Elixys Web Server Interface

Contents

[Overview 2](#_Toc296991171)

[Configuration 2](#_Toc296991172)

[User 3](#_Toc296991173)

[Client State 4](#_Toc296991174)

[Server State 4](#_Toc296991175)

[Sequences 6](#_Toc296991176)

[Components 9](#_Toc296991177)

[CASSETTE 9](#_Toc296991178)

[ADD 9](#_Toc296991179)

[EVAPORATE 10](#_Toc296991180)

[TRANSFER 10](#_Toc296991181)

[ELUTE 10](#_Toc296991182)

[REACT 11](#_Toc296991183)

[PROMPT 11](#_Toc296991184)

[MOVE 11](#_Toc296991185)

[INSTALL 12](#_Toc296991186)

[COMMENT 12](#_Toc296991187)

[ACTIVITY 12](#_Toc296991188)

[Reagents 12](#_Toc296991189)

[Run 13](#_Toc296991190)

[Run Sequence 14](#_Toc296991191)

[Manual Run 15](#_Toc296991192)

[Thoughts 18](#_Toc296991193)

# Overview

This document describes the internal interface between the web server and the core Python server. All of the functions take the name of the current user as a parameter. Many of these functions return a simple value indicating success or failure which can be used to inform the user if something went wrong. A user with sufficient privileges can then view the system logs to obtain additional error information in the event of a failure.

# Configuration

**GetConfiguration()** – Returns details of the Elixys system.

Parameters:

* Username – Name of the current user.

Returns:

* Name – Name of this Elixys system (e.g. “Mini cell 3”).
* Version – System version (e.g. “2.0”).
* Debug – Boolean values that specifies if the client will display additional debug information to the user.

**GetSupportedOperations()** – Returns an array of operations supported by this system.

Parameters:

* Username – Name of the current user.

Returns:

* Supported operations – Array of operations supported by this system (e.g. “Add”, “Evaporate”, etc.).

**GetUserAccessLevels()** – Returns the user access levels recognized by this system.

Parameters:

* Username – Name of the current user.

Returns:

* User access level strings – Array of valid user access level strings (e.g. “Administrator”, “Tech”).

# User

**GetUser()** – Returns details of the given user.

Parameters:

* Username – Name of the current user.

Returns:

* User access string – String describing the user access level (e.g. “Administrator”).
* User access value – Word describing the permissions of the user’s access level. This value will be the same for all users with the same access level. Each bit of this string corresponds to a given action, e.g. create/edit sequence, run sequence, create/edit/delete users, etc.

**SaveUser()** – Creates a new user in the system or updates an existing user.

Parameters:

* Username – Name of the current user. If this name corresponds to an existing user then that user will be updated, otherwise a new user will be created.
* Password – MD5 hash of the user’s password if (1) this is a new user or (2) the user’s password is being changed. This field can be blank otherwise.
* User access string – String describing the user’s desired access level if (1) this is a new user or (2) the user’s access level is being changed. This field can be blank otherwise. This string must match a predefined user access level as returned by **GetUserAccessLevels()**.

Returns:

* Result – Boolean value (true on success, false otherwise).

**DeleteUser()** – Deletes the specified user from the system.

Parameters:

* Username – Name of the current user.

Returns:

* Result – Boolean value (true on success, false otherwise).

# Client State

**GetClientState()** – Returns the state of the client.

Parameters:

* Username – Name of the current user.

Returns:

* Client state – String describing the state of the client (e.g. “HOME”). This string may be delimited and contain state-specific information that will be understood by the web server (e.g. “VIEWSEQUENCE.14.52”) but should be treated by the core server as just a string.

**SaveClientState()** – Saves the client state to the database.

Parameters:

* Username – Name of the current user.
* Client state – String describing the state of the client.

Returns:

* Result – Boolean value (true on success, false otherwise).

# Server State

**GetServerState()** – Returns the state of the server.

Parameters:

* Username – Name of the current user.

Returns:

* Run status:
  + Mode – Run mode. Return values include “idle”, “runsequence” and “manualrun”.
  + Sequence ID – ID of the currently running sequence.
  + Username – Name of the user that is operating the system.
  + Status – String describing the current system status (e.g. “Reacting, 8:23 minutes”).
  + Active reactor – The active reactor number.
* Hardware state:
  + Pressure regulators – Details of each pressure regulator in the system:
    - Name – String describing the pressure regulator (e.g. “Main value pressure”).
    - Set pressure – The target pressure in millimeters of mercury.
    - Actual pressure – The actual pressure in millimeters of mercury.
  + Cooling – Boolean values that specifies if the cooling system is on.
  + Vacuum – Boolean value that specifies if the vacuum system is on.
  + Door – Boolean value that specifies if the main door is open.
  + Reagent Robot – Details of the reagent robot:
    - Position – Descriptive string of the robot position.
    - Raw X – Gives the raw X position of the robot in millimeters.
    - Raw Y – Gives the raw Y position of the robot in millimeters.
    - Actuator – String that specifies the state of the actuator. Possible values are “up”, “down” and “indeterminate”.
    - Gripper – Boolean value that specifies if the robot gripper is closed.
  + Reactors – Details of each reactor:
    - Number – The reactor number.
    - Set temperature – The set temperature of the reactor in degrees Celsius.
    - Actual temperature – The actual temperature of the reactor in degrees Celsius.
    - Position – The reactor position.
    - Vial – The vial state. Possible values are “up”, “down” and “indeterminate”.
    - Activity – The last know radiation activity level of the active reactor in millicuries.
    - Activity time – The time the activity was last measured.
    - Evaporation valves – Value that specifies if the evaporation values (nitrogen and vacuum) are open. Possible values are “open” and “closed”.
    - Transfer valve – Value that specifies the state of the transfer valve.
    - Reagent 1 transfer valve – Value that specifies the state of the first reagent transfer valve.
    - Reagent 2 transfer valve – Value that specifies the state of the second reagent transfer valve.
    - Stopcock 1 valve – Value that specifies the state of the first stopcock valve.
    - Stopcock 2 valve – Value that specifies the state of the second stopcock valve.
    - Stopcock 3 valve – Value that specifies the state of the third stopcock valve.

# Sequences

**GetSequenceList()** – Returns a list of sequences in the database:

Parameters:

* Username – Name of the current user.
* Type – String describing the type of sequence. Possible values are “Saved” and “Manual” for saved sequences and manual runs, respectively.

Returns:

* The following data are returned for each sequence:
  + Name – Sequence name.
  + Date – The date the sequence was created.
  + Time – The time the sequence was created.
  + Comment – Any comment associated with the sequence.
  + ID – Unique ID that is sent from the client to the server when the user selects the sequence.
  + Creator – User that created the sequence.
  + Operations – Number of operations.

**GetSequence()** – Returns details of a sequence and all components.

Parameters:

* Username – Name of the current user.
* Sequence ID – String that uniquely identifies the sequence.

Returns:

* Sequence metadata – Returns metadata for this sequence as described above in **GetSequenceList()**.
* Component information – Returns the following information for each sequence component:
  + Component Type – String that specifies the type of component. Possible values are the subheading under **Components** (e.g. “EVAPORATE”).
  + Component ID – Unique ID that is used by the client to refer to the component when communicating with the server.
  + Component Name – Display name of this component.
  + Reactor – The reactor associated with this component.
  + Reactor description – Description of the reactor field.
  + Reactor validation – Contains a string describing the reactor validation.
  + Additional details – Each component type contain additional information as documented below under **Components**.

**GetSequenceComponent()** – Returns details of a sequence and a single component.

Parameters:

* Username – Name of the current user.
* Sequence ID – String that uniquely identifies the sequence.
* Component ID – String that uniquely identifies a single component in the sequence.

Returns:

* Sequence metadata – Returns metadata for this sequence as described above in **GetSequenceList()**.
* Component information – Returns the information about the sequence component as described in **GetSequence()**.

**SaveSequence()** – Creates a new sequence in the system or updates an existing sequence’s metadata.

Parameters:

* Username – Name of the current user.
* Sequence ID – The ID of the sequence to update or blank to create a new sequence.
* Type – String describing the type of sequence. Possible values are “Saved” and “Manual” for saved sequences and manual runs, respectively.
* Sequence metadata – Metadata for the sequence that is being created or updated:
  + Name – Sequence name.
  + Comment – Any comment associated with the sequence.
  + Creator – User that created the sequence.

Returns:

* The ID of the newly created or updated sequence on success, blank otherwise.

**SaveSequenceComponent()** – Creates a new component in an existing sequence or updates an existing sequence component.

Parameters:

* Username – Name of the current user.
* Sequence ID – The ID of the sequence to associated with the component.
* Component information – Returns the following information for the sequence component:
  + Component Type – String that specifies the type of component. Possible values are the subheading under **Components** (e.g. “EVAPORATE”).
  + Component ID – The ID of the component to update or blank to create a new component.
  + Reactor – The reactor associated with this component.
  + Additional details – Each component type contain additional information as documented below under **Components**.

Returns:

* The ID of the newly created or update sequence component on success, blank otherwise.

**DeleteSequence()** – Deletes a sequence from the database.

Parameters:

* Username – Name of the current user.
* Sequence ID – The ID of the sequence to delete.

Returns:

* Result – Boolean value (true on success, false otherwise).

**DeleteSequenceComponent()** – Deletes a component from a sequence.

Parameters:

* Username – Name of the current user.
* Sequence ID – The ID of the sequence to associated with the component.
* Component ID – The ID of the component to delete.

Returns:

* Result – Boolean value (true on success, false otherwise).

**CopySequence()** – Duplicates a sequence. The source sequence may be either a saved sequence or a manual run. The sequence copy will always be a saved sequence.

Parameters:

* Username – Name of the current user.
* Sequence ID – The ID of the sequence to copy.
* Sequence metadata – Metadata of the new sequence:
  + Name – Sequence name.
  + Comment – Any comment associated with the sequence.
  + Creator – User that created the sequence.

# Components

Each component type has specific information associated with it in addition to the common information listed under **GetSequence()** and **SaveSequenceComponent()**. All of this information is writable by the latter unless explicitly indicated below as read-only.

## CASSETTE

The cassette component contains the following additional information:

* Available – Boolean value that indicates if this cassette is used in this synthesis.
* Reagents – Array of reagent IDs (read only).

## ADD

The add component contains the following additional information:

* Reagent – The reagent ID to add to the reactor.
* Reagent description – Description of the reagent field.
* Reagent validation – Contains a string describing the reagent validation.

## EVAPORATE

The evaporate component contains the following additional information:

* Duration – The length of the reaction.
* Duration description – Describes the duration field.
* Duration validation – Contains a string describing the reactor validation.
* Evaporation temperature – The evaporation temperature in Celsius.
* Evaporation temperature description – Describes the evaporation temperature field.
* Evaporation temperature validation – Contains a string describing the evaporation temperature validation.
* Final temperature – The final temperature in Celsius.
* Final temperature description – Describes the final temperature field.
* Final temperature validation – Contains a string describing the final temperature validation.
* Stir speed – The stir speed in rotations per minute.
* Stir speed description – Describes the stir speed field.
* Stir speed validation – Contains a string describing the stir speed field.

## TRANSFER

The transfer component contains the following additional information:

* Target – The target ID.
* Target description – Description of the target field.
* Target validation – Contains a string describing the target validation.

## ELUTE

The elute component contains the following additional information:

* Reagent – The reagent ID.
* Reagent description – Description of the reagent field.
* Reagent validation – Contains a string describing the reagent validation.
* Target – The target ID.
* Target description – Description of the target field.
* Target validation – Contains a string describing the target validation.

## REACT

The react component contains the following additional information:

* Position – The react position.
* Position description – Describes the position field.
* Position validation – Contains a string describing the position validation.
* Duration – The length of the reaction.
* Duration description – Describes the duration field.
* Duration validation – Contains a string describing the reactor validation.
* Reaction temperature – The reaction temperature in Celsius.
* Reaction temperature description – Describes the reaction temperature field.
* Reaction temperature validation – Contains a string describing the reaction temperature validation.
* Final temperature – The final temperature in Celsius.
* Final temperature description – Describes the final temperature field.
* Final temperature validation – Contains a string describing the final temperature validation.
* Stir speed – The stir speed in rotations per minute.
* Stir speed description – Describes the stir speed field.
* Stir speed validation – Contains a string describing the stir speed field.

## PROMPT

The prompt component contains the following additional information:

* Message – Text to display to the user.
* Message description – Contains a string describing the message field.
* Message validation – Contains a string describing the message validation.

## MOVE

The move component contains the following additional information:

* Position – The react position.
* Position description – Description of the position field.
* Position validation – Contains a string describing the position validation.
* State – Boolean value that specifies if the reactor is in the closed state.
* State description – Description of the state field.
* State validation – Contains a string describing the state validation.

## INSTALL

The install component contains the following additional information:

* Message – Text to display to the user.
* Message description – Contains a string describing the message field.
* Message validation – Contains a string describing the message validation.

## COMMENT

The comment component contains the following additional information:

* Comment – User-specified comment.
* Comment description – Description of the comment field.
* Comment validation – Contains a string describing the comment validation.

## ACTIVITY

The activity component contains no additional information.

# Reagents

**GetReagent()** – Returns details of a specific reagent.

Parameters:

* Username – Name of the current user.
* Sequence ID – Unique ID of the sequence associated with this reagent.
* Reagent ID – The unique ID of the reagent.

Returns:

* Used – Flag that indicates if this reagent position is used in this cassette.
* Position – The reagent position in the cassette.
* Name – The short name of the reagent.
* Description – The long description of the reagent.

**SaveReagent()** – Updates an existing reagent.

Parameters:

* Username – Name of the current user.
* Reagent ID – Unique ID that specifies the reagent to update.
* Component ID – The unique ID of the cassette associated with this reagent.
* Sequence ID – The unique ID of the sequence associated with this reagent.
* Used – Flag that indicates if this reagent position is used in this cassette.
* Position – The reagent position in the cassette.
* Name – The short name of the reagent.
* Description – The long description of the reagent.

Returns:

* Result – Boolean value (true on success, false otherwise).

# Run

**GetRunState()** – Returns the run state of the system.

Parameters:

* Username – Name of the current user.

Returns:

* Run state – String describing the state of the system. This string may be delimited and contain state-specific information that will be understood by the web server (e.g. “SEQUENCE.1.15”) but should be treated by the core server as just a string.

**SaveRunState()** – Updates the run state of the system.

Parameters:

* Username – Name of the current user.
* Run state – String describing the state of the system.

Returns:

* Result – Boolean value (true on success, false otherwise).

**GetRunUser()** – Returns the user currently running the system.

Parameters:

* Username – Name of the current user.

Returns:

* Run username – Name of the user currently operating the system or an empty string if the system is not running.

## Run Sequence

**RunSequence()** – Starts executing a sequence from the database.

Parameters:

* Username – Name of the current user.
* Sequence ID – Unique ID of the sequence to run.

Returns:

* Result – Boolean value (true on success, false otherwise).

**AbortRun()** – Aborts the run that is in progress.

Parameters:

* Username – Name of the current user.

Returns:

* Result – Boolean value (true on success, false otherwise).

**ContinueRun()** – Continues the run that has paused for a Prompt or Install unit operation.

Parameters:

* Username – Name of the current user.

Returns:

* Result – Boolean value (true on success, false otherwise).

## Manual Run

**StartManualRun()** – Starts a manual run and create a new manual run sequence in the database.

Parameters:

* Username – Name of the current user.

Returns:

* Sequence ID – Unique ID of the newly created manual run sequence or blank on error.

**PerformOperation()** – Performs a unit operation that has been added to the manual run sequence using **SaveSequenceComponent()**.

Parameters:

* Username – Name of the current user.
* Component ID – The unique ID of the sequence component. This unit operation will always be the last one in the sequence and will have just recently been added.
* Sequence ID – The unique ID of the manual run sequence that is associated with the component.

Returns:

* Result – Boolean value (true on success, false otherwise).

**AbortOperation()** – Abort the unit operation that is in progress.

Parameters:

* Username – Name of the current user.

Returns:

* Result – Boolean value (true on success, false otherwise).

**ContinueOperation()** – Continues the operation that has paused for a Prompt or Install unit operation.

Parameters:

* Username – Name of the current user.

Returns:

* Result – Boolean value (true on success, false otherwise).

**FinishManualRun()** – Completes the manual run and releases the lock on the Elixys system.

Parameters:

* Username – Name of the current user.

Returns:

* Result – Boolean value (true on success, false otherwise).