none>	An object of key=values to be turned into cookies on success login. <Optional> Any cookies defined in this hash will be sent to the client in a cookie string.

Table 18: Executable Login Module Parameters

Note that when this module is used to login, any dataset requests will also have access to any safe parameters returned by the result in the additional object.

These variables can be accessed in a dataset e.g. by specifying {$__variable_name}. 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.

Note it is up to the developer of the executable to ensure the key names are suitable for cookie key names.

An example of success is:

```
{
  "success": 1,
  "groups": ["admin"],
  "additional": { "__user_id": 1 },
  "cookies": { "PHPSESID": "…" }
}
```

An example of fail is:

```
{
  "success": 0,
  "message": "Banned until 2017/01/01."
}
```

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, with a response body which gives a description of the problem.

- 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.
- Invalid value for select parameter.

Store Errors

Store errors are errors related to the actual user-supplied data values being rejected by the database. These occur when Jarvis cannot insert/update/delete data due to primary key, foreign key, unique constraints or bad type.

In such cases, a “200 Success” result is returned, but the “success” attribute in the returned JSON or XML will indicate the failure, and a “message” parameter will indicate the reasons.

7 Fetch Results

7.1 Format = json

Consider the Demo application supplied with Jarvis. This application uses the “json” format. The following chapter will describe the definition of datasets and the use of parameters. For now we consider only the structure of the returned content.

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

The JSON format response for a data fetch is wrapped in an outer object.

- The returned tuples are contained in an array with object key “data”.
- Each tuple is represented as an object. Missing fields are omitted.

Response is:

```
{
   "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"
}
```

Top level attributes are:

Attribute	Notes
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 19: Dataset Fetch Top-Level Attributes

7.2 Format = json.array

The JSON Array format response for a data fetch is wrapped in an outer object.

- The returned tuples are contained in an array with object key “data”.
- Each tuple is represented as an array.
- The column names are given separately
- The order of the array elements for each tuple matches the given column order.
- Empty elements in the array are given as as JSON null value.

Consider the following request:

```
http://localhost/jarvis-agent/demo/boat_class?format=json.array
```

Response is:

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

7.3 Format = json.rest

The JSON Rest format is a simplified version of the default “json” format which returns purely the data component. This is suitable for pure REST-ful frameworks which do not expect any metadata.

- The “data” value is returned a top-level array.
- All of the other attributes are discard.

Consider the following request:

```
http://localhost/jarvis-agent/demo/boat_class?format=json.rest
```

Response is:

```
"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."
   }
]
```

7.4 Format = xml

The XML format response for a data fetch is wrapped in an outer object.

- The returned tuples are contained in a “data” element.
- Each tuple is represented as a “row” element.
- Fields are specified as attributes of the “row” element.

Consider the following 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>
```

7.5 Format = csv

The CSV format response for a data fetch is “standard” CSV encoding with a header row.

- The header row gives column names.
- Fields are separated by commas.
- Strings containing spaces or special characters are in double quotes.

- Multi-line strings are supported, a newline character is used.

Consider the following request:

```
http://localhost/jarvis-agent/demo/boat_class?format=csv
```

The returned content in CSV format is:

```
active,class,description,id
Y,Makkleson,"Suitable for infants and those of timid heart.",1
N,"X Class","Product of a deranged mind.",2
```

7.6 Format = xlsx

The response is identical in structure to the CSV format. However, it is encoded in XLSX format, used by Microsoft Excel 2000 and later.

8 Datasets and Parameters

8.1 Dataset Definition

The dataset is defined in a .XML file containing up to four SQL statements. The “boat_class.xml” file used for the preceding examples is defined as follows.

```
<dataset read="*" write="*">
    <transform fetch="notnull" store="trim,null" />
    <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 = datetime ('now')
WHERE id = {$id};
    </update>
    <insert returning="yes">
INSERT INTO boat_class (class, active, description, change_user, change_date)
VALUES ({$class}, {$active}, {$description}, {$__username}, datetime ('now'));
    </insert>
    <delete>
DELETE FROM boat_class
WHERE id = {$id};
    </delete>
</dataset>
```

Note the use of the <transform> element to request that data is pre-processed before fetch and save. The transform tag is described elsewhere. However, in this case we have requested some typical options:

- On fetch, any NULL value in the database is represented as a zero length string.
- On store, leading and trailing white space should be removed before saving to database.
- On store, empty strings from the client should be stored as NULL in the database.

This example is a basic dataset definition for a CRUD application, with “fetch” (i.e. select) requests and “store” (i.e. insert, update, delete) requests.

Additional supported features in datasets include:

- <hook> call to a Perl module for additional processing on each request.
- <before> execute an SQL statement once before any store requests are performed.
- <after> execute an SQL statement once after any store requests are performed.

These are described in more detail subsequently.

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

Attribute	Default	Notes
dbname	default	Specify which database connection should be used for fetching and storing this dataset. By default, the database named “default” is used for executing the set.
read	<empty>	Defines which groups may execute the SQL defined in the “select” element. This is an access identifer as per the following table.
write	<empty>	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	no	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	no	Enables dump on a per-dataset basis.
filename_parameter	filename	Specifies the user-supplied parameter which specifies the filename for the returned attachment when requesting contents to be returned in “csv” for “xlsx” format.

Table 20: Attributes of the <dataset> Element

The elements within the <dataset> definition for DBI (standard SQL) datasets 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.
child	One or more nested child dataset definitions. See chapter on Nested Datasets.
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 21: Dataset Sub-Elements

The select, update, insert and delete sub-elements support the following sub-parameters:

Sub-element	Notes
prepare	Optional parameters for statement preparation with DBI::prepare. <update prepare=“pg_server_prepare => 0, something_else => 1”> Overwrites global database prepare attributes.
returning	For “insert” statement only. Discussed later under the “insert” statement section.

Sub-element	Notes
nolog	Suppress logging and tracking of errors matching given pattern. Empty string matches nothing, otherwise interpreted as perl pattern.
ignore	Suppress errors/warnings matching given pattern. Empty string matches nothing, otherwise interpreted as perl pattern.

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 22: Access Control Specifiers

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

8.2 Dataset Bind Parameters

In nearly all cases, the dataset statements to execute will depend on client-side parameters. In the general case for SQL queries this is performed by using SQL bind parameters, as follows:

The SQL statements in the dataset query definition may contain bind substitution parameters defined in curly braces with a preceding dollar symbol. e.g. {$parameter}.

Note: The use of double braces (and/or with missing dollar symbol) is very deprecated and will be removed at some time in the very near future. E.g. {name}, {{name}}, or {{$name}} will no longer be supported.

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, the parameter may be followed by one or more bind flags which modify the bind behavior. The supported bind flags are.

Flag	Notes
!out	Bind as an in/out variable (instead of “in”).
!varchar	Bind as SQL_VARCHAR (this is the default). Used only when at least one parameter is bound as in/out.
!numeric	Bind as SQL_NUMERIC. Used only when at least one parameter is bound as in/out.
!quote	Used only for textual substitution binding. Always use DBI->quote.
!noquote	Used only for textual substitution binding. Do not quote. Instead remove all space and special characters.
!raw	Used only for textual substitution binding. Do not quote. Use exactly as specified. Use with extreme caution.

Table 23: Jarvis Bind Flags

8.3 Binding In/Out Parameters (under DBD::Oracle)

By default, all variables are bound as “in” parameters. When data is returned under a “returning” clause, this typically occurs automatically using a post-insert fetch result. This default behavior works fine for:

- Microsoft SQL Server
- Postgres

However, for OracleDB, you must use in/out parameter binding in order to get data returned. For example, after inserting a row you wish to return the auto-generated sequential row ID. Under OracleDB you must use in/out parameters for this purpose.

You can request Jarvis to use an in/out bind by specifying the “**!out**” flag on the variable.

```
<insert returning="no">
  insert into MY_TABLE (CUSTOMER, NAME, DESCRIPTION)
    values ({$customer_id}, {$name}, {$description})
    returning ID into {$id!out}
</insert>
```

When Jarvis detects that an in/out binding has been requested, it will also do the following:

- Use “in/out” binding for all parameters flagged as “**!out**” in the statement.
- Use “in” binding for all other parameters in the statement.

Note that Jarvis will use SQL_VARCHAR for all parameter binding, unless you request otherwise with a “**!numeric**” flag.

The output values will be placed into a row inside the “returning” element of the response given back to the client, in a manner that is structurally identical to the “returning” elements given by the default “returning” mechanism.

8.4 Fallback Parameters

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 if no other value is found.

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 in this context:

```
http://localhost/jarvis-agent/demo/boat_filter/a
http://localhost/jarvis-agent/demo/boat_filter?class_name=a
```

The first is an indexed RESTful parameter. It is the first indexed 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 indexed 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 the indexed RESTful parameters “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 “&”.

Note: This example shows the use of indexed RESTful parameters. To use named RESTful parameters, you must configure a “route” with variable parts. See the “router” configuration section for a worked example using named RESTful parameters.

Note: 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.

Note that any bind flags must be placed at the end of the fallback parameter list.

8.5 Other Parameter Notes

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 is not valid in this case:

```
<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. In such cases you may wish to use textual substitution parameters.

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}
```

8.6 Safe 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 indexed 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 default safe 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 24: Jarvis Secure Variables

In addition, safe parameters may come from the following other sources:

- Login modules may define additional safe parameters, e.g. {$__user_id}.
- Hooks may define additional safe parameters.

- Default Parameters defined in the application config file may be safe parameters.

Remember. Client-supplied values will never be allowed to modify these safe parameters.

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>
```

8.7 Textual Substitution

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. This is guaranteed safe against any form of SQL injection.

However, there are situations where you want to use textual substitution. This is where a parameter is inserted as text into the statement before 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 provides a built-in post-processing paging and sorting function. There are two reasons to consider using textual parameter substitution instead of the built-in mechanism.

- To sort by two or more columns. The built-in mechanism supports only one.
- To perform paging within the database query, for performance reasons.

The parameter values available for textual substitution are identical to those available for DBI bind parameters. However, the need to guard against SQL injection leads to special considerations:

1. By default, the following quoting rules apply:
    - If the parameter value is an integer or floating point, it is unquoted. Exponential numbers will be recognized as numbers.
    - Otherwise the parameter is quoted using the DBI database handle $dbh->quote(...) method appropriate for this database. According to the DBI documentation, this is safe from SQL injection.
2. Alternatively, if the "**!quote**" flag is specified, e.g. [$stringvar!quote] then the value is always quoted with the DBI quote function.
3. Alternatively, if the "**!noquote**" flag is specified, e.g. [$sortorder!noquote] then the value is never quoted. Instead, all characters except the following are deleted: 0-9, a-z, A-Z, *space*, *underscore*, *hyphen*, *comma*.
4. Alternatively, if the “**!raw**” flag is specified, e.g. [$__safevar!raw] then the value is textually substituted into the SQL with no checks or restrictions. Note that the “**!raw**” can only be used with “safe” substitutions variables, i.e. those that begin with a double underscore. Safe variables can never be supplied by the client. The “**!raw**” flag is restricted to use with variables that are provided by Jarvis itself, or by server-side hooks and plugins.

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.

9 SSAS Data Pump Datasets

9.1 MDX Queries

The entire preceding discussion is accurate for the use of DBI datasets with SQL statements. For SSAS Data Pump datasets using MDX requests over SOAP, there are some key differences.

The top level attributes are very similar.

Attribute	Notes
dbname	Specify which database connection should be used for fetching and storing this dataset. By default, the database named “default” (of type “sdp”) is used for executing the set. Note that the type of dataset (“dbi” or “sdp”) is not configurable on a per-dataset basis. It is determined by the <dataset_dir> element defining the directory containing the dataset. All datasets in the same directory must execute against databases of the same type.
read	Identical to DBI datasets. Access control identifiers are also identical.
write	Identical to DBI datasets. Access control identifiers are also identical.
debug	Identical to DBI datasets.
dump	Identical to DBI datasets.

Table 25: Attributes of the <dataset> Element

The elements within the <dataset> definition for SSAS Data Pump datasets are as follows:

Element	Notes
transform	Identical to DBI datasets.
hook	Identical to DBI datasets.
mdx	MDX to execute when dataset is invoked with the http GET request method.

Table 26: Dataset Sub-Elements

Here is an example SDP dataset which includes a parameter:

```
<dataset read="**" write="">
    <mdx row_label="GL Code">
SELECT
  [Dw Target Planning].[Category].Children ON ROWS,
  [Time Target Planning].[Month].Members ON COLUMNS
FROM [Cube Name]
WHERE ([Time Target Planning].[Year].[Calendar {$year}])
    </mdx>
</dataset>
```

9.2 Parameter Processing

Parameter sources are identical to the DBI case:

- Client Parameters (REST, Query, POST parameters).
- Server Parameters (Safe parameters, Default parameters).
- Parameter Fallback.

However, since the SOAP mechanism does not support binding, the are some key differences in the parameter handling:

- All SDP parameter substitution is textual substitution. Always.
- The {$parameter} or [$parameter] forms for MDX query parameters are identical.
- The supported formatting flags are “!string”, “!bracket” and “!raw”.

The “!bracket” flag is intended for use when the MDX parameter is to be expanded within the context of MDX brackets. If the “!bracket” flag is specified, then the following rules apply:

- Any client-supplied “]” character is replaced with “]]”.

The “!string” flag is intended for use when the MDX parameter is to be expanded within the context of double quotes:

- Any client-supplied “\” character is replaced with “\\”.
- Any client-supplied *double-quote* is replaced with *backslash double-quote*.

The “!raw” flag is permitted only for safe variables, i.e. those whose names begin with underscore, underscore. Variables with these names can never be provided by the client. They are only permitted from Jarvis itself, including hooks and plugins. The “!raw” flag means that the variable is expanded verbatim with no quotes or modification.

If no flag is specified, then the value is unquoted. However, all characters except the following are deleted: 0-9, a-z, A-Z, *space*, *underscore*, *hyphen*, *comma*.

10 Storing Datasets

10.1 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 27: Top-Level Attributes of Array Modification Request

10.2 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...
<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 28: Attributes of the “row” Sub-Elements in Array Modification Response

10.3 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.

10.4 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:

1. Secure Dataset Parameters (such as “{__username}” and “{__group_list}”.
2. Indexed RESTful arguments as specified in the URL e.g. “{1}” or “{2}”.
3. Named RESTful arguments assigned by a “route” rule (see the “router” configuration).

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.

10.5 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.

Note that SSAS Data Pump datasets support only the “fetch” mechanism. There is no writeback.

11 Dataset Transformations

11.1 Introduction

Refer back to the original “boat_class.xml” example given earlier in this document. It used Jarvis “transform” configurations to ensure that empty strings were always written as NULL to the database, and that NULL values in the database were always given as empty string to the client.

This allows the client to be confident that empty values were always represented in a consistent manner. The alternative is to explicitly code this into the SQL statements in the dataset, e.g. with NULLIF, COALESCE, BTRIM, etc.

The <transform> element allows you configure transforms for “store” and “fetch”. The attribute value is a comma-separated list of transformations to apply when fetching (select) and when storing (insert, update, delete).

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 29: Transformation keywords for “store” and “fetch”

12 Nested Datasets

12.1 Introduction & Fetching Nested Datasets

Nested datasets allow a single query to return an “object-style” fetch result, where the returned result can include child elements.

Consider the following JSON result returned from a nested dataset example included in the demo Jarvis application. The “parts” entry is an array generated by performing a nested child dataset query within the top-level dataset.


```
{
   "data" : [
      {
         "owner" : "",
         "registration_num" : "",
         "parts" : [
            {
               "name" : "Gadget",
               "id" : 15
            },
            {
               "name" : "Sprocket",
               "id" : 14
            },
            {
               "name" : "Widget",
               "id" : 13
            }
         ],
         "name" : "Empty Nest",
         "id" : 52,
         "class" : "X Class",
         "description" : ""
      }
   ],
   "logged_in" : 1,
   "group_list" : "admin,default",
   "username" : "admin",
   "returned" : 1,
   "fetched" : 1,
   "error_string" : ""
}
```


12.2 Configuration

Child datasets are attached by defining a “child” attribute in the parent datasets definition.

Consider the following example:

```
<dataset read="*" write="*" key_list="id">
    <child field="parts" dataset="boat_part">
        <link parent="name" child="boat_name"/>
    </child>
    ...
```

The “child” element attributes are:

Attribute	Notes
field	The name of the parent row field to contain the child rows.
dataset	The name of the dataset which will generated the nested child sub-query data. This must be an XML-defined dataset. Exec or Plugin datasets can not be used in nested child queries.
link	These elements define the link fields between the parent and child records.

Table 30: Dataset “child” Sub-Elements

The “link” element attributes are:

Attribute	Notes
parent	The column name from the parent row which will be used to link to the child records. This must be one of the fields returned by the parent “select” clause. In the given example, the parent field is “name”, and the associated value selected for this row is "Empty Nest".
child	The corresponding variable within the child sub-query “select” definition. In the given example, the value “Empty Nest” is used as {{boat_name}} within the child dataset “select” definition for this sub-query.

Table 31: Dataset Child “link” Sub-Elements

General notes:

- Child datasets may themselves have grand-child datasets, and so on.
- Do not use recursive child relationships. That will not end well.
- All relevant hooks will be invoked for the child datasets.

Regarding links and parameters:

- If multiple keys or filter fields are required for the sub-key, there will be more than one “link” element in the parent dataset definition.
- When the sub-query is executed, only the linked variables are available as user variables for the sub-query (and associated hooks). The top-level CGI parameters and numbered/named REST args are not passed through to the child query.
- Safe parameters (e.g. __username) and application default parameters will be passed to the child sub-query.

Regarding performance:

- Each child query will generate significant overhead with Jarvis as it manages per-dataset configuration, hooks and security.
- In general, nested datasets should be used only for singleton queries.
- If you need to perform child sub-queries on large datasets, you may wish to consider writing a custom plugin with an optimized query sequence.

12.3 Inserting/Updating Nested Datasets

It is also possible to perform "store" operations on nested datasets. The "insert" and "update" actions will perform nested dataset operations. You may also use "mixed", noting that the "_ttype" parameter must be specified on all child data.

Submitting the following JSON request will (with the correct datasets) insert a single top-level master object, and then will also insert three sub-objects. With correct link definitions, the returned parent ID will be passed through for the inserts on the child elements.

```
[
  {
    "parts": [
      {
        "name":"Doodad"
      },
      {
        "name":"Whatsit"
      },
      {
        "name":"Hoosit"
      }
    ],
    "name" : "Mother Hubbard",
    "class" : "X Class"
  }
]
```

An example update is as follows. This updates the class of the top-level "Boat" object, and performs one insert, one update and one delete of the child "Boat Part" elements.

Note that you must supply the necessary key values (e.g. the "id" fields in this case) when updating or deleting an existing parent or child row.

```
[
  {
    "_ttype": 'update',
    "id": "1017" ,
    "name": "Mother Hubbard",
    "class": "Makkleson",
    "parts": [
      { "_ttype": "delete", id: "23",  },
      { "_ttype": "update", id: "24", name: "Whatsitt" },
      { "_ttype": "insert", name: "Thingey" },
    ]
  }
]
```

JSON examples are given. XML examples are left as an exercise for the reader.

13 Exec Datasets

13.1 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/
```

13.2 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 exec 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 exec, 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 exec only. This parameter can not be used to disable global debugging.
dump	<n/a>	Enable “dump” output to standard error for this exec only. This parameter can not be used to disable global dump output.

Table 32: Exec Configuration Attributes

13.3 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 33: Exec Return Mechanisms

13.4 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.

13.5 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.

13.6 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.
- Indexed RESTful parameters specified in the URI. These are passed as “p1”, “p2”, etc.
- Named RESTful parameters assigned by a “route” rule. See the “router” configuration.
- 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.

13.7 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.

14 Plugin Datasets

14.1 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.

14.2 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 “plugin”. Specify a group name, comma-separated list, “*” or “**”. See the documentation for “read” and “write” groups on regular datasets.
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.
mime_type	<none>	Only used if add_headers is “yes”. Allow plugin to override the default MIME type. This takes precedence over any mime type which may be derived from a filename.
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 34: Plugin Configuration Attributes

14.3 Plugin ::do Method

The Plugin is implemented by a “::do” method in a Perl Module. The “::do” method will receive the following args:

- $jconfig – The internal Jarvis context.
- $user_args – A HASH of the CGI args, plus numbered and named RESTful args.
- $plugin_parameters – A HASH of the static plugin parameters from the Jarvis XML file.

The $jconfig context is internal to Jarvis. The following fields are available for read-only access:

Attribute	Notes
$jconfig->{app_name}	Name of Jarvis application.
$jconfig->{cgi}	The CGI object for this CGI request.
$jconfig->{username}	Current Logged-In Username or ''.
$jconfig->{group_list}	Current Logged-In Group List or ''.
$jconfig->{logged_in}	0/1 are we logged in?
$jconfig->{error_string}	Login error message if not logged-in.
$jconfig->{debug}	0/1 is debug tracing enabled?
$jconfig->{dump}	0/1 is dump tracing enabled?
$jconfig->{format}	json/json.rest/xml/csv/xlsx/etc...
$jconfig->{action}	select/insert/update/delete.
$jconfig->{dataset_name}	Name of top-level Dataset implemented by this request.
$jconfig->{dataset_type}	Type of top-level Dataset implemented by this request (i/s/e/p).

Table 35: Available $jconfig Attributes for Plugins

The $jconfig object is also required as a parameter for invoking Jarvis services such as the Jarvis::Error::debug and Jarvis::Error::dump methods.

Note: The $user_args parameter has NOT been checked for safety. It is user-supplied values and must be treated with suspicion.

Note: The “dataset_type” is either “i” = Internal, “s” = SQL, “e” = Exec, “p” = Plugin.

14.4 Output & Headers

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

14.5 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.\n";
```

Note the use of the newline escape at the end of the error. The newline at the end of the error will suppress the default Perl behavior of including the location of the error (file and row number). Normally it would not be appropriate to include the file and line number of the error when explicitly calling die, and in this manner it can be removed.

14.6 Getting Dataset Content with “fetch_rows”.

A plugin may often wish to perform complex calculations and combine multiple datasets in non-standard ways. To support this, Jarvis provides an “official” mechanism for plugins to execute a dataset and get the results for their own purposes.

```
use Jarvis::Dataset;

my $rows = &Jarvis::Dataset::fetch_rows (
               $jconfig, $dataset_name, $user_args, $extra_href
           );
...
```

The $jconfig object is the one given to the plugin.

The $dataset_name is the name of the dataset you wish to execute.

The $user_args are the numbered and named args to use when performing the dataset fetch. Named parameters to the dataset must be provided here. The fetch_rows () method does not access any parameters from the $jconfig->{cgi} object. You may pass undef if you have no user args.

The $extra_href parameter should be a hash reference. Dataset hooks may add additional entries to this hash and will expect the plugin to return them as top-level parameters. You may pass undef if you do not plan to take notice of hook-requested extra top level-parameters.

The result is an ARRAY reference containing the returned rows. Note:

- Sorting and paging may have been applied if the relevant paging parameters were provided.
- Dataset security checks will be performed.

- The Dataset/Global “dataset_pre_fetch” and “dataset_fetched” hooks will be applied.
- Nested datasets will be executed and merged into the results.

14.7 Example Plugin

The Demo application includes a sample “FileDownload” plugin, configured as follows:

```
<jarvis>
    <app>
        <plugin dataset="FileDownload" access="**"
                lib="/usr/share/jarvis/demo" module="plugin::FileDownload"
                add_headers="yes" mime_type="text/plain">

            <parameter name="interview" value="Cross-Sectional"/>
        </plugin>
```

The source code is as follows:

```
use strict;
use warnings;

use Jarvis::Error;
use Jarvis::DB;

sub plugin::FileDownload::do {
    my ($jconfig, $user_args, %plugin_args) = @_;

    # User args includes numbered and named REST args.
    my $rest0 = (defined $user_args->{0}) ? $user_args->{0} : '<undef>';
    my $rest1 = (defined $user_args->{1}) ? $user_args->{1} : '<undef>';
    my $boat_class = (defined $user_args->{boat_class})
                   ? $user_args->{boat_class} : '<undef>';

    # User args also includes the CGI-supplied parameters.
    my $app_name = $jconfig->{app_name};
    my $cgi_myval = (defined $user_args->{cgi_myval})
                   ? $user_args->{cgi_myval} : '<undef>';

    &Jarvis::Error::debug ($jconfig, "App Name: '%s'.", $app_name);
    &Jarvis::Error::dump ($jconfig, "CGI MyVal: '%s'.", $cgi_myval);
    my $interview = $plugin_args{interview} || 'Unknown';

    my $dbh = &Jarvis::DB::handle ($jconfig);
    my $rows = $dbh->selectall_arrayref ("SELECT COUNT(*) as count FROM boat",
                                         { Slice => {} });

    my $num_boats = $$rows[0]{count};
    my $content =
"Param|Value
App Name|$app_name
Interview|$interview
Rest 0|$rest0
Rest 1|$rest1
Boat Class|$boat_class
All Boats|$num_boats";
```

```
    return $content;
}

1;
```

This demonstrates:

- Using the Jarvis debug mechanism.
- 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/boat/X%20Class/download/?filename=x
```

The output is a text file “x.txt” with content similar to the following:

```
Param|Value
App Name|demo
Interview|Cross-Sectional
Rest 0|boat
Rest 1|X Class
Boat Class|X Class
All Boats|45
```

15 Hook Modules

15.1 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.

Note:

- A “store” request is a dataset insert/update/delete operation.
- A “fetch” request is a dataset select operation.

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

Here is the Global Hook invocation matrix.

GLOBAL Hook	**__status**	**__habitat**	**__login**	**__logout**	**Exec**	**Plugin**	**DBI Fetch**	**SDP Fetch**	**DBI Store**
start	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**
return_status	**YES**								
return_fetch							**YES**	**YES**	
return_store									**YES**
finish	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**
after_login	*YES*	*YES*	*YES*	*YES*	*YES*	*YES*	*YES*	*YES*	*YES*
before_logout				**YES**					
pre_connect	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**	**YES**
dataset_pre_fetch							**YES**	**YES**	
dataset_fetched							**YES**	**YES**	
dataset_pre_store									**YES**
dataset_stored									**YES**
before_all									**YES**
after_all									**YES**
before_one									**YES**
after_one									**YES**

Table 36: Global Hooks Invocation

Note that the “after_login” hook is only actually called when:

- The client did not provide a valid session ID.
- The client did provide enough information to be logged-in during this request.

DATASET Hook	**__status**	**__habitat**	**__login**	**__logout**	**Exec**	**Plugin**	**DBI Fetch**	**SDP Fetch**	**DBI Store**
start							**YES**	**YES**	**YES**
finish							**YES**	**YES**	**YES**
dataset_pre_fetch							**YES**	**YES**	
dataset_fetched							**YES**	**YES**	
dataset_pre_store									**YES**
dataset_stored									**YES**
before_all									**YES**
after_all									**YES**
before_one									**YES**
after_one									**YES**

Table 37: Dataset Hook Invocation

Note that:

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

15.2 Hook 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 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 38: Hook Configuration Attributes

15.3 Global Per-Query Hook Points

The Global Per-Query hook points are.

Hook	Notes
GLOBAL start	The start hook method on global hooks is called when the global hook module is first loaded. No dataset analysis has been performed yet. No database transaction is open. Login check has not yet occurred. Args are: • $jconfig, • $hook_parameters_href **$jconfig** is the internal Jarvis configuration object containing the query context. Refer to the description in the preceding “Plugin” chapter. **$hook_parameters_href** is a hash containing the name and value from any <parameter> tags configured for this hook in the application XML.

GLOBAL return_status	This global hook method 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 • $return_text_ref **$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. **$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.
GLOBAL return_fetch	This global hook method 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 • $user_args_href • $results_aref • $extra_href • $return_text_ref **$user_args_href** is the hash of CGI parameters plus numbered and named RESTful args that were used by the top-level dataset. **$rows_aref** is the array of rows returned from the top level dataset query. May included nested child dataset arrays.

GLOBAL return_store	This global hook method 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/XLSX. 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_ref) Args are: • $jconfig • $hook_parameters_href • $user_args_href • $results_aref • $extra_href • $return_text_ref **$results_aref** is the array of returned information that will be passed back to the client. There is one array element for each requested change. Each of these is a HASH with the following elements: • success → 0/1 flag did this update succeed? • message → Error message if not success. • modified → Number of modifications associated with this change. • returning → Array of “returning” values from this change.
GLOBAL finish	The finish hook method on global hooks is called after processing is completed for any operation, 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 39: Global Per-Query Hook Points

15.4 Global Utility Hook Points

The Global Utility hook points are.

Hook	Notes
GLOBAL after_login	This global hook method 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 interactions on this session. Args are: • $jconfig,

Hook	Notes
	• $hook_parameters_href • $additional_safe_href **$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. These changes will affect all requests in the newly created session.
GLOBAL before_logout	This global hook method is called just before the session is deleted from an explicit “__logout” dataset request. The hook cannot prevent the logout, however it may perform auditing functions. Args are: • $jconfig, • $hook_parameters_href
GLOBAL pre_connect	This global hook method allows you to modify the database connection string at the time of connection. This may be useful e.g. if different login users have different, private database instances. Args are: • $jconfig, • $hook_parameters_href • $dbname • $dbtype • dbconnect_ref • $dbusername_ref • $dbpassword_ref • $parameters_href **$dbname** is the identifier for the database within the Jarvis configuration file. E.g. “default”. **$dbtype** is the key “dbi” or “sdp” used within the Jarvis configuration file to distinguish between DBI and other database mechanisms. **$dbconnect_ref** is the DBI or SDP connection string that has been determined. This may be modified by the hook. **$dbusername_ref** is the configured username. The hook may modify. **$dbpassword_ref** is the configured password. The hook may modify. **$parameters_href** is the hash of additional database connection parameters for this database type/name as read from the Jarvis configuration file. The hook may modify. Note that currently there must be an existing <database> entry in the Jarvis configuration file before this hook is invoked. If the configuration file does not contain an entry for the requested database type/dbname then an error will be raised and the hook is not invoked.

Table 40: Global Utility Hook Points

15.5 Per-Dataset Hook Points

The Per-Dataset hook points are.

Hook	Notes
DATASET **start**	For per-dataset hooks, the start hook method is invoked when the dataset configuration is loaded, before any dataset security checking occurs. However, the user security has been completed and any login process will have taken place. Note that dataset hooks receive an additional $dsxml parameter when they are called. Args are: • $jconfig, • $hook_parameters_href • $dsxml **$dsxml** is an XML::Smart object containing the dataset's XML definition.
GLOBAL/DATASET **dataset_pre_fetch**	This global or per-dataset hook method is called only for “fetch” (select) requests. It is called before the SQL select statement is constructed. Args are: • $jconfig • $hook_parameters_href • $dsxml • $safe_params_href **$safe_params_href** is the hash of safe variables, indexed REST args and named REST args that apply to all rows. The hook may modify values in $safe_params_href to change the input parameters to the select statement.
GLOBAL/DATASET **dataset_pre_store**	This global or per-dataset hook method is called only for “store” (insert/update/delete/mixed) requests. It is called before the SQL statements are constructed. Args are: • $jconfig • $hook_parameters_href • $dsxml • $safe_params_href • $rows_aref The hook may modify values in $safe_params_href to change the input parameters to the store statements. **$rows_aref** is a reference to the array of row changes to be stored. The hook may change these parameters and may also add/remove rows.

Hook	Notes
GLOBAL/DATASET before_all	This global or per-dataset hook method is called after Jarvis has loaded the dataset configuration. 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 • $safe_params_href • $fields_aref **$fields_aref** is the array of user change objects submitted for this request. Changes to values in this array will be reflected in the subsequent insert/update/delete operations. Changes to $safe_params_href will affect only the <before> SQL.
GLOBAL/DATASET after_all	This global or per-dataset hook method 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 • $safe_params_href • $fields_aref • $results_aref **$results_aref** is a reference to the returned row 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.

Hook	Notes
GLOBAL/DATASET dataset_fetched	This global or per-dataset hook method is called only for “fetch” (select) requests. It is called after the SQL select is performed. 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 • $safe_params_href • $rows_aref • $extra_href • $column_names_aref **$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 objects in this array will be represented in the data returned to the client. Note that you must take care with $extra_href when using nested datasets, as child datasets may conflict with parent datasets if they are both setting return global attributes. **$column_names_aref** is a reference to the array of all the column names returned by the SQL query.
GLOBAL/DATASET dataset_stored	This global or per-dataset hook method is called only for “fetch” (select) requests. It is called after the SQL select is performed. 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 • $safe_params_href • $results_aref • $extra_href **$results_aref** is the same format as described for the “return_store” hook. Note that you must take care with $extra_href when using nested datasets, as child datasets may conflict with parent datasets if they are both setting return global attributes.

Hook	Notes
DATASET finish	The finish hook method on dataset hooks is called after processing is completed for the dataset, but before the result is returned to the client. It is called before any global return_fetch or return_store hooks are invoked. Any database transaction associated with the dataset is now finished. The hook module should use this hook only for cleanup and/or auditing. This hook is not suitable for modifying the content returned to the client. Please use a global return_fetch or return_store hook method instead. Args are: • $jconfig • $hook_parameters_href • $dsxml

Table 41: Per-Dataset Hook Points

15.6 Per-Row Hook Points

Hook	Notes
GLOBAL/DATASET before_one	This global or per-dataset hook method is called after Jarvis has assembled all parameters in order to execute a single insert/update/delete statement. It is called before the actual execution of the row modification SQL. Args are: • $jconfig • $hook_parameters_href • $dsxml • $safe_row_params_href **$safe_row_params_href** is the hash of variables to be bound to the single SQL row update. Changes to values in this hash will be reflected in the database for this row update only.

Hook	Notes
GLOBAL/DATASET after_one	This global or per-dataset hook method 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 • $safe_row_params_href • $row_result_href **$safe_row_params_href** is the hash of variables previously bound to the single SQL row update. Changing values in this hash will have no effect. **$row_result_href** is the contents of any returning row. The hook code may change values in this hash and the change will be reflected in the corresponding object returned to the client.

Table 42: Per-Row Hook Points

15.7 Sample Global Hook Template

The following empty module should help you get your first Global Hook module under way. You may safely delete any hook methods you do not wish to implement.

```
use strict;
use warnings;

# PER QUERY GLOBAL HOOKS

# Global "start" hook.
# CALLED: After all Jarvis setup is complete.
sub hook::ExampleGlobalHook::start {
    my ($jconfig, $hook_params_href) = @_;
    return 1;
}

# Global "return_status" hook.
# CALLED: Before returning result of a "__status" request.
sub hook::ExampleGlobalHook::return_status {
    my ($jconfig, $hook_params_href, $extra_href, $return_text_aref) = @_;

    return 1;
}

# "return_fetch" hook.
# CALLED: Before returning result of a “fetch” request.
sub hook::ExampleGlobalHook::return_fetch {
    my ($jconfig, $hook_params_href, $user_args_href, $rows_aref,
                  $extra_href, $return_text_aref) = @_;
    return 1;
}
```

```
# "return_store" hook.
# CALLED: Just before returning result of a "store" request.
sub hook::ExampleGlobalHook::return_store {
    my ($jconfig, $hook_params_href, $user_args_href, $results_aref,
                  $extra_href, $return_text_aref) = @_;
    return 1;
}

# "finish" hook.
# CALLED: After Jarvis processing is complete, at cleanup time.
sub hook::ExampleGlobalHook::finish {
    my ($jconfig, $hook_params_href) = @_;
    return 1;
}

# UTILITY GLOBAL HOOKS

# "after_login" hook.
# CALLED: After first login for the session.
sub hook::ExampleGlobalHook::start {
    my ($jconfig, $hook_params_href, $additional_safe_href) = @_;
    return 1;
}

# "before_logout" hook.
# CALLED: Before explicit session logout by the client
sub hook::ExampleGlobalHook::before_logout {
    my ($jconfig, $hook_params_href) = @_;
    return 1;
}

# "pre_connect" hook.
# CALLED: Before first connecting to each database.
sub hook::ExampleGlobalHook::pre_connect {
    my ($jconfig, $dbname, $dbtype, $dbconnect_ref, $dbusername_ref,
                  $dbpassword_ref, $parameters_href) = @_;
    return 1;
}

# PER-DATASET GLOBAL HOOKS

# "dataset_pre_fetch" hook.
# CALLED: ("fetch" only) Before constructing the fetch request.
sub hook::ExampleGlobalHook:: dataset_pre_fetch {
    my ($jconfig, $hook_params_href, $dsxml, $safe_params_href) = @_;
    return 1;
}

# "dataset_pre_store" hook.
# CALLED: ("store" only) Before constructing the store request.
sub hook::ExampleGlobalHook:: dataset_pre_store {
    my ($jconfig, $hook_params_href, $dsxml, $safe_params_href,
                  $rows_aref) = @_;
    return 1;
}

# "before_all" hook.
```

```
# CALLED: ("store" only) After transaction begins.  Before any "before" SQL.
sub hook::ExampleGlobalHook::before_all {
    my ($jconfig, $hook_params_href, $dsxml, $safe_params_href,
                  $fields_aref) = @_;
    return 1;
}

# "after_all" hook.
# CALLED: ("store" only) After any "after" SQL.  Before transaction ends.
sub hook::ExampleGlobalHook::after_all {
    my ($jconfig, $hook_params_href, $dsxml, $safe_params_href, $fields_aref,
                  $results_aref) = @_;
    return 1;
}

# "dataset_fetched" hook.
# CALLED: ("fetch" only) After fetching results array for a single dataset.
sub hook::ExampleGlobalHook::dataset_fetched {
    my ($jconfig, $hook_params_href, $dsxml, $safe_params_href, $rows_aref,
                  $extra_href, $column_names_aref) = @_;
    return 1;
}

# "dataset_stored" hook.
# CALLED: ("store" only) After storing results array for a single dataset.
sub hook::ExampleGlobalHook::dataset_stored {
    my ($jconfig, $hook_params_href, $dsxml, $safe_params_href,
                  $results_aref, $extra_href) = @_;
    return 1;
}

# PER-ROW GLOBAL HOOKS

# "before_one" hook.
# CALLED: ("store" only) Before we execute the row insert/update/delete SQL.
sub hook::ExampleGlobalHook::before_one {
    my ($jconfig, $hook_params_href, $dsxml, $safe_params_href) = @_;
    return 1;
}

# "after_one" hook.
# CALLED: ("store" only) After insert/update/delete and "returning" extraction.
sub hook::ExampleGlobalHook::after_one {
    my ($jconfig, $hook_params_href, $dsxml, $safe_params_href,
                  $row_result_href) = @_;
    return 1;
}

1;
```

15.8 Sample Dataset Hook Template

A dataset hook module may use any of the Per-Dataset or Per-Row hooks listed above. Note that the “start” hook for a Dataset has the additional **$dsxml** parameter available.

```
use strict;
```

```
use warnings;

# PER-DATASET DATASET HOOKS

# Dataset "start" hook.
# CALLED: ("fetch" AND "store") Just after loading dataset config.
sub hook::ExampleGlobalHook::start {
    my ($jconfig, $hook_params_href, $dsxml) = @_;
    return 1;
}

# Others: Same as the Global Hooks of the same name.
# sub dataset_pre_fetch ()
# sub dataset_pre_store ()
# sub before_all ()
# sub after_all ()
# sub dataset_fetched ()
# sub dataset_stored ()

# PER-ROW DATASET HOOKS

# Same as the Global Hooks of the same name.
# sub before_one ()
# sub after_one ()

1;
```

15.9 Exception Handling

To handle an exception in your hook, 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 "Hook cannot continue, moon not found in expected phase.\n";
```

Note the use of the newline escape at the end of the error. The newline at the end of the error will suppress the default Perl behavior of including the location of the error (file and row number). Normally it would not be appropriate to include the file and line number of the error when explicitly calling die, and in this manner it can be removed.

16 Performance Notes

16.1 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

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.