Taverna 2.5.1 Server: Installation and Administration Guide

This document relates to the first public release of Taverna Server 2.5 that is based on the Taverna 2.5 Platform, from the myGrid team at the University of Manchester.

About

This release is a version of the Taverna 2.5 Server that is provided as a basis for deployments of server-ized Taverna in a multi-user environment.

In addition to its improved performance, this release supports a number of new features:

- *Major Feature:* Support for Taverna **Components**.
- *Major Feature:* Support for the **Interaction** "service".
- *Major Feature:* Updated execution engine to latest version (2.5).
- Start a workflow run by supplying a reference to a workflow document.
- Set an arbitrary name for a workflow run.
- Describes system capabilities, so that it is possible to determine whether a workflow can be run prior to sending that workflow to the server.
- Improved resource management and logging.

This is in addition to these features supported by Taverna Server 2.4.

- Based on **Taverna 2.4**
- **Multiple users**, with strong separation between them.
- Limited **persistence over service restarts**, depending on exact deployment.
- **Workflow run introspection** capabilities; clients can ask the server what inputs they should supply and what outputs were provided.
- **Workflow run termination notifications** through multiple mechanisms (RSS feed, email, SMS, twitter, etc. depending on deployment).
- **Security**, both of access to the service and access by the workflow runs to other services.
- **Administrative REST interface** including resource accounting
- Streaming of **large files** both for download and upload.

And these features of Taverna Server 2.2:

- **Upload and Execution of arbitrary Taverna 2 workflows**
- **Access to Workflow's Interim Output Files**; no need to wait for the workflow to finish if the results are available sooner
    - **Safe File Management** for handling results; workflows cannot interfere with each others files
- Simple mechanism for **Removal of Expired Workflows**
- Support for both **RESTful and SOAP APIs**, for easier tooling

- **JMX-based Management API**

There remain a number of known-missing features; notably these include:

- Support for execution on a back-end cluster, Cloud or Grid.
- Access to the workflow run provenance information (other than by downloading a raw Apache Derby database).
- Full access to the workflow run working directory via WebDAV.
- Fully surfaced workflow execution model, including to intermediate state information.

Installation

Prerequisites

You will need **Unix** of some kind.

> This software was developed against Debian Linux 5.0.9, but we anticipate that these instructions will apply equally to other Unixes.
>
> *The system security integration does not work with the Windows security model, and the restrictions on command-line lengths on that platform are another significant issue. Because of this, Taverna Server 2.5 does not support Windows at this stage. Tools and services hosted on Windows* **may** *be used, but only via remote access protocols such as SSH, SOAP, etc.*

You will need a **Java 7** (or later) installation.

> This software was developed and tested against the Oracle JRE 1.7.0_21, and has also been tested against OpenJDK 7. Use of the current version of Java is *recommended*. OpenJDK 6 is known to be problematic, and should not be used.

You will need a suitable **servlet container**.

> This software was developed using Tomcat 6.0.26 as the servlet container, but other versions of Tomcat are known to work (back to at least 6.0.20) and other containers may also function correctly as no Tomcat-specific APIs are used in the deployable code. We welcome feedback on which containers work, as well as on how to configure them (if they are not Tomcat versions). Using the current production release of Tomcat 6 is *recommended*.

Installation into Tomcat

Note that these instructions are Tomcat-specific.

Step 1. Configure Tomcat for JMX

If you're going to use JMX to administer the server (good for demos; `jvisualvm` is recommended if you've got the JMX support plugin, and `jconsole` is acceptable) then you need to edit Tomcat's *«TOMCATDIR»*`/bin/startup.sh` script to include the setting:

```
export CATALINA_OPTS=-Dcom.sun.management.jmxremote
```

This works around a minor bug in Spring which prevents correct registration of management beans in the default internal management service. You should also add additional options there to ensure that the JMX management layer is secure; see the Java JMX documentation for a discussion of how to do this.

Step 2. Configure Tomcat for General Management

Add a user entry in *«TOMCATDIR»*`/conf/tomcat-users.xml` so that the manager webapp can know who you are and that you have permission to deploy webapps (i.e., the "manager" role).

You also *need* to configure Tomcat to support HTTPS if you are planning to use the default secure configuration; to do this, follow the instructions on the Tomcat site (http://tomcat.apache.org/tomcat-6.0-doc/ssl-howto.html).

> *The default configuration of the server* ***requires HTTPS*** *— either on port 443 or on port 8443 — and will refuse to provide the large majority of its operations (including all SOAP actions and all REST actions relating to workflow runs) if it is not detected. We* ***recommend*** *that a public Certificate Authority sign the public certificate of the server, as this tremendously simplifies client deployment.*
>
> *The* non-default *insecure configuration does not require connections to be made by HTTPS or that different users be configured. When it is used, other steps should be taken to ensure that the connections to the server are secure (e.g., through the use of a strong firewall and a co-located front-end service or a VPN).*
>
> *There is also a part-secure configuration that requires HTTPS, but which does not enable the advanced user separation mechanism. This makes it easier to configure, but reduces the degree of separation between users' workflow runs.*

Now start Tomcat (or restart it).

Step 3. Prepare for T2Server WebApp Installation

Save the text below as `context.xml` on the machine where you are going to install the server. This is the minimum content of that file:

```
<Context path="/taverna-server">
</Context>
```

Additional configuration properties can be set in the `context.xml` file; see the detailed deployment description section of this document for more information.

Step 4. Download the Webapp ARchive

Make sure that the `.war` file is also saved to the machine on which you will be installing the server.

Step 5. Install the WebApp

Navigate to `http://`*«SERVER:PORT»*`/manager/html` and go to the Deploy box. Fill in the form there with:

Field	Value
Context Path (required):	`/taverna-server`
XML Configuration File URL:	`file:/path/to/context.xml`
WAR or Directory URL:	`file:/path/to/TavernaServer.war`

Press the Deploy button; after a few seconds, Tomcat should respond with OK (at the top of the reloaded page) and you'll have the Taverna Server webapp installed at `http://`*«SERVER:PORT»*`/taverna-server`.

Note that you **should** also review the section below on impersonation via `sudo`, which describes a reasonable minimal approach for securing the invocation of workflows as limited-authority users. Also be aware that *many features of this server software are disabled by default*, especially in relation to the pushing of notifications through other services (e.g., email, SMS). These features would be set through the use of context parameters, as described in the next section.

Firewall Requirements

Taverna Server is a fairly network-intensive application, and the workflows it runs typically assume that they have access to a wide range of external ports by default. Because of that, it is **recommended** that outgoing ports on the system running Taverna Server be mainly not blocked (a few exceptions are reasonable, such as preventing access to external SMTP services). In particular, it is not safe to just assume that all web services accessed by workflows run on "standard" ports like 80 or 443; this is not seen in practice.

For *incoming* ports, we recommend restricting them as much as is practical. In particular, we note that port 1099 is used by the service along with a substantial number of high-numbered ports: none of those need any access from outside the host system. The only ports that need to be opened are those that handle incoming user requests: in the default configuration of Tomcat, this would be port 8080 for HTTP traffic and 8443 for HTTPS traffic. (You will probably want to keep some other ports at least partially open for administration traffic, e.g., 22 for *ssh* access, but this is formally outside the scope of this document: Taverna Server itself does not need that.)

If you configure your firewall (or some sort of proxy) so that the host, port and web-application root that the user sees is not that which you are actually running the service on, you should set the *default.webapp* application parameter — typically via a deployment parameter — to a URL fragment giving the host, port and webapp root that you wish to use. (Taverna Server forces the protocol used separately.)

Details of Configuration

Deployment of web applications into Tomcat can be done through multiple mechanisms, notably through command line tools and through Tomcat's online administration interface. This document describes the latter mechanism. Note also that we currently only support the use of Taverna Server within a Unix environment (e.g., Linux, Mac OS X); there is no reason in principle why the code should not be adaptable to Microsoft Windows, but there is currently no impersonation module written to integrate Taverna Server with that operating platform.

This assumes that you installing into Tomcat 6 running on top of Java 7; this was tested with Tomcat 7.0.21 over the Oracle JRE 1.7.0_21. Later patch versions from the same major version are recommended, if available. *This software has not been fully tested with Tomcat 7.*

The configuration of the Taverna Server installation is done by writing a context descriptor document, only some parts of which can be configured afterwards via the management interface. An *example* of that XML document is below:

```
<Context path="/taverna-server">
    <!-- Sample logging configuration. -->
    <Valve className="org.apache.catalina.valves.AccessLogValve" />

    <Parameter name="default.localusername"
        value="localtavernauser" />

    <!-- For email-dispatched notifications. -->
```

```
    <Parameter name="email.host" value="localhost" />
</Context>
```

The context descriptor is typically in a file called `context.xml` and there is a sample context descriptor with this distribution, in the `context.sample.xml` file. There are a substantial number of properties that may be tuned during installation (see below).

The actual deployment is done by giving the actual context location (i.e., the base URL of the webapp relative to the whole Tomcat container) as a separate field, together with URLs (it is useful to use file: URLs for this) to the context descriptor document and the distributed WAR file.

Configuration Property List

This is a list of all the properties that are set by default in the Server (NB: the properties in red are actually unset or commented out by default, but they are all understood). They may be *all* overridden by the use of context configuration parameters as described above; for example, to override the default local user name, the `default.localusername` configuration parameter would be set.

The majority of these properties should not be set, as they default to reasonable parameters. The exceptions are:

- Those used to control how many runs exist and are operating at once. The `default.runlimit` and `default.operatinglimit` have reasonable initial values, but must be tuned according to the actual expected workload. They can be set at runtime via the administration web interface and via JMX.
- Those used to enable optional notification mechanisms; those are all disabled by default unless the required extra properties are set (see below for instructions).

```
# Script in Taverna installation to run to actually execute workflows
executeWorkflowScript:      /usr/taverna/executeworkflow.sh

# Override the hostname, port and webapp; leave at 'NONE' if no
# override desired. If set, set it to something like:
#      foo.example.com:8000/tav-serv
default.webapp:             NONE

# User name to use by default for impersonation if nothing else
# specified
default.localusername:      taverna

# The HTTP authorization realm; should be different from all other
# webapps on the deployment server
http.realmName:         tavernaserver

# Force the RMI registry to only listen to connections from localhost
# Should be true unless you have a good reason to open it up.
rmi.localhostOnly:      true

# How to pick a user name out of a global identity
localusernameregexp:    ^TAVERNAUSER=(.*)$
# Whether to log incoming workflows; noisy if enabled
default.logworkflows:   false
```

```
# Whether to log outgoing exceptions; noisy if enabled
default.logexceptions:  false
# Whether to allow workflows to be submitted; adjustable via admin
# interfaces
default.permitsubmit:   true
# How long a workflow run should live by default, in seconds
default.lifetime:       1440
# Maximum number of simultaneous workflow runs, in any state
default.runlimit:       100
# Maximum number of simultaneous *operating* workflow runs (i.e.,
# that are actually running)
default.operatinglimit: 10

# Location of impersonation credentials
secureForkPasswordFile:     /usr/local/tomcat6.0/conf/sudopass.txt

# URI to server's REST interface
taverna.preferredUserUri:
    https://some.host:8443/taverna-server/rest/

# Delays used in the task executor, both in milliseconds
purge.interval:             30000
finish.interval:            10000

# Thread pool sizing
pool.size:                  2

### Usage Record handling
usage.logFile:              none
usage.disableDB:            no

### General configuration of messaging
# cooldown in seconds
message.cooldown:           300
message.termination.subject:  Taverna workflow run finished
message.termination.body:   Your job with ID={0} has finished with
                            exit code {1,number,integer}.

### Email-specific options
email.from:                 taverna.server@localhost
email.type:                 text/plain
email.host:                 localhost

### Jabber-specific options
xmpp.server:                xmpp://some.host:5222
xmpp.resource:              TavernaServer
xmpp.user:                  taverna
xmpp.password:              *******

### Atom/RSS feed; lifespan in days, cleaninterval in milliseconds
atom.language:              en
atom.lifespan:              7
atom.cleaninterval:         3600000

### SMS-specific options
sms.service:
        https://www.intellisoftware.co.uk/smsgateway/sendmsg.aspx
sms.userfield:              username
sms.passfield:              password
sms.destfield:              to
sms.msgfield:               text
```

```
sms.user:                    taverna
sms.pass:                    *******

### Twitter-specific options
twitter.oauth.accessToken:         *******
twitter.oauth.accessTokenSecret:   *******

### Special options
# Do detailed logging of security. The information logged is enough
# to allow an administrator to recover arbitrary user credentials,
# so this should be false under normal circumstances.
log.security.details:    false

# Enables a special project-specific security setting.
# Leave at false unless you have been specifically told otherwise.
helio.cis.enableTokenPassing:    false
```

Enabling Notification Options

With the exception of the Atom feed, all the notification methods supported by Taverna Server are *disabled* by default. They require additional configuration in order to work correctly. This is done through configuration options. Only once all options for a particular notification method are set will Taverna Server enable that method for use. Note that many of these methods require additional services to be deployed or accounts with remote services to be created.

Method	"URL" Scheme	Properties
General	*N/A*	`message.cooldown` — All the notification methods below *except for the Atom feed* are subject to rate limiting; this property is the minimum amount of time (in seconds) between two notifications by the same mechanism. The default is 300 seconds (5 minutes). `message.termination.subject` — Where a notification mechanism needs to attach a subject to a message, this property contains the value to use. Does not need to be changed unless the service is being adapted to use a language other than English. `message.termination.body` — This property contains a template that is used to produce a notification message. The template itself contains `{0}` to indicate where the terminating run's ID goes, and `{1}` (or its derivatives) to indicate where the termination code goes. Does not need to be changed unless the service is being adapted to use a language other than English.

Method	"URL" Scheme	Properties
Atom	*N/A*[1]	`atom.language` — Language to claim that the message is published in. Recommended left at default (English of unspecified locale). `atom.lifespan` — How many days will a particular notification persist in the feed. `atom.cleaninterval` — How often will the code check for whether it can remove a particular notification, in milliseconds.
Email	`mailto:`	`email.host` — Name of a machine with a suitable relaying SMTP server that all emails will be sent via. **Must be set** to enable this notification method. `email.from` — What account will the message appear to be sent from. Changing from the default is *recommended*. `email.type` — What MIME type will be used with the message. Can be left at the default.
Jabber	`xmpp:`	`xmpp.server` — Name of a machine that runs a suitably-configured XMPP server. **Must be set** to enable this notification method. `xmpp.user` — User name to use when contacting the XMPP server. **Must be set** to enable this notification method. `xmpp.password` — Password to use when contacting the XMPP server. **Must be set** to enable this notification method. `xmpp.resource` — Resource descriptor used to disambiguate messages sent by Taverna Server. Can be left at the default.

[1] This is always enabled; termination notifications are automatically published to a per-user Atom feed that users cannot post directly to. This feed is located at `http://`*«SERVER:PORT»*`/taverna-server/feed` relative to the webapp root resource.

Method	"URL" Scheme	Properties
SMS	`sms:`	`sms.service` — Address of a RESTful SMS service interface for sending SMS messages. **Must be set** to enable this notification method. Note that this has only ever been developed against a single service interface[2], and is not guaranteed to work with any other. `sms.user` — The user account to use with the above service. **Must be set** to enable this notification method. Note that creating such an account has some financial implications; these are out of the scope of this document. `sms.pass` — The password to use with the above service. **Must be set** to enable this notification method. `sms.userfield` — The name of the field to use for the user name when conveying the message in the POST request. Allows for limited adaptation to other services, but may be left at the default. `sms.passfield` — The name of the field to use for the password when conveying the message in the POST request. Allows for limited adaptation to other services, but may be left at the default. `sms.destfield` — The name of the field to use for the destination phone number when conveying the message in the POST request. Allows for limited adaptation to other services, but may be left at the default. `sms.msgfield` — The name of the field to use for the message content when conveying the message in the POST request. Allows for limited adaptation to other services, but may be left at the default.
Twitter	`twitter:`	`twitter.oauth.accessToken` — The public part of the OAuth access token to use when authenticating a Taverna Server deployment to Twitter. **Must be set** to enable this notification method. Note that users must take additional steps to allow the ability to set status messages; this is outside the scope of this document. `twitter.oauth.accessTokenSecret` — The private part of the OAuth access token to use when authenticating a Taverna Server deployment to Twitter. **Must be set** to enable this notification method. Note that users must take additional steps to allow the ability to set status messages; this is outside the scope of this document.

[2] `https://www.intellisoftware.co.uk/smsgateway/sendmsg.aspx`

User Accounts

Once you have deployed the server, you can use either JMX or the `http://«SERVER:PORT»/taverna-server/admin` interface to create and manage accounts. Only accounts with administrative permission can do such management. The initial set of users is loaded into the database from the `WEB-INF/security/users.properties` file in the deployment package; see the comments in that file for a more complete description of its contents; the file is only used if the user database is empty.

By default, two enabled users are created. One is a normal user (`taverna`, with password `taverna`) and the other is an administrative user (`admin`, password `admin`). These defaults ***should be changed*** after installation, as they are not considered secure by default. More information about the mapping process is in the security summary document.

Configuration of Impersonation

If it is desired to separate each user of Taverna Server from the others, it is **necessary** to configure impersonation of users. That is, the user account that is running the servlet container (Tomcat, etc.) must have permission somehow to execute code as other users. (If this is not desired, the service should be configured to use the simpler non-impersonating worker factory — see the `backEndFactory` property above — or the fall-back user identity to use for impersonation should be set in the `default.localusername` to the same identity as the user account used for running the server.)

This is done by either instructing the service what password is to be used with `sudo` (typically the password for the account that is invoking the `sudo` command) or by configuring `sudo` itself so that the service account is more highly authorized than a normal account. The first style of impersonation, which requires that the service account have a password at all, is enabled by creating a file (in a suitably secured directory) that contains the password as its only content, and telling Taverna Server about it during deployment by giving the full pathname of the file in the `secureForkPasswordFile` deployment parameter.

The second style of impersonation is done by leaving that parameter unset and instead adding some extra configuration to the system's `/etc/sudoers` file, as seen below (typically set with the `visudo` command). Note that conventionally the three parts of the configuration are in separate sections of the file, and that care should be taken during configuration as mistakes can result in a system that is broken. In the example below, we assume that the servlet container is running as the Unix user `tavserv` and that local user accounts that may be targets for impersonation are all members of the `taverna` UNIX group.

```
# Flags for the tavserv user (keep things quiet)
Defaults:tavserv   !lecture, timestamp_timeout=0, passwd_tries=1

# Who can we impersonate? Manage via Unix group called 'taverna'
Runas_Alias        TAV = %taverna

# The actual permission to impersonate, with permission to run
# anything
```

```
tavserv             ALL=(TAV) NOPASSWD: ALL
```

Care should be taken as without a password specified and without permission to execute as another user, an attempt to create a workflow run will hang instead of failing.

Security in Taverna 2 Server

General

Taverna Server normally operates in a mode where it executes each user's workflow runs under a user-id that is specific to that user. This keeps the users from seeing each other's workflow runs by back-door mechanisms, and makes it far easier to apply standard server resource accounting.

In order to do this, it needs to be able to run code (specifically, a Java program) as effectively arbitrary other users. On Unix (currently the only fully supported hosting platform) this is implemented through the use of `sudo` with a special configuration, which allows the user hosting the Java container special access. Because of this, it is strongly **recommended** that other web applications be not run in the same container, or that if it is necessary to share webapps that way, the subprocess execution module be instructed where to find a password for use with the `sudo` thunk.

It is **recommended** that Taverna Server always be operated in secure mode, with all connections normally being made via HTTPS. Given that this requires that the container be configured with an SSL certificate, it should be noted that a single-host certificate is available from many certificate authorities for extremely limited cost (even free in some cases). Please consult your container's documentation on how to install the SSL certificate and configure the container for HTTPS operation.

If JMX is used for the management interface (depends on the container) it is **recommended** that it be configured to only accept authenticated connections over SSL. There is also an `http://`*«SERVER:PORT»*`/taverna-server/admin` REST interface to the server, which allows access to the same capabilities; it is only accessible to users which have the `ROLE_tavernasuperuser` authority granted to them. Not all parts of the configuration can be managed in this way though; some properties are sufficiently fundamental that they can only be set through the configuration of the deployment descriptor.

Architecture

The communication between the back end workflow executor managers and the front-end webapp is done via RMI, which has been configured to not accept connections from off the local host or class definitions that it does not already know about.

> *You **should** configure your firewall to not permit incoming connections to port 1099 (the default RMI registry port). This is not critical though; if there is no existing RMI registry running on the machine, the version created will not allow connections from other than (the non-routable) local address, 127.0.0.1, by default.*

Authorisation of users is done through the use of Spring Security to assign them *on each connection* a set of security authorities. In particular, the key authorities are:

- `ROLE_tavernauser` — allows the user to access the main operational parts of the server.
- `ROLE_tavernasuperuser` — allows the user to read and write all non-security characteristics of all workflows, and also grants access to the `http://`*«SERVER:PORT»*`/taverna-server/admin` pages.
- `LOCALUSER_*` (where the * is replaced with a local user name) — specifies what local user name the user should be executing workflows as; the prefix (`LOCALUSER_`) is simply stripped and the remainder is used as a user name. If absent, the default user name (`taverna` in the default configuration) will be used; two users mapped to the same user name will be able to see each others workflows if they can work out where the job working directories are located, but will not be able to see them inside Taverna Server itself (unless one user grants the other authority to do so, of course).

The default source of authorities is the file `WEB-INF/security/users.properties` (relative to the directory which is the expanded webapp) that is used to populate the database if that is empty.

Insecure Mode

The server can be switched into insecure mode by editing its `WEB-INF/web.xml` file so that it pulls its Spring configuration from `insecure.xml` instead of from `secure.xml` (the default) via the `contextConfigLocation` parameter. When editing `WEB-INF/web.xml`, the webapp must be stopped and restarted for any changes to be noticed. This alternate configuration disabled URI rewriting, restricts the set of users to a single one (`taverna` with a password `taverna`) and arranges for execution of workflow runs to be done in the same local userid as is running the host servlet container (Tomcat, etc.)

If you are using this, it is *strongly* ***recommended*** that you place the server behind a strong firewall and portal, and only permit vetted workflows to be used.

Part-Secure Mode

There is a partially-secured configuration as well. This enables the forced use of HTTPS, but disables the use of user separation by the back-end engine, giving an intermediate level of security suitable for the case where the network is *not* trusted but the permitted users *are* trusted. You may enable this mode by using `partsecure.xml` as the value of the `contextConfigLocation` parameter in `WEB-INF/web.xml` after installation (the webapp must be stopped while you make this change). Note that because HTTPS is being used, you will still need to configure the servlet container with an SSL keypair for this to work.

Managing the Server

The server is designed to support JMX for management. This allows the use of tools such as `jconsole` or `jvisualvm` (with appropriate plugin) to connect to the server so that they can view, chart, and manipulate properties of the server. The exact list of properties is liable to change, but is as follows in this release:

Component: Taverna/Server/Webapp

This is the component that interfaces with the external world.

Property	Type	Description
`AllowNewWorkflowRuns`	*Writable*	Whether to permit any new workflow runs to be created; has no effect on existing runs.
`CurrentRunCount`	*Read-Only*	Count of currently existing runs.
`InvocationCount`	*Read-Only*	Count of SOAP and REST calls made to the Webapp.
`LogIncomingWorkflows`	*Writable*	Whether to put submitted workflows in the log.
`LogOutgoingExceptions`	*Writable*	Whether outgoing exceptions should be extensively logged.

Component: Taverna/Server/RunFactory

This component is responsible for manufacturing workflow runs and maintaining connections to existing runs. Note that the writable properties typically have sensible values by default.

Property	Type	Description
`CurrentRunNames`	*Read-Only*	The names of the currently existing runs.
`DefaultLifetime`	*Writable*	How many minutes should a workflow live by default?
`ExecuteWorkflowScript`	*Writable*	The full pathname of the script to run to start running a workflow. Must be *readable* by any user of the system.
`ExtraArguments`	*Writable*	The list of additional arguments used to make a worker process.
`FactoryProcessMapping`	*Read-Only*	The mapping of user names to RMI factory IDs.
`JavaBinary`	*Writable*	The full pathname of the Java executable to run.
`LastExitCode`	*Read-Only*	What was the exit code from the last time the factory subprocess was killed?
`LastStartupCheckCount`	*Read-Only*	How many checks were done for the worker process the last time a spawn was tried. (Larger values indicate problems with system loading.)
`MaxRuns`	*Writable*	The maximum number of simultaneous runs supported by the server. Note that this includes runs that have finished executing but have not yet been deleted.

Property	Type	Description
`OperatingLimit`	*Writable*	The maximum number of simultaneous operating runs supported by the server. This is only the runs that are in the *Operating* state.
`PasswordFile`	*Writable*	The full pathname of a file containing a password to use when running a program as another user (e.g., with `sudo`).
`RegistryHost`	*Writable*	The host holding the RMI registry to communicate via.
`RegistryPort`	*Writable*	The port number of the RMI registry. Should not normally be set.
`ServerForkerJar`	*Writable*	The full pathname of the JAR implementing the secure-fork process.
`ServerWorkerJar`	*Writable*	The full pathname of the JAR implementing the server worker processes.
`SleepTime`	*Writable*	How many milliseconds to wait between checks to see if a worker process has registered.
`TotalRuns`	*Read-Only*	How many times has a workflow run been spawned by this engine.
`WaitSeconds`	*Writable*	How many seconds to wait for a worker process to register itself before causing the creation operation to fail.

Component: Taverna/Server/Users

This is an interface for adding, deleting and otherwise managing user accounts on the server. It does not manage the underlying system accounts, but does allow control over the mapping of users to those accounts. Note that newly created accounts are disabled by default. More information about the mapping process is in the security summary document.

Property	Type	Description
`UserNames`	*Read-Only*	The list of server accounts known about.

Operation	Description
`addUser`(*nm*,*pw*,*cpl*)	Adds the user called *nm* to the database, with password *pw*. If *cpl* is true, set the local user account to be the same as the user name, otherwise use a default set at system configuration time. The user will be a non-admin and disabled by default.
`deleteUser`(*nm*)	Remove the user called *nm* from the database.
`getUserInfo`(*nm*)	Get a description of the user called *nm* from the database.
`setUserAdmin`(*nm*,*ad*)	Set whether the user called *nm* is an admin or not (according to the boolean, *ad*).
`setUserEnabled`(*nm*,*en*)	Set whether the user called *nm* is an admin or not (according to the boolean, *en*).
`setUserLocalUser`(*nm*,*lu*)	Set what the user called *nm* will be mapped to as a local user to *lu* (which must be the name of an account understood by the local system).
`setUserPassword`(*nm*,*pw*)	Set the password for the user *nm* to be *pw*. This imple-

mentation stores the value directly in the database.

The server also supports a RESTful administration interface on its `http://«SERVER:PORT»/taverna-server/admin` resource (a sibling to the main RESTful `http://«SERVER:PORT»/taverna-server/rest` resource and the Atom feed on `http://«SERVER:PORT»/taverna-server/feed`). This interface is only available to users who authenticate with admin permissions. Currently, there is no rendering of the interface in a form that is suitable for use from a normal web browser; this is expected to change in future versions.