

# Table of Contents

<b>1 AdminAPI.....</b>	<b>1</b>
1.1 Introduction.....	1
1.2 PRINCIPLES.....	1
1.3 AdminXML API and Repositories.....	1
1.4 AdminXML API and User Syncs.....	1
1.5 USER ADMIN API.....	1
1.6 Valid User sub-elements and attributes.....	3
1.7 Responses.....	5
1.8 Error Handling.....	5
1.9 The Difference Between Admin XML and Agent XML.....	6
<b>2 Agent-XML.....</b>	<b>7</b>
2.1 Agent-XML Overview.....	7
2.2 Swivel requirements.....	7
2.3 Agent-XML.....	7
2.4 Principles.....	7
2.5 Structure.....	8
2.6 Commands.....	8
2.7 Agent-XML Errors.....	9
2.8 The Difference Between Admin XML and Agent XML.....	9
<b>3 API How to Guide.....</b>	<b>10</b>
3.1 Principles.....	10
3.2 Structure.....	10
3.3 Commands.....	11
3.4 Examples.....	11
<b>4 API Usage Examples.....</b>	<b>12</b>
4.1 Preparation.....	12
4.2 General Advice.....	12
4.3 The Swivel Client.....	12
4.4 Assign Tokens.....	13
4.5 Attribute Manager.....	13
4.6 Change PIN.....	13
4.7 Bulk Set PINs.....	13
<b>5 AuthenticationAPI.....</b>	<b>14</b>
5.1 Background.....	14
5.2 Starting a Session.....	14
5.3 Authentication.....	15
5.4 Change PIN.....	16
5.5 Security Strings.....	16
5.6 Mobile Client Provision (Version 3.8).....	16
5.7 Ping.....	17
5.8 Reset.....	17
5.9 User Exists.....	17
5.10 User Exists By Attribute.....	18
5.11 Token Synchronisation.....	18
5.12 Token Challenge-Response.....	18
5.13 Command Examples.....	18
5.14 Error and Warning Messages.....	19
<b>6 HelpdeskAPI.....</b>	<b>21</b>
6.1 Introduction.....	21
6.2 PRINCIPLES.....	21
6.3 Helpdesk API.....	21
6.4 Responses.....	22
6.5 Error Messages.....	23
<b>7 ReportingAPI.....</b>	<b>24</b>
7.1 Introduction.....	24
7.2 PRINCIPLES.....	24
7.3 REPORTING API.....	24
7.4 Responses.....	25
<b>8 Rng.....</b>	<b>26</b>
<b>9 Session Sharing.....</b>	<b>27</b>
<b>10 Introduction.....</b>	<b>28</b>
<b>11 Session Sharing Explained.....</b>	<b>29</b>
11.1 How it works.....	29
<b>12 Session Sharing Implementation.....</b>	<b>31</b>
12.1 Network Configuration.....	31
12.2 Firewalls Settings.....	31
12.3 Time Synchronisation.....	32
12.4 Swivel Settings.....	32
<b>13 Testing.....</b>	<b>33</b>
<b>14 Considerations.....</b>	<b>34</b>

# Table of Contents

<b>15 Troubleshooting.....</b>	<b>35</b>
<b>16 Swivel Combined Client.....</b>	<b>36</b>
16.1 Overview.....	36
16.2 Prerequisites.....	36
16.3 Using the Client.....	36
<b>17 Category:Symptom.....</b>	<b>38</b>
<b>18 Category:Whitepaper.....</b>	<b>39</b>

# 1 AdminAPI

## 1.1 Introduction

Swivel has developed an API, the **User Admin API**, to allow an external application to perform CRUD (Create, Read, Update, Delete) style operations on users. For most installations, these operations are performed by the User Synchronization job but for larger user populations synchronization may be impractical. The User Admin API addresses this need. In conjunction with the User Admin API, Swivel has also developed another API allowing external applications to perform the functions usually performed by a helpdesk operator from the admin console. These are typically day to day functions not viewed as part of the User Admin API such as user unlock, PIN reset etc. This is the [Helpdesk API](#)

There is additionally read-only functionality provided by the [Reporting API](#) that can be used to read the status (eg idle, locked) of user accounts.

## 1.2 PRINCIPLES

Following the principles behind the existing Agent API used for authorisation, both of the API's are based around XML documents for both request and reply. The basic idea is to build a document containing details of operation(s) to be performed and the user(s) on which they are to be performed, submit it to PINsafe via HTTP and PINsafe will reply with a document detailing which operations succeeded and which, if any, failed. All operations via the API will be logged in the standard PINsafe log. Only authorised PINsafe Agents will be able to submit requests. This is in line with Agents used for authorisation. The API is case sensitive, the correct case shown in the included examples.

In order to use the Admin API, you must create a PINsafe Agent for the computer, or sub-net of computers, that will execute AdminAPI commands. Only the computer(s) defined by this Agent can create, read, update or delete users belonging to this Agent, and the Agent cannot read, update or delete users created by other repositories. The Agent can therefore be regarded as another type of repository. It is possible, but not recommended, to manipulate users created by a normal (e.g. XML or Active Directory) repository, by giving the Agent exactly the same name as the repository. Be aware if you do this, however, that any subsequent User Sync of the repository will potentially overwrite modifications made by the Agent. At present, any PINsafe Agent can act as an AdminAPI repository, but future versions of PINsafe will require you to explicitly enable the "Act as Repository" property of an Agent.

These restrictions do not apply to HelpdeskAPI Agents. Although only Agents can use these commands, it is possible to specify that the HelpdeskAPI command operates on a different repository, or on all repositories.

### 1.2.1 Sending an Admin API request

An Admin API request must be sent as an HTTP(S) request to the pinsafe application, either as a GET or POST request.

For a GET request, the format is

```
https://[swivelserver]:8080/pinsafe/AdminXML?xml=[request body]
```

For a POST request, the URL is

```
https://[swivelserver]:8080/pinsafe/AdminXML
```

and the XML request should form the content of the POST.

## 1.3 AdminXML API and Repositories

Users created through the API are assigned to a repository named after the Agent from which the Swivel server receives AdminXML commands. Only users belonging to this repository can be managed by AdminXML commands.

Swivel must create a repository for each Agent. The differences in how the repositories appear vary with Swivel version as follows:

3.9 onwards There is an option on each Agent to enable it to act as a repository and this appear in the User Administration Repository drop-down.

3.8 Only Agents that actually have created users appear in the User Administration Repository drop-down.

3.4 to 3.7 All Agents are listed in the User Administration Repository drop-down.

## 1.4 AdminXML API and User Syncs

Repositories managed through the AdminXML are usually entirely managed through the AdminXML without user syncs against external user repositories. If changes are made through the API, they may be overwritten by a user sync with a repository, such as a user being added through the API will be deleted if a user sync is made if that user is not present in the repository.

### 1.4.1 AdminXML workaround for standard repositories

If you need to use the AdminXML API on other repositories, there is a workaround by defining an Agent with exactly the same name as an existing Repository (case-sensitive). However, this is not recommended for write and edit commands, as any changes made using the API will be overwritten the next time a User Sync runs.

Read commands are safe, however, as they do not change the data. If you need to do this, but you already have an Agent that you do not want to rename, you can define multiple Agents on the same IP address, as long as the secret is different.

## 1.5 USER ADMIN API

An example User Admin request is show below; Note the AdminRequest version number should match the version number of the Swivel server

```
<?xml version="1.0" ?>
<AdminRequest secret="MyAdminAgent" version="3.4">
```

```

<Create>
  <User name="bob">
    <Credentials pin="1234"/>
    <Groups>
      <Group name="EmailUsers"/>
    </Groups>
    <Policy changePin="true"/>
    <Rights dual="true" single="true"/>
    <Attributes>
      <Attribute name="email" value="bob@home"/>
    </Attributes>
  </User>
</Create>
</AdminRequest>

```

The above example shows a request to create a new user called ?bob? and his associated account details. Bob is a dual and single channel user with a pin of ?1234? that he must change next time he authenticates.

The AdminRequest element contains a ?secret? attribute, here set to ?MyAdminAgent?, which has the same function as it does in the Agent API ? when coupled with the source IP address it proves that the Agent is authorised to access Swivel.

The version attribute is required, but is only checked to ensure the value is not greater than the maximum allowed. For software versions up to and including 4.0.5, the maximum allowed version is "3.97", but there is no problem if the version is less than the target version, even if it uses features not available in earlier versions. It is essential that version is a valid number, though, so it must be "3.97" rather than "3.9.7".

This request would be POSTed and the response received via HTTP to the AdminXML servlet on the Swivel server, usually to be found here : <http://<ip.address.goes.here>:8080/pinsafe/AdminXML>

The Create operation is one of several possible operations :

- Create ? create a new user.
- Delete ? delete a user.
- Read ? read details of an existing user.
- Reset ? reset user authentication credentials (Note: the user must have an alert transport to receive the new PIN).
- Sync - Synchronise Swivel with a repository
- Update ? update user details.

All of these operations can take a list of users so multiple users can be supplied in a single request and there are no specific ordering requirements although, obviously, users must have been Created before operations can be performed on them.

A breakdown of the individual operations follows.

### 1.5.1 Create

Create a new user. Supply any or all relevant details for the user. Refer to the section on sub-elements to see what can be included within the User element

```

<Create>
  <User name="bob">
    <Credentials password=?itsasecret? pin="1234"/>
    <Attributes>
      <Attribute name="email" value="bob@home.com"/>
    </Attributes>
    <Groups>
      <Group name="EmailUsers"/>
    </Groups>
    <Policy changePin="true"/>
    <Rights dual="true" single="true"/>
    <String name="SMTP" destination="bob@home"/>
  </User>
</Create>

```

### 1.5.2 Delete

Delete an existing user. Supply only the username, nothing else is required or allowed.

```

<Delete>
  <User name="bob"/>
</Delete>

```

### 1.5.3 Read

Read user details. Supply only the username, nothing else is required or allowed.

```

<Read>
  <User name="bob"/>
</Read>

```

If the user exists the response will include the all the users details setable by the API, apart from their credentials.

### 1.5.4 Reset

Reset user credentials. Supply only the username, nothing else is required or allowed. This creates new credentials for the user and sends them out via their alert transport, if they have one configured. This is the equivalent of pressing the RESEND button on the user-administration screen.

```

<Reset>
  <User name="bob"/>

```

```
</Reset>
```

### 1.5.5 Sync

This can be used to synchronise Swivel with a repository. This can be used in scenarios where an external application populates a repository and then needs to action changes made within that repository.

The element needs to include a Repository element with the the name of the repository, as named on the Swivel server.

For example

```
<Sync>
<Repository repository="ActiveDirectoryOne"/>
</Sync>
```

### 1.5.6 Update

Update user details. Supply any elements you wish to update. Refer to the section on sub-elements to see what can be included within the User element

```
<Update>
  <User name="bob">
    <Attributes>
      <Attribute name="email" value="bob@home"/>
    </Attributes>
  </User>
</Update>
```

### 1.5.7 Purge

Purge deleted users from the repository:

```
<PurgeDeleted />
```

No content is permitted.

### 1.5.8 Message

Send an arbitrary message to a user. The message can be sent using either the user's alert transport (more usually) or their security strings transport.

```
<Message>
  <User name="bob">
    <Alert text="message" />
  </User>
</Message>
```

## 1.6 Valid User sub-elements and attributes

The user sub-element defines attributes for a new user when within a Create element or new attributes for an existing user when within an Update element Valid sub-elements and their permissible attributes are as follows.

### 1.6.1 Attributes

This element defines custom attributes for the user. These can be used as message destinations (e.g. email and phone), alternative identifiers when logging in, for searching the user list, or simply as additional information.

The Attributes element should contain 1 or more Attribute elements. Each of these must have a name and value attribute. The name must match the name of an attribute defined in the Repository -> Attributes configuration. The Attribute names should be used directly when using the API: there is no need to map Attributes to local names as with Active Directory / LDAP.

example

```
<Update>
  <User name="bob">
    <Attributes>
      <Attribute name="phone" value="447817360285"/>
      <Attribute name="email" destination="bob@home"/>
    </Attributes>
  </User>
</Update>
```

### 1.6.2 Alert & String

**Deprecated as of 3.9.6**

In 3.9.6 and later there is no need to explicitly set transport and alert transports and destinations. This will be determined by setting attributes for the user and by allocating the user to a group configured to use the required transport.

Alert and String refer to the Alert transport and String transport to be used by the user for sending system alerts (Alert) and dual-channel security strings (String).

A name must be supplied and this must match the name on a transport defined on the Swivel server. The destination must be appropriate to that transport

name Required, the Alert or String transport name. destination Required, the destination attribute required by the transport.

eg

```
<Update>
  <User name="bob">
    <Strings name="AQL" destination="447817360285"/>
    <Alert name="SMTP" destination="bob@home"/>
  </User>
</Update>
```

### 1.6.3 Credentials

Used to set PIN, Password or both. As agents are trusted, the credentials are not tested for conformance to PIN policies.

password Optional, the new password. pin Optional, the new PIN.

```
<Update>
  <User name="bob">
    <Credentials password=?itsasecret? pin="1234"/>
  </User>
</Update>
```

### 1.6.4 Groups

Groups element is used to specify to what group a user is a member. Group membership is used to determine certain rights within Swivel, for example what authentication agents and NASs via which they can authenticate. Group membership can be specified by using a <Groups> element contains a <Group> element for each group the user is a member.

All group must be included in all updates/creates

```
<Update>
  <User name="bob">
    <Groups>
      <Group name="DualChannelUsers"/>
      <Group name="EmailUsers"/>
      <Group name="AQLUsers"/>
    </Groups>
  </User>
</Update>
```

### 1.6.5 Policy

There are a number of policies that can be set within a policy element. These relate to the individual and overrule any overall server policies. To apply the policy set the policy to true, to remove the policy set to false

changePin Optional, user must change PIN next authentication

disabled Optional, user account is disabled

locked Optional, user account is locked (until 4.1)

lockedByAdmin Optional, user account is locked by administrator (4.2 onwards)

deleted Optional, user account is marked as deleted

inactive Optional, user account is inactive

lockedPinExpired Optional, user account is locked because the PIN has expired (4.2 onwards)

lockedFailures Optional, user account is locked because of too many authentication failures (4.2 onwards)

pinNeverExpires Optional, user PIN never expires

eg to unlock bob

```
<Update>
  <User name="bob">
    <Policy locked="false" />
  </User>
</Update>
```

NOTE: "locked" does not work in version 4.2: use "lockedByAdmin" instead. The "locked" attribute will be reinstated as a synonym for "lockedByAdmin" in future releases.

### 1.6.6 Rights

This element can be used to allocate the user rights within Swivel. **Note** that a user cannot be given Administrator rights via this interface

- dual Optional, user can use dual channel.
- helpdesk Optional, user is a helpdesk operator.
- pinless Optional, user is PINless.
- single Optional, user can use single channel.
- swivlet Optional, user can use the Mobile Phone Client or Swivlet.

Rights are added by setting the attribute to true and removed by setting to false.

eg to allow bob to use single channel but to remove helpdesk rights.

```
<Update>
```

```

    <User name="bob">
    <Rights single="true" helpdesk="false"/>
  </User>
</Update>

```

## 1.6.7 Oath

This element can be used to assign a previously-imported token to a user.

for example

```

<Update>
  <User name="bob">
    <Oath SerialNumber="12345678" />
  </User>
</Update>

```

## 1.7 Responses

Swivel responses to requests are similar in format. They reflect the command that has been submitted with a FAIL result should any of the commands not been executed.

For example:

```

<?xml version="1.0"? >
<AdminResponse>
  <Create>
    <User name="ann"/>
  </Create>
  <Read>
    <User name="ann">
      <Alert/>
      <Credentials/>
      <Groups/>
      <Policy pinNeverExpires="true" disabled="true"/>
      <Rights dual="true" single="true"/>
      <String/>
    </User>
  </Read>
</AdminResponse>

```

The above shows a response to an AdminRequest where user ?ann? was successfully created and read back.

Successful Create or Update sub-elements (Alert, Credentials?) are not returned, so we can not see from the response which were supplied but they all succeeded.

Obviously the account is only partially populated as ann has no groups, alert or string transport. Any elements that failed would have been returned containing the error ?FAIL? and the cause of the failure logged in the Swivel log. Credentials are never returned for security reasons.

```

<?xml version="1.0"?>
<AdminResponse>
  <Create>
    <User name="sid">FAIL</User>
  </Create>
</AdminResponse>

```

The above response shows a failure to create user ?sid? presumably because the account already exists. The full error will be shown in the Swivel server log.

```

<?xml version="1.0"?>
<HelpdeskResponse>
  <Update>
    <User name="ann"/>
  </Update>
</HelpdeskResponse>

```

The above is a Helpdesk response to a successful update.

## 1.8 Error Handling

### 1.8.1 PARSE ERRORS

Parse errors prevent execution of the request. Parse errors are typically caused by incorrect document structure and missing or invalid attributes; they indicate an application fault. In the event of a parse error, your reply will be a ParseError:

```

<ParseError>
  <Result>FAIL</Result>
  <Error>ADMIN_ERROR_UNSUPPORTED_ATTRIBUTE</Error>
</ParseError>

```

A list of errors can be found later in this document.

### 1.8.2 EXECUTION ERRORS

Execution errors can occur if a request was successfully parsed but part of the execution failed for some reason:

```

<HelpdeskResponse>

  <Reset>
    <User name="bob">FAIL</User>
  </Reset>

</HelpdeskResponse>

```

Execution errors can occur due to application faults i.e. trying to create a user that already exists or trying to send security strings to a user who has no transport set or doesn't exist.

They can also occur if a more serious database fault occurs. In both cases, a FAIL is returned for the element in question and the real cause of the error logged in the Swivel log.

### 1.8.3 POSSIBLE PARSE ERRORS

Error	Meaning
ADMIN_ERROR_DOCUMENT_MALFORMED	The document supplied, while possibly valid XML, wasn't a valid API request.
ADMIN_ERROR_INVALID_START_DATE	An invalid start date was supplied for a report.
ADMIN_ERROR_MISSING_DESTINATION	The destination attribute is required for Transports.
ADMIN_ERROR_MISSING_NAME	The name attribute was missing.
ADMIN_ERROR_MISSING_START_DATE	The start date was missing for a report.
ADMIN_ERROR_UNKNOWN_REPOSITORY	The supplied repository doesn't exist.
ADMIN_ERROR_UNSUPPORTED_ATTRIBUTE	An unsupported attribute was found.
ADMIN_ERROR_UNSUPPORTED_VERSION	This should be no higher than the API version number supported by the target server. Up to and including software version 4.0.5 the maximum value is "3.97".
ADMIN_ERROR_XML	General server side failure; consult the Swivel log for more details.
AGENT_ERROR_UNAUTHORIZED	

The agent is not authorized to access Swivel

## 1.9 The Difference Between Admin XML and Agent XML

Admin XML is used when a program needs to make changes to the user database, for example, to add and remove users, or change their details.

Agent XML is used when a program needs to authenticate users to Swivel. It also has functions for changing PIN and other features, but does not have the ability to change user details in the Swivel database.

Both APIs can be used from the same program, and they both require that an Agent is configured on the Swivel server. You can use the same Agent for both APIs, but to use the Admin API to manage users, the Agent must have "Can act as Repository" set to "Yes".



## 2 Agent-XML

### 2.1 Agent-XML Overview

Agent XML is used when a program needs to authenticate users to Swivel. It also has functions for changing PIN and other features, but does not have the ability to change user details in the Swivel database.

Swivel can authenticate users with 2 different protocols.

- RADIUS is a standards-based protocol, for further information see [RADIUS How To Guide](#)
- AgentXML is a proprietary Swivel standard. Agent-XML allows greater control as it also allows Reporting, Admin and Helpdesk functionality.

### 2.2 Swivel requirements

#### 2.2.1 Configuring Swivel to use Agent-XML Requests

Swivel must be configured to allow an agent to communicate, details on how to permit an agent to communicate with the Swivel server are here: [Agents How to Guide](#)

#### 2.2.2 Repository and Agent configuration

AdminRequest only works with users in the repository with the same name as the corresponding Agent. The work around is to give the Repository exactly the same name as the Agent (or vice versa). From version 3.9.2 Helpdesk requests are allowed for agents that are not repositories.

If you use HelpdeskRequest instead, you can specify a repository attribute on the Read element, but AdminRequest doesn't support this.

The reason for the distinction is that the AdminXML API was designed to support creation and manipulation of users from outside Swivel. The ability to access users from other repositories was an added feature, hence the use of HelpdeskRequest, where the commands are largely read-only.

### 2.3 Agent-XML

The Agent-XML API is an XML based API used for integrating Swivel with other applications.

There are 4 subsets of the API, that cover the following areas

- [Authentication](#)

Authentication, Change PIN, PIN Reset, Start Authentication Session, Request security Strings etc

- [Admin functions](#)

Add new users, set user details (eg mobile phone number), synchronise a repository, delete users

- [Helpdesk functions](#)

Unlock user, send user new credentials, set user policies etc

- [Reporting functions](#)

Retrieving lists of idle users etc

All these APIs follow the structure described in this article.

### 2.4 Principles

The APIs are based around XML documents for both request and reply.

The basic idea is to build a document containing details of operation(s) to be performed and the user(s) on which they are to be performed, submit it to Swivel via HTTP and Swivel will reply with a document detailing which operations succeeded and which, if any, failed. All operations via the API will be logged in the standard Swivel log.

Only authorised Swivel Agents will be able to submit requests. This is in line with Agents used for authorisation. The API is case sensitive, the correct case shown in the included examples.

Requests are sent to Swivel via an HTTP POST using http or https depending on the configuration of the Swivel server. For appliances the default will be https over port 8080.

The requests are posted to the Swivel context followed by AgentXML for the authentication API and AdminXML for the Admin, Helpdesk and reporting API.

For example `http://<ip address>:8080/pinsafe/AgentXML?xml=<?xml version="1.0"?>`

or

`https://<ip address>:8080/pinsafe/AdminXML?xml=<?xml version="1.0"?>`

## 2.5 Structure

The structure of a request that all API requests are contained within a request element. This request element specifies the type of request and also includes the shared secret for agent authentication.

e.g. an authentication request would be contained within an SASRequest tag

```
<?xml version="1.0" ?>
<SASRequest secret="MyAdminAgent" version="3.4">
  .
  .
</SASRequest>
```

Whereas an admin API request would be contained within an AdminRequest element.

```
<?xml version="1.0" ?>
<AdminRequest secret="MyAdminAgent" version="3.4">
  .
  .
</AdminRequest>
```

The responses from the Swivel server are similarly contained within response tags eg

```
<?xml version="1.0" ?>
<SASResponse secret="MyAdminAgent" version="3.4">
  .
  .
</SASResponse>
```

and

```
<?xml version="1.0" ?>
<AdminResponse secret="MyAdminAgent" version="3.4">
  .
  .
</AdminResponse>
```

Within these elements for elements that specify the particular request. For details of these elements refer to the article relating to that part of the API.

## 2.6 Commands

**Authentication** for more information see: [Authentication](#)

The following are valid values for the <Action> element:

- changepin : Change Users Credentials
- changepassword : Change a user password (Sentry or repository)
- checkpassword : Check that a password is correct (Sentry or repository)
- exists : Check if a user have an account.
- existsbyattribute : Check if a user has an account based on custom attributes
- increaselock : Increase a user's lock count (simulate a login failure)
- getuserattribute : Return the value of a named attribute for a user
- getusernamebyattribute : Return the username given an attribute name and value
- hascachedpassword : Check if a user has an AuthControl Desktop password cached
- killsession : terminate a Sentry session
- logevent : add a message to the Sentry logs (available from 4.2.1 onwards)
- login : Perform Sentry authentication
- loginbyattribute : Perform Sentry authentication using an alternative username attribute
- oathsync : resynchronise an OATH token
- ocraverify : Verify an OATH OCRA response to challenge
- ondemandmessage : send an On Demand message
- ping : Ping Swivel application
- provision : provision a mobile application given the provision code
- provisioncode : request a provision code
- reset : Perform a PIN reset (Note: the user must have a transport to receive the new PIN)
- resetcode : Request a PIN Reset code
- sendconfirmationcode
- sessionstart : Start an authentication session
- securitystrings : Request a set of security strings
- transportindex
- validateconfirmationcode

**Helpdesk** for more information see: [Helpdesk functions](#)

<Reset> : Send user new credentials (Note: the user must have a transport to receive the new PIN)

<Strings> : send dual channel security string(s) to user

<Update> : Change user policies or credentials

<OathSync> : Synchronise an OATH token

<PurgeDeleted> : Purge deleted users

**Admin** for more information see: [Admin functions](#)

<Create> : create a new user and optionally set user details.

<Delete> : delete a user.

<Read> : read the details of a given user.

<Reset> : Send user new credentials (Note: the user must have a transport to receive the new PIN)

<Sync> : Synchronise with a repository

<Update> : Change user policies, credentials, transports etc

<PurgeDeleted> : Purge deleted users.

**Reporting** for more information see: [Reporting functions](#) and [Reporting Using Agent-XML How to Guide](#)

<disabled> : Report on disabled users

<idle> : Report on idle users

<locked> : Report on locked users

## 2.7 Agent-XML Errors

**AgentXML request failed, error: The agent is not authorised to access the server.**

An Agent-XML request is being made against the Swivel server but is not permitted to do so. If access should be allowed create an entry on the Swivel Administration Console under Server/Agents. If an entry exists verified the shared secret is the same on Swivel and the access device. See also [Agents How to Guide](#). Try with the IP address instead of hostname, the hostname entry may be case sensitive

**AgentXML request failed, error: The XML request sent from the agent was malformed.**

The Agent XML request contains an error check the format. Spaces may also cause errors.

### ADMIN\_ERROR\_UNSUPPORTED\_VERSION

The agent version is incorrect.

<https://127.0.0.1:8080/pinsafe/AdminXML?xml=<AdminRequest%20secret='secret'%20version='3.7'><Report%20repository='local'><Idle%20since='01-jul-20>

Swivel 3.7 should use 3.6 for the version.

<https://127.0.0.1:8080/pinsafe/AdminXML?xml=<AdminRequest%20secret='secret'%20version='3.4'><Report%20repository='local'><Idle%20since='01-jul-20>

Agent XML Versions

Swivel Version	Agent-XML Version
3.7	3.6
3.6	3.6
3.5	3.4
3.4	3.4

### Malformed XML

This can occur when a ../AgentXML URL request is used to send an Admin request. Use the ../AdminXML instead.

## 2.8 The Difference Between Admin XML and Agent XML

Admin XML is used when a program needs to make changes to the user database, for example, to add and remove users, or change their details.

Agent XML is used when a program needs to authenticate users to Swivel. It also has functions for changing PIN and other features, but does not have the ability to change user details in the Swivel database.

Both APIs can be used from the same program, and they both require that an Agent is configured on the Swivel server. You can use the same Agent for both APIs, but to use the Admin API to manage users, the Agent must have "Can act as Repository" set to "Yes".

## 3 API How to Guide

The Agent-XML API is an XML based API used for integrating PINsafe with other applications.

There are 4 subsets of the API, that cover the following areas

- **Authentication**

Authentication, Change PIN, PIN Reset, Start Authentication Session, Request security Strings etc

- **Admin functions**

Add new users, set user details (eg mobile phone number), synchronise a repository, delete users

- **Helpdesk functions**

Unlock user, send user new credentials, set user policies etc

- **Reporting functions**

Retrieving lists of idle users etc

All these APIs follow the structure described in this article.

### 3.1 Principles

The APIs are based around XML documents for both request and reply.

The basic idea is to build a document containing details of operation(s) to be performed and the user(s) on which they are to be performed, submit it to PINsafe via HTTP and PINsafe will reply with a document detailing which operations succeeded and which, if any, failed. All operations via the API will be logged in the standard PINsafe log.

Only authorised PINsafe Agents will be able to submit requests. This is in line with Agents used for authorisation. The API is case sensitive, the correct case shown in the included examples.

Requests are sent to PINsafe via an HTTP POST using http or https depending on the configuration of the PINsafe server. For appliances the default will be https over port 8080.

The requests are posted to the pinsafe context followed by AgentXML for the authentication API and AdminXML for the Admin, Helpdesk and reporting API.

For example `http://<ip address>:8080/pinsafe/AgentXML` or `https://<ip address>:8080/pinsafe/AdminXML`

### 3.2 Structure

The structure of a request that all API requests are contained within a request element. This request element specifies the type of request and also includes the shared secret for agent authentication.

eg an authentication request would be contained within an SASRequest tag

```
<?xml version="1.0" ?>
<SASRequest secret="MyAdminAgent" version="3.4">
  .
  .
</SASRequest>
```

Whereas an admin API request would be contained within an AdminRequest element.

```
<?xml version="1.0" ?>
<AdminRequest secret="MyAdminAgent" version="3.4">
  .
  .
</AdminRequest>
```

...or else a HelpdeskRequest element.

```
<?xml version="1.0" ?>
<HelpdeskRequest secret="MyAdminAgent" version="3.4">
  .
  .
</HelpdeskRequest>
```

The syntax of AdminRequest and HelpdeskRequest are more or less identical. The major difference is that AdminRequest can make major changes to the database, such as adding and deleting users, whereas HelpdeskRequest is largely read-only, although it does support some changes to existing users, such as PIN change. Conversely, AdminRequest can only affect users in the repository with the same name as the Agent, so the Agent must have "Can Act as Repository" set to Yes. HelpdeskRequest can affect any users.

The responses from the PINsafe server are similarly contained within response tags eg

```
<?xml version="1.0" ?>
<SASResponse secret="MyAdminAgent" version="3.4">
  .
  .
</SASResponse>
```

```
</SASResponse>
```

and

```
<?xml version="1.0" ?>
<AdminResponse secret="MyAdminAgent" version="3.4">
.
</AdminResponse>
```

Within these elements for elements that specify the particular request. For details of these elements refer to the article relating to that part of the API.

## 3.3 Commands

### Authentication

<changePIN> : Change Users Credentials  
<exists> : Does a user have an account.  
<login> : Perform PINsafe authentication  
<ping> : Ping PINsafe application  
<reset> : Perform a PIN reset  
<resetcode> : Request a PIN Reset code  
<startsession> : Start an authentication session  
<securitystrings> : Request a set of security strings

### Helpdesk functions

<reset> : Send user new credentials  
<strings> : send dual channel security string(s) to user  
<update> : Change user policies or credentials

### Admin functions

<create> : create a new user and optionally set user details.  
<delete> : delete a user.  
<read> : read the details of a given user.  
<reset> : Send user new credentials  
<sync> : Synchronise with a repository  
<update> : Change user policies, credentials, transports etc

### Reporting functions

<disabled> : Report on disabled users  
<idle> : Report on idle users  
<locked> : Report on locked users

## 3.4 Examples

We have a number of applications that use these APIs, available as Windows executable programs. These can be found on a separate page.

## 4 API Usage Examples

This page describes a number of small applications that have been developed to help customers manage their users. These are Windows applications, and use the AdminXML API.

DISCLAIMER: although all of these programs have undergone some testing, they have been developed for specific requirements, so may not be suitable in all scenarios. Read the documentation before using.

### 4.1 Preparation

In order to use these applications, you must define an Agent in the Sentry administration console (Server -> Agents). The Hostname/IP must correspond to the address of the Windows PC that the applications will be running on. If you need to allow multiple addresses, you can use network masks in the Agent definitions, if appropriate. You can define multiple Agents in certain circumstances, but see the next paragraph.

#### 4.1.1 Special Considerations for AdminRequest

Where applications add or delete users, the corresponding Agent must be set to Act as a Repository. In this case, the name of the Agent defines the name of the Repository. This has two consequences:

1. You can only define one Agent for each Repository
2. If you need to manage users that are in an existing Repository, the Agent name must exactly match the Repository name

In the latter case, if the Repository is linked to an external source, such as Active Directory, be aware that synchronising with the Repository may undo changes made by the API (and vice versa).

### 4.2 General Advice

These applications are not provided with installers. Simply unzip all files (in most cases there are 3) into the same folder and run from the command line.

All these applications (with one exception) are written to be run from the command line, and require additional arguments on the command line.

They also come with a configuration file, named as the executable program but with ".config" on the end. Before you use the application for the first time, you will need to edit this file and enter values appropriate for your environment. Below is an example of this file:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <sectionGroup name="userSettings" type="System.Configuration.UserSettingsGroup, System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
    <section name="AssignTokens.Settings" type="System.Configuration.ClientSettingsSection, System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  </configSections>
  <userSettings>
    <Properties.Settings>
      <setting name="PINsafeUrl" serializeAs="String">
        <value>https://primary.swiveldev.local:8080/sentry</value>
      </setting>
      <setting name="PINsafeSecret" serializeAs="String">
        <value>secret</value>
      </setting>
      <setting name="AllowSelfSignedSSL" serializeAs="String">
        <value>False</value>
      </setting>
      <setting name="Repository" serializeAs="String">
        <value>AD</value>
      </setting>
    </Properties.Settings>
  </userSettings>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6.1" />
  </startup>
</configuration>
```

You should only change the values within the <value>...</value> elements - do not change anything else.

The above settings exist in the majority of applications, and are described below.

**PINsafeUrl** : the full URL of the Sentry administration URL. Be aware that this must be the core Sentry application - these applications will not work when connected to Sentry SSO or Proxy on port 8443. Do not include the final part of the path: "/AdminXML" - this will be added by the application.

**PINsafeSecret** : the secret as included in the Agent definition

**AllowSelfSignedSSL** : True or False, depending on whether the connection should work if the Sentry certificate is self-signed, has expired, or the URL does not match the hostname in the certificate. "False" will allow the connection to work in all cases, but is less secure.

**Repository** : the name of the repository on which the application is to act. This is only required for applications that use the Helpdesk API: as described above, the Admin API requires that the Repository name matches the Agent name. Instructions for each application will state which type of request it uses. In most cases, you can simply specify "" to indicate that any repository can be used.

Where additional settings are required, or the settings names are different, this will be described under the individual applications.

### 4.3 The Swivel Client

[Download link.](#)

This is the client DLL that is required by all the applications on this page. A copy of it is provided in each download, so this stand-alone copy is provided for customers who wish to develop their own applications. It is written in C#, as are all the applications, so requires the Microsoft.Net framework to operate.

## 4.4 Assign Tokens

[Download link.](#)

This application allows you to assign OATH tokens to users, either individually or in bulk. It uses the AdminRequest, so the Agent must be set to Act as Repository.

Usage:

```
AssignTokens <username> <tokenid>
```

Assigns a single token with id <tokenid> to the user <username>

```
AssignTokens <filename>
```

Reads a list of users and tokens from the file <filename>.

Each line in the file should contain a username followed by a token serial ID.

The username and token ID can be separated by spaces, tabs or commas.

Both username and token must already exist in the database.

## 4.5 Attribute Manager

[Download link](#)

This is a form-based application. It allows you to view and alter certain custom attributes for users - specifically Email address, Phone number, and given and family names. The application uses AdminRequest, so the Agent must be set to Act as Repository. However, it can be used to read attributes from other repositories, but not write, so includes the Repository property.

Note that if the attributes are set to "Synchronised" under the Repository Attributes configuration, they will be overwritten by a user sync.

## 4.6 Change PIN

[Download link.](#)

This application simply allows you to change the PIN for a single user without knowing the current PIN. It uses HelpdeskRequest, so can be applied to users in any repository.

Usage:

```
ChangePIN username newPIN
```

## 4.7 Bulk Set PINs

[Download link.](#)

This application allows you to set the PIN for a list of users, either to the same initial PIN, or to individual PINs. It uses HelpdeskRequest, so can be applied to users in any repository.

Usage:

```
InitialPINs <pin> <filename>
```

Where <pin> is the PIN to use for all named users, or "?" to read PIN from the file, and <filename> is the name of a file containing a list of usernames.

If the PINs are in the file, they should be after the username, separated by a Tab character.

## 5 AuthenticationAPI

The Swivel Authentication API (Agent-XML) is a means by which external application can make authentication requests to Swivel.

The API is XML-Based and is a subset of the overall [Agent-XML](#) API.

### 5.1 Background

All [Agent-XML](#) requests include a shared secret (with the exception of ping). In order for an authentication API to be acted upon by Swivel, the IP addressed and shared secret presented within the request must match that configured on the Swivel server.

All requests also include a version number. This should be set according to the target version of Swivel:

- 3.1 for all versions of Swivel from 3.1 - 3.3.
- 3.4 for versions 3.4 and 3.5.
- 3.6 for versions 3.6 onwards.

(NOTE: the value of version is not actually checked, but the element must be present).

Authentication requests must be sent via an HTTP Post to the Swivel server, to the AgentXML context.

For example `http://<ip address>:8080/pinsafe/AgentXML`

Note that for appliances this will be https by default. Agent XML requests are sent direct to the Swivel application on port 8080 and not via the proxy.

There is a optional field called requestID. If this is included then it will be echoed back in the corresponding response.

```
<?xml version="1.0" ?>
<SASRequest>
<Version>3.6</Version>
<RequestID>1000</RequestID>
.
.
</SASRequest>

<?xml version="1.0" ?>
<SASResponse>
<Version>3.6</Version>
<RequestID>1000</RequestID>
.
.
</SASResponse>
```

### 5.2 Starting a Session

Some Swivel authentication modes are session based.

In this mode an Agent must start a session for a user and then request the security string for that session, either via an Image ([Single Channel How To Guide Single Channel](#), eg [TURING](#)) or a message( [Dual Channel](#), eg [SMS](#) message).

When an agent starts a session an ID for that session is returned. This session ID can then be used to request the string either via an image

`img src = http://pinsafe:8080/pinsafe/SCImage?sessionid=3bfwuefi37tr`

or via a message

`img src = http://pinsafe:8080/pinsafe/DCMessage?sessionid=3bfwuefi37tr`

The a session start message is as follows

The Agent gathers the user's username and sends the following XML Request to the Swivel Server:

The above XML Request will start a session on the Swivel Server, which will respond with the following XML:

```
<?xml version="1.0" ?>
<SASRequest>
<Version>3.6</Version>
<Secret>shared_secret</Secret>
<Action>sessionstart</Action>
<Username>some_user</Username>
</SASRequest>

<?xml version="1.0" ?>
<SASResponse>
<Version>3.6</Version>
<Result>PASS</Result>
<SessionID>c7379ef1b41f90a4900548a75e13f62a</SessionID>
</SASResponse>
```

If for any reason the session start fails a fail message will be return along with the reason for the failure, for example

```
<?xml version="1.0" ?>
<SASResponse>
<Version>3.6</Version>
<Result>FAIL</Result>
<Reason>AGENT_ERROR_NO_USER_FOUND</Reason>
</SASResponse>
```



A list of possible errors is shown in the Error Messages Section below.

## 5.3 Authentication

The following example shows how to perform a login using the Agent XML Interface. The login request is used for both Single Channel and Dual Channel logins.

The Agent gathers the user's username, optional password, One-Time-Code and sends the following XML Request to the Swivel Server:

```
<?xml version="1.0" ?>
<SASRequest>
<Version>3.6</Version>
<Secret>shared_secret</Secret>
<Action>login</Action>
<Username>some_user</Username>
<Password>password</Password>
<OTC>1234</OTC>
</SASRequest>
```

If the authentication request is successful the server will respond with the following XML:

```
<?xml version="1.0" ?>
<SASResponse>
<Version>3.6</Version>
<Result>PASS</Result>
</SASResponse>
```

If there are any problems the Result element will contain a FAIL and if an error has occurred there will be an Error element, for example:

```
<?xml version="1.0" ?>
<SASResponse>
<Version>3.6</Version>
<Result>FAIL</Result>
<Error> AGENT_ERROR_NO_SECURITY_STRINGS</Error>
</SASResponse>
```

If the Result element's value is FAIL and there is no Error value the user has simply failed to authenticate successfully, ie supplied incorrect credentials.

### 5.3.1 Authentication By Attribute

The standard login request must contain the users Swivel Username.

However some services may require another attribute to be used, eg email address. There is a separate API call to handle this case whereby the agent provides the attribute name and attribute value is part of the authentication request.

"Note that the Swivel Authentication Platform will need to be configured to allow other attributes to be used in this way"

```
<?xml version="1.0" ?>
<SASRequest>
<Version>3.6</Version>
<Secret>shared_secret</Secret>
<Action>login</Action>
<Username>some_user@domain.com</Username>
<Attribute>email</Attribute>
<Password>password</Password>
<OTC>1234</OTC>
</SASRequest>
```

### 5.3.2 Warnings

A successful authentication request can include warnings, these usually refer to the fact that a PIN needs to be changed due to a policy that is set or due to the fact that it will soon expire. These warnings can be used by the agent to re-direct the user to a change-PIN page after authentication.

```
<?xml version="1.0" ?>

<SASResponse>
<Version>3.6</Version>
<Result>PASS</Result>
<Warning>AGENT_WARN_CHANGE_PIN</Warning>
</SASResponse>
```

### 5.3.3 Check Password

from Version 3.10

It is possible to use Agent XML to check a user's password. This will either be the user's Swivel password or their repository password depending on the policy set for this agent. The attempt will be logged but an incorrect password will not be treated as a failed authentication attempt

```
<?xml version="1.0" ?>
<SASRequest>
<Version>3.6</Version>
<Secret>shared_secret</Secret>
<Action>checkpassword</Action>
<Username>some_user</Username>
<Password>password</Password>
</SASRequest>
```

Response is a simple pass or fail.

## 5.4 Change PIN

Change PIN is very similar to a login request with the inclusion of the user's new password and new PIN, sent has a new One-Time-Code. The Agent gathers the username, existing password, new password, existing One-Time-Code, and new One-Time-Code. To start the change PIN process the following XML is sent to the Swivel Server:

If no password is being used the <Password> and <NewPassword> elements should be empty.

```
<?xml version="1.0" ?>
<SASRequest>
<Version>3.6</Version>
<Secret>shared_secret</Secret>
<Action>changePin</Action>
<Username>some_user</Username>
<Password>password</Password>
<NewPassword>newpassword</NewPassword>
<OTC>1234</OTC>
<NewOTC>4321</NewOTC>
</SASRequest>
```

If the request is successful the Swivel Server will change the user's password and PIN, and will respond with a PASS in the Result element:

```
<?xml version="1.0" ?>
<SASResponse>
<Version>3.6</Version>
<Result>PASS</Result>
</SASResponse>
```

The change pin can fail for a number of reasons. If the password and one-time code submitted are incorrect the response will just indicate a failure. If the current credentials were correct, then the pin change may fail because the new PIN does not confirm to PIN composition policies, such as not allowing sequences such as 1234.

```
<?xml version="1.0" ?>

<SASResponse>
<Version>3.6</Version>
<Result>FAIL</Result>
<Error>AGENT_ERROR_PIN_COMPOSITION</Error>
</SASResponse>
```

## 5.5 Security Strings

This API is also used by a midlet to request a batch of security strings.

For Versions of Swivel prior to Version 3.8 the format is as follows

```
<?xml version="1.0" ?>
<SASRequest>
<Version>3.6</Version>
<Action>securitystrings</Action>
<Username>some_user</Username>
</SASRequest>
```

For Version 3.8 the format is

```
<?xml version="1.0" ?>
<SASRequest>
<Version>3.6</Version>
<Action>SecurityStrings</Action>
<Id>client-id</Id>
</SASRequest>
```

In this version the mobile client needs to provide a unique user-id in order to receive security strings. The mobile client obtains this client-id by completing the provision process described below.

The response is a set of 99 security strings

## 5.6 Mobile Client Provision (Version 3.8)

In order for a mobile client to download security strings it needs to obtain a unique client-id. This is obtained by submitting a provision code to Swivel. This provision code will be sent to the user, usually via a text message.

There is an API call that will request a provision code to be sent to the user. (Or this can be done via the admin console)

```
<?xml version="1.0" ?>
<SASRequest>
<Version>3.6</Version>
<Action>provisioncode</Action>
<Username>some_user</Username>
</SASRequest>
```

If successful a Pass packet will be returned and a provision code will be sent to the end-user.

If not successful a FAIL will be returned along with any error.

The user will then enter the provision code. The provision code will then be presented to Swivel by the client as

```
<?xml version="1.0" ?>
<SASRequest>
<Version>3.6</Version>
<Action>provision</Action>
<ProvisionCode>code</ProvisionCode>
<Username>username</Username>
```

```
</SASRequest>
```

If the code is correct Swivel will return a PASS along with the clients Unique Client ID (UCID)

```
<?xml version="1.0" ?>
<SASResponse>
<Version>3.6</Version>
<Result>PASS</Result>
<Id>id</Id>
</SASResponse>
```

If there is a problem, then an error will be returned

```
<?xml version="1.0" ?>
<SASResponse>
<Version>3.6</Version>
<Result>FAIL</Result>
<Error>AGENT_NO_SESSION</Error>
</SASResponse>
```

## 5.7 Ping

You can use a ping command to test that the Swivel application is available. The response is a pass response.

```
<?xml version="1.0" ?>
<SASRequest>
<Version>3.6</Version>
<Action>ping</Action>
</SASRequest>
```

The response being

```
<?xml version="1.0" ?>
<SASResponse>
<Version>3.6</Version>
<Result>PASS</Result>
</SASResponse>
```

## 5.8 Reset

If it is enabled on the Swivel server, a user can request a reset code to be sent to them, then of they enter that reset code, they are sent a new PIN (Note: the user must have a transport to receive the new PIN).

Two API commands support this, <Resetcode> that sends the code to the user and then <Reset> that submits the reset code to Swivel.

```
<?xml version="1.0" ?>
<SASRequest>
<Version>3.6</Version>
<Secret>shared_secret</Secret>
<Action>resetcode</Action>
<Username>some_user</Username>
</SASRequest>
```

```
<?xml version="1.0" ?>
<SASResponse>
<Version>3.6</Version>
<Result>PASS</Result>
</SASResponse>
```

This sends the reset code to the user. The user enters the code on a form on the agent and submits.

```
<?xml version="1.0" ?>
<SASRequest>
<Version>3.6</Version>
<Secret>shared_secret</Secret>
<Action>reset</Action>
<Username>some_user</Username>
<Resetcode>87456hfiu7634</Resetcode>
</SASRequest>
```

```
<?xml version="1.0" ?>
<SASResponse>
<Version>3.6</Version>
<Result>PASS</Result>
</SASResponse>
```

If the reset code is entered correctly then the user is sent a new set of credentials.

## 5.9 User Exists

This method can be used as a pre-authentication check to see if an account exist on Swivel.

```
<?xml version="1.0" ?>
<SASRequest>
<Version>3.6</Version>
<Secret>shared_secret</Secret>
<Action>exists</Action>
<Username>some_user</Username>
</SASRequest>
```

If the user exists a Pass is returned, if the user does not exist a Fail is sent back.

```
<?xml version="1.0" ?>
<SASResponse>
<Version>3.6</Version>
```

```
<Result>PASS</Result>
</SASResponse>
```

## 5.10 User Exists By Attribute

A variation of the above method allows the query to pass in an attribute, eg email address.

The response to the query will be the name of the attribute that matched if the user could be found

Or fail if not match (or multiple matches) were found

eg

```
<?xml version="1.0" ?>
<SASRequest>
  <Version>3.6</Version>
  <Secret>shared_secret</Secret>
  <Action>ExistsByAttribute</Action>
  <Username>firstname.lastname@domain.com</Username>
</SASRequest>

<?xml version="1.0" ?>
<SASResponse>
  <Version>3.6</Version>
  <Result>PASS</Result>
  <AttributeName>email</AttributeName>
</SASResponse>
```

## 5.11 Token Synchronisation

This API calls attempts to synchronise a user's token but submitting two consecutive OTPs

```
<SASRequest>
<Version>3.6</Version>
<Secret>shared_secret</Secret>
<Username>tokenuser</Username>
<Action>OathSync</Action>
<OTP1>481262</OTP1>
<OTP2>579024</OTP2>
</SASRequest>
```

If the request is successful a PASS message will be returned. If not the possible failures are SYNC\_FAILURE (meaning the OTPs were not valid) or OATH\_TOKEN\_NOT\_FOUND (meaning the user does not have a token).

## 5.12 Token Challenge-Response

This API call validates the users response to an OCRA challenge. (Where the user enters the challenge on the OCRA token keypad)

```
<SASRequest>
<Version>3.1</Version>
<Secret>shared_secret</Secret>
<Username>tokenuser</Username>
<Action>OcraVerify</Action>
<OcraChallenge>8765432</OcraChallenge>
<OcraResponse>12345678</OcraResponse>
</SASRequest>
```

If the response is verified a PASS message will be returned. If not an OCRA\_RESPONSE\_FAILURE error will be returned

## 5.13 Command Examples

Below is the example of a ping command

http://127.0.0.1:8080/pinsafe/AgentXML?xml=<?xml version="1.0"?><SASRequest><Version>3.1</Version><Action>ping</Action></SASRequest>

Expected output

```
<?xml version="1.0" ?>
- <SASResponse>
  <Version>3.6</Version>
  <RequestID />
  <Result>PASS</Result>
</SASResponse>
```

Below is the example of a session request command

http://127.0.0.1:8080/pinsafe/AgentXML?xml=<?xml version="1.0"?><SASRequest><Version>3.6</Version><Secret>secret</Secret><Action>sessionstart</Action><Username>graham</Username></SASRequest>

Expected output

```
<?xml version="1.0" ?>
- <SASResponse>
  <Version>3.6</Version>
  <RequestID />
  <Result>PASS</Result>
  <SessionID>f853792503f2e83f3ff55f693f631537</SessionID>
</SASResponse>
```

Swivel log

```
local:Session started for user: graham.
```

Below is the example of a login command

```
http://127.0.0.1:8080/pinsafe/AgentXML?xml=<?xml version="1.0"
?><SASRequest><Version>3.6</Version><Secret>secret</Secret><Action>login</Action><Username>graham</Username><Password></Password><OTC>8
```

Expected output

```
<?xml version="1.0" ?>
- <SASResponse>
  <Version>3.6</Version>
  <RequestID />
  <Result>PASS</Result>
  <Channel>SINGLE</Channel>
</SASResponse>
```

Swivel log

```
local:Login successful for user: graham.
```

Below is the example of a changepin command

```
http://127.0.0.1:8080/pinsafe/AgentXML?xml=<?xml
version="1.0"?><SASRequest><Version>3.6</Version><Secret>secret</Secret><Action>changepin</Action><Username>graham</Username><Password></P
```

Expected output

```
<?xml version="1.0" ?>
- <SASResponse>
  <Version>3.6</Version>
  <RequestID />
  <Result>PASS</Result>
  <Channel>SINGLE</Channel>
</SASResponse>
```

Swivel log

```
local:Change PIN successful for user: graham.
```

Below is the example of a resetcode (reset PIN) request command

```
http://127.0.0.1:8080/pinsafe/AgentXML?xml=<?xml
version="1.0"?><SASRequest><Version>3.6</Version><Secret>secret</Secret><Action>resetcode</Action><Username>graham</Username></SASRequest>
```

Expected output

```
<?xml version="1.0" ?>
- <SASResponse>
  <Version>3.6</Version>
  <RequestID />
  <Result>PASS</Result>
</SASResponse>
```

Swivel log

```
local:Self-reset code request successful for user: graham.
```

Below is the example of a reset (reset PIN) command

```
http://127.0.0.1:8080/pinsafe/AgentXML?xml=<?xml
version="1.0"?><SASRequest><Version>3.6</Version><Secret>secret</Secret><Action>reset</Action><Username>graham</Username><Resetcode>697501
```

Expected output

```
<?xml version="1.0" ?>
- <SASResponse>
  <Version>3.6</Version>
  <RequestID />
  <Result>PASS</Result>
</SASResponse>
```

Swivel log

```
local:Self-reset successful for user: graham.
```

## 5.14 Error and Warning Messages

Error	Meaning
AGENT_ERROR_AGENT_ACCESS	The user is not in the correct group to use this agent
AGENT_ERROR_ACTION_TYPE	The XML Request sent by the Agent did not contain an unrecognised Action element.
AGENT_ERROR_GENERAL	An unspecified error occurred.
AGENT_ERROR_NO_ACTION	The XML Request sent by the Agent did not contain an Action element.
AGENT_ERROR_NO_AUTH	Swivel does not know how to authenticate this user
AGENT_CANNOT_CHANGE_REPOSITORY_PASSWORD	A pin change failed as the two different passwords were submitted but the agent was configured to use repository passwords
AGENT_ERROR_NO_CHANGE	A pin change failed as the credentials submitted were the same
AGENT_ERROR_NO_PIN	The user has no PIN set

AGENT_ERROR_AUTH_METHOD_UNSUPPORTED	This agent cannot authenticate a user using this method, eg attempting a single channel authentication on a dual-channel only agent
AGENT_ERROR_NO_OTC	One-time code was missing or malformed
AGENT_ERROR_BAD_OTC	A Session could not be created by the Swivel server. Please try again at a later time.
AGENT_ERROR_SESSION	The Agent is not authorised to use the Swivel server.
AGENT_ERROR_UNAUTHORIZED	The Username element in the XML Request sent by the Agent contained invalid characters.
AGENT_ERROR_USERNAME	The XML Request sent by the Agent to the Swivel server was malformed.
AGENT_ERROR_XML	The user is required to change PIN before their next login
AGENT_WARN_CHANGE_PIN	The users PIN will shortly expire, ie within the period sepcified by the change pin warning on the Swivel server.
AGENT_WARN_PIN_EXPIRY	

# 6 HelpdeskAPI

## 6.1 Introduction

Swivel has developed an API, the **User Admin API**, to allow an external application to perform CRUD (Create, Read, Update, Delete) style operations on users. For most installations, these operations are performed by the User Synchronization job but for larger user populations synchronization may be impractical. The User Admin API addresses this need. In conjunction with the User Admin API, Swivel has also developed another API allowing external applications to perform the functions usually performed by a helpdesk operator from the admin console. These are typically day to day functions not viewed as part of the User Admin API such as user unlock, PIN reset etc. This is the **Helpdesk API**

There is additionally read-only functionality provided by the **Reporting API** that can be used to read the status (eg idle, locked) of user accounts.

## 6.2 PRINCIPLES

Following the principles behind the existing Agent API used for authorisation, both of the API's are based around XML documents for both request and reply. The basic idea is to build a document containing details of operation(s) to be performed and the user(s) on which they are to be performed, submit it to PINsafe via HTTP and PINsafe will reply with a document detailing which operations succeeded and which, if any, failed. All operations via the API will be logged in the standard PINsafe log. Only authorised PINsafe Agents will be able to submit requests. This is in line with Agents used for authorisation. The API is case sensitive, the correct case shown in the included examples.

In order to use the Admin API, you must create a PINsafe Agent for the computer, or sub-net of computers, that will execute AdminAPI commands. Only the computer(s) defined by this Agent can create, read, update or delete users belonging to this Agent, and the Agent cannot read, update or delete users created by other repositories. The Agent can therefore be regarded as another type of repository. It is possible, but not recommended, to manipulate users created by a normal (e.g. XML or Active Directory) repository, by giving the Agent exactly the same name as the repository. Be aware if you do this, however, that any subsequent User Sync of the repository will potentially overwrite modifications made by the Agent. At present, any PINsafe Agent can act as an AdminAPI repository, but future versions of PINsafe will require you to explicitly enable the "Act as Repository" property of an Agent.

These restrictions do not apply to HelpdeskAPI Agents. Although only Agents can use these commands, it is possible to specify that the HelpdeskAPI command operates on a different repository, or on all repositories.

### 6.2.1 Sending an Admin API request

An Admin API request must be sent as an HTTP(S) request to the pinsafe application, either as a GET or POST request.

For a GET request, the format is

```
https://[swivelserver]:8080/pinsafe/AdminXML?xml=[request body]
```

For a POST request, the URL is

```
https://[swivelserver]:8080/pinsafe/AdminXML
```

and the XML request should form the content of the POST.

## 6.3 Helpdesk API

An example Helpdesk request is shown below;

```
<?xml version="1.0" ?>
<HelpdeskRequest secret="MyHelpdeskAgent" version="3.4">
  <Reset repository=?repository1?>
    <User name="bob"/>
  </Reset>
</HelpdeskRequest>
```

The above example shows a request to reset bob's credentials and send them out. Obviously bob must have a valid alert transport defined. As with all Agent-XML requests, the HelpdeskRequest element contains a ?secret? attribute.

The Helpdesk API is more restrictive than the Admin API; only the following operations are supported :

- ♦ Reset ? reset user credentials.
- ♦ Strings ? send new security strings to the user.
- ♦ Update ? update user Credentials or Policy.

As with an AdminRequest, all of these operations can take a list of users so multiple users can be supplied in a single request and again there are no specific ordering requirements although as before, obviously, users must have been Created (possibly with an appropriate AdminRequest) before operations can be performed on them.

Also as with the AdminRequest, a HelpdeskRequest is posted to the AdminXML servlet and the response received in the same way. All Helpdesk operations support the (optional) ?repository? attribute, used to specify which repository the user belongs to. If not supplied, the user is assumed to reside in the repository belonging to the calling Agent. A breakdown of the individual operations follows.

### 6.3.1 Reset

Reset user credentials and send them to the user. Supply only the username(s), nothing else is required or allowed (Note: the user must have a transport to receive the new PIN).

```
<Reset repository=?repository1?>
  <User name="bob"/>
  <User name="joe"/>
  <User name="ann"/>
</Reset>
```

### 6.3.2 Strings

Send security strings to the user, ie via their assigned transport. Supply only the username, nothing else is required or allowed.

```
<Strings>
  <User name="ann"/>
</Strings>
```

### 6.3.3 Sync Token

Two consecutive one-time passwords are entered in order to synchronise the token with the Swivel server. Can be used of either HOTP or TOTP tokens, but probably more likely to be required for HOTP (or event-based) tokens.

```
<OathSync>
  <User name="tokenuser"/>
  <OTP1>123456</OTP1>
  <OTP2>456789</OTP2>
</OathSync>
```

If the Synchronisation is successful as pass message is returned.

Possible fail messages are

OATH\_SYNC\_FAILURE : The token could not be synced using the submitted codes

OATH\_TOKEN\_NOT\_FOUND\_FOR\_USER : The user does not have a token

OATH\_SEED\_ERROR : There is an invalid seed associated with the token

### 6.3.4 Update

Update user details. See [Admin API](#) for valid sub-elements

```
<Update>
  <User name="bob">
    <Credentials password=?mydogsname?/>
    <Policy pinNeverExpires="false"/>
    <Oath SerialNumber="12345"/>
  </User>
</Update>
```

All of the above operations support the optional ?repository? attribute used to specify which repository the user belongs to.

### 6.3.5 Purge

Purge deleted users from a named repository:

```
<PurgeDeleted repository="repos-name" />
```

The repository attribute is the only content permitted.

The response will contain the number of users purged.

## 6.4 Responses

Responses to helpdesk requests follow the same structure as that for [admin requests](#). The consist of a HelpDeskResponse element containinf the actions completed. eg for the above update request the response would be

```
<?xml version="1.0" ?>
<HelpdeskResponse>
  <Update>
    <User name="bob" />
  </Update>
</HelpdeskResponse>
```

If the request fails for any reason then a FAIL is included in the the User element. Any additional information why the command failed maybe in the PINsafe server logs.

```
<?xml version="1.0" ?>
<HelpdeskResponse>
  <Update>
    <User name="bob">FAIL </User>
  </Update>
</HelpdeskResponse>
```

A response may be a mixture of successes and fails, so in the example above where a request was made to reset the credentials of bob, joe and ann may get the response through their alert transport.

```
<?xml version="1.0" ?>
<HelpdeskResponse>
  <Reset>
    <User name="bob"/>
    <User name="joe">FAIL</User>
    <User name="ann"/>
  </Reset>
</HelpdeskResponse>
```



## 6.5 Error Messages

Refer to the [Admin API](#) article for a list of error responses.

# 7 ReportingAPI

## 7.1 Introduction

Swivel has developed an API, the [User Admin API](#), to allow an external application to perform CRUD (Create, Read, Update, Delete) style operations on users. For most installations, these operations are performed by the User Synchronization job but for larger user populations synchronization may be impractical. The User Admin API addresses this need. In conjunction with the User Admin API, Swivel has also developed another API allowing external applications to perform the functions usually performed by a helpdesk operator from the admin console. These are typically day to day functions not viewed as part of the User Admin API such as user unlock, PIN reset etc. This is the [Helpdesk API](#)

There is additionally read-only functionality provided by the **Reporting API** that can be used to read the status (eg idle, locked) of user accounts.

## 7.2 PRINCIPLES

Following the principles behind the existing Agent API used for authorisation, both of the API's are based around XML documents for both request and reply. The basic idea is to build a document containing details of operation(s) to be performed and the user(s) on which they are to be performed, submit it to PINsafe via HTTP and PINsafe will reply with a document detailing which operations succeeded and which, if any, failed. All operations via the API will be logged in the standard PINsafe log. Only authorised PINsafe Agents will be able to submit requests. This is in line with Agents used for authorisation. The API is case sensitive, the correct case shown in the included examples.

In order to use the Admin API, you must create a PINsafe Agent for the computer, or sub-net of computers, that will execute AdminAPI commands. Only the computer(s) defined by this Agent can create, read, update or delete users belonging to this Agent, and the Agent cannot read, update or delete users created by other repositories. The Agent can therefore be regarded as another type of repository. It is possible, but not recommended, to manipulate users created by a normal (e.g. XML or Active Directory) repository, by giving the Agent exactly the same name as the repository. Be aware if you do this, however, that any subsequent User Sync of the repository will potentially overwrite modifications made by the Agent. At present, any PINsafe Agent can act as an AdminAPI repository, but future versions of PINsafe will require you to explicitly enable the "Act as Repository" property of an Agent.

These restrictions do not apply to HelpdeskAPI Agents. Although only Agents can use these commands, it is possible to specify that the HelpdeskAPI command operates on a different repository, or on all repositories.

### 7.2.1 Sending an Admin API request

An Admin API request must be sent as an HTTP(S) request to the pinsafe application, either as a GET or POST request.

For a GET request, the format is

```
https://[swivelserver]:8080/pinsafe/AdminXML?xml=[request body]
```

For a POST request, the URL is

```
https://[swivelserver]:8080/pinsafe/AdminXML
```

and the XML request should form the content of the POST.

## 7.3 REPORTING API

See Also: [Reporting Using Agent-XML How to Guide](#)

Although listed here as a separate API, the Reporting API is actually part of the [User Admin API](#) and these elements are only valid as part of an AdminRequest eg

```
<?xml version="1.0" ?>
<AdminRequest secret="MyHelpdeskAgent" version="3.4">
  <Report repository="local">
    <Disabled/>
  </Report>
</AdminRequest>
```

### 7.3.1 Disabled

Report disabled user accounts. This returns a list of all users from a given repository that are marked as disabled in the PINsafe database. **Note** that this interrogates the PINsafe database and not the repository.

```
<Report repository="local">
  <Disabled/>
</Report>
```

### 7.3.2 Idle

Report users idle since the specified date.

```
<Report>
  <Idle repository="myRepository" since="12-Mar-2007"/>
</Report>
```

Note: This does not include users that have been created, but have never used PINsafe.

### 7.3.3 Locked

Report locked user accounts.

```
<Report repository="*>
  <Locked/>
</Report>
```

All elements support the optional repository attribute and `??` is taken to mean `?all repositories?`. Therefore the above example will instruct PINsafe to return a list of ALL locked users.

### 7.3.4 Count

(Since 3.8)

Rather than list users you can request a count of users: For example

```
<AdminRequest secret="secret" version="3.8">
<Report repository="">
<CountUsers/>
</Report>
</AdminRequest>
```

### 7.3.5 License information

Check the license details (expiry and number of users)

```
<SASRequest>
<Version>3.8</Version>
<Secret>secret</Secret>
<Action>GetLicenceInfo</Action>
<LicenceNumber>your-license-key</LicenceNumber>
</SASRequest>
```

### 7.3.6 AllUsers

(Since 3.8)

Report all users in a repository.

```
<Report repository="">
<AllUsers/>
</Report>
```

### 7.3.7 AllUsersDetailed

(Since 3.9)

Report details of all users in a repository.

```
<Report repository="">
<AllUsersDetailed/>
</Report>
```

## 7.4 Responses

The responses to reporting requests are a list of users that meet the definition of the request.

For example

```
<?xml version="1.0"?>
<AdminResponse>
<Report repository="fred">
<Locked>
<User name="jim1"/>
</Locked>
</Report>
</AdminResponse>
```

The exception to this is the idle report as this also includes for each user the time that they last authenticated

```
<?xml version="1.0"?>
<AdminResponse>
<Report repository="">
<Idle>
<User name="test" lastLogin="2009-06-03 10:38:46.648"/>
<User name="test2" lastLogin="2009-06-03 16:45:19.413"/>
</Idle>
</Report>
</AdminResponse>
```

Response to the Count report is the number of users.

```
<?xml version="1.0"?>
<AdminResponse>
<Report repository="">
<CountUsers>
<total>53</total>
<licensed>101</licensed>
</CountUsers>
</Report>
</AdminResponse>
```

**Note:** The `<licensed>` element in the Count report response only appears from 3.9.2 onwards.

General Errors and parsing errors are described in [Admin API](#).

## 8 Rng

How does Swivel Core generate security strings?.

Swivel uses the Java Secure Random number generator for all random numbers within PINsafe.

This class provides a cryptographically strong random number generator (RNG). .....

"A cryptographically strong random number minimally complies with the statistical random number generator tests specified in FIPS 140-2, Security Requirements for Cryptographic Modules, section 4.9.1. Additionally, SecureRandom must produce non-deterministic output and therefore it is required that the seed material be unpredictable and that output of SecureRandom be cryptographically strong sequences as described in RFC 1750: Randomness Recommendations for Security."

This is from the Sun JAVA website <http://java.sun.com/javase/6/docs/api/java/security/SecureRandom.html>

## 9 Session Sharing

Swivel Session Sharing

White Paper

## 10 Introduction

From version 3.9.5 onwards, Session sharing is included as part of [Appliance Synchronisation](#). See Also [Single Channel Session Cache](#).

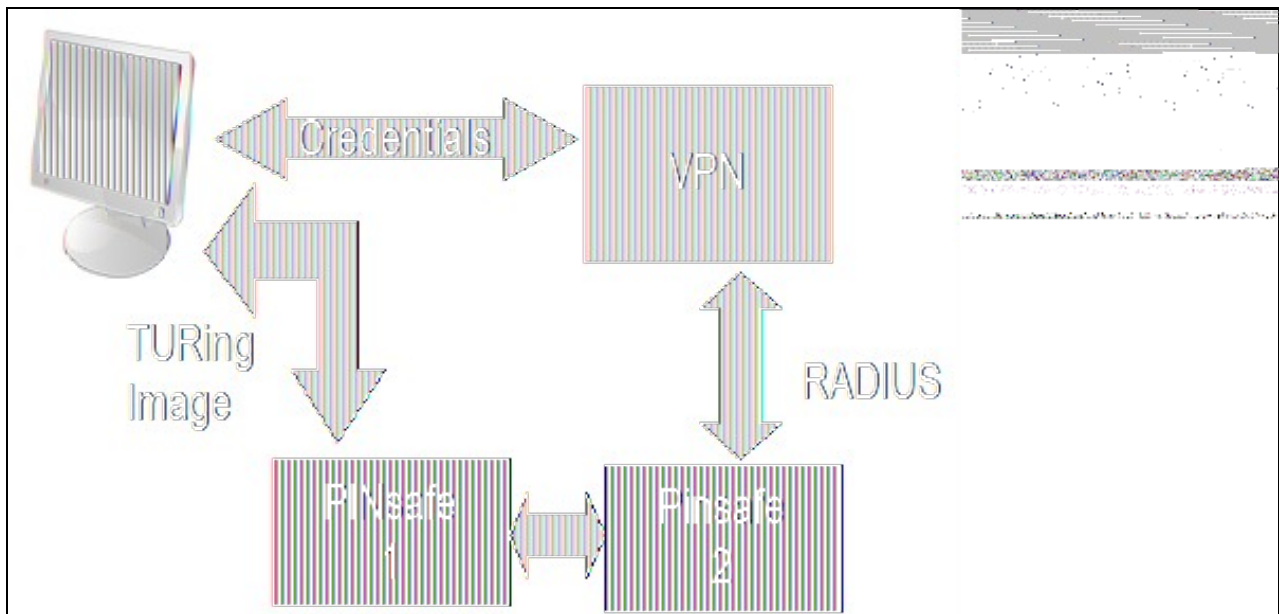
This white-paper explains the session sharing feature that was added in Version 3.5 of Swivel. It explains what this feature is, why you may wish to and how to implement and enable it on Swivel instances (it is not enabled by default).

Note also that [PINsafe RADIUS Proxy](#) may also be used in PINsafe 3.8 onwards as an alternative to Session Sharing.

## 11 Session Sharing Explained

For Active-Active HA pairs, when using single channel or on-demand dual channel, it has been a requirement that the user authenticates to the same server from which they retrieved the security string. This has prevented Swivel servers from being deployed in a truly load-balanced way. Active/Passive installations would not use session sharing.

Version 3.5 of the Swivel software now removes this requirement. Now when a session is started on one Swivel server, that session can be shared with the other Swivel server, so that either server can accept the subsequent authentication request.



Session sharing means if Swivel server 1 supplied the **TURING** image the VPN can still authenticate the user via the Swivel 2 server

The mechanism for this is that the sessions that are 'in progress', ie where a user has requested an image or message, but not yet authenticated, are stored in a cache that is shared between the two servers. In a similar way to the way that the databases are shared in a Swivel active-active configuration.

Swivel servers configured to share sessions must also share a common user-population.

The session sharing can, but need not be, used with an Active-Active pair. Clearly to support the use of this feature there must be sufficient bandwidth, with low-enough latency, between the two servers.

At the network level, session sharing uses IP Multicast, therefore it is a requirement that the network infrastructure supports multicast.

### 11.1 How it works

When a user uses TURING images or on-demand SMS messages to authenticate, the associated security string is only valid for a short time. When a security string is requested in either of these two ways, a session is created within Swivel. This records, in memory, all the information required to allow the user to authenticate using that security string. If the user does not authenticate in time, and the security string becomes invalid, the session is invalidated and destroyed.

Similarly if the user requests another security string, a new session is created and the previous session is deleted.

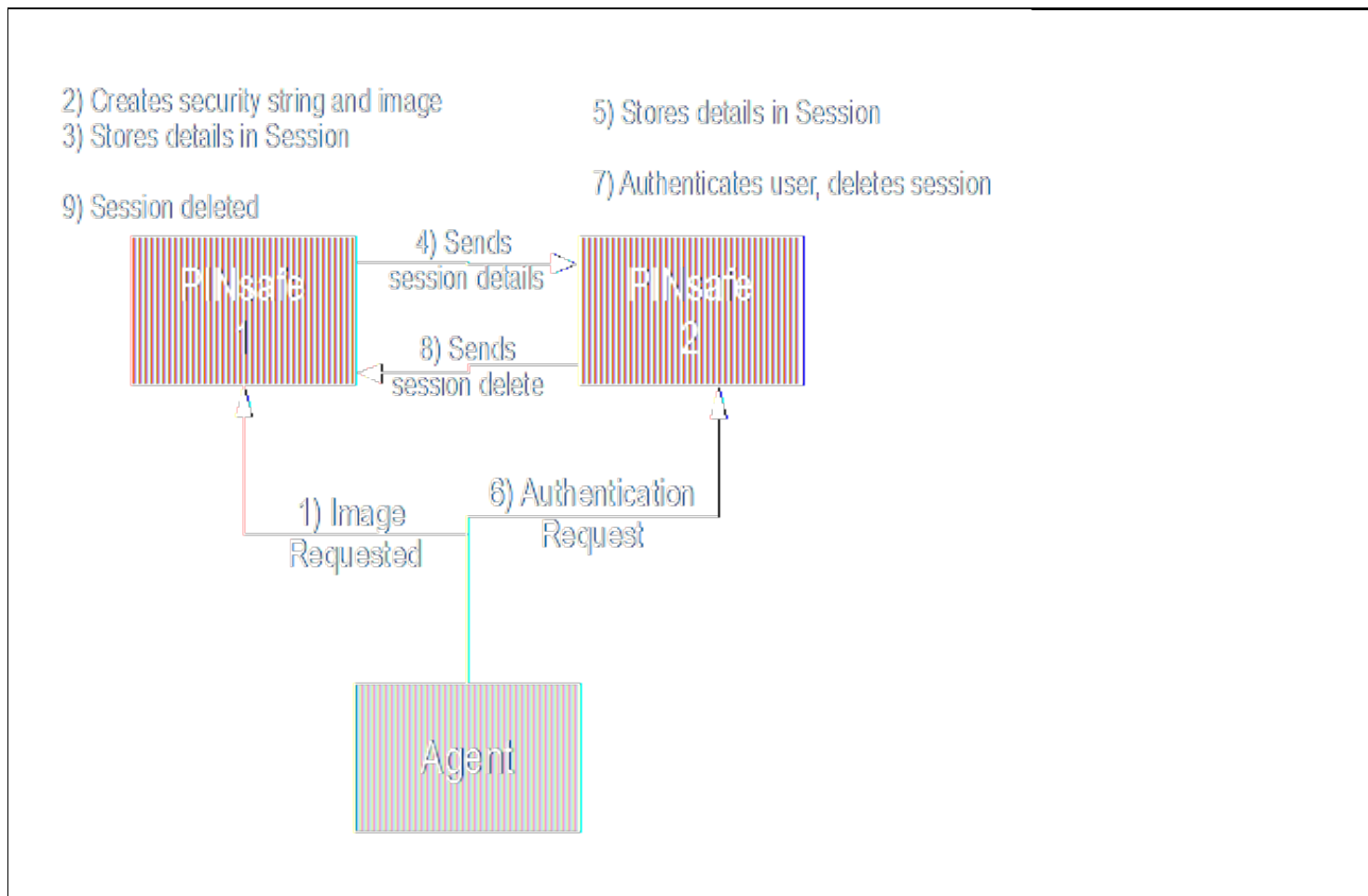
These sessions, as they are short-lived, are stored in memory rather than in a database (unlike the dual-channel security string messages sent out automatically after every authentication attempt, which are stored in the database and therefore replicated across both Swivel servers in an active-active pair).

When a pair of Swivel servers have been set up to share their sessions every time a change is made to the session cache on one server a message is sent across to the other Swivel server informing it of the change. This means that, in effect, both servers share the same session-cache.

This common view of the current active sessions means that a user can authenticate to any server.

#### Example.

An authentication agent (e.g. a VPN) may be configured to use either one of a pair of Swivel servers



Session sharing example

1. The agent submits a request for a TURING image to the first Swivel server.
2. The first Swivel server receives this request, creates the security string and the required TURING image, which is returns to the Agent.
3. It then stores the details in a session that it stores in memory.
4. In addition, as it is configured for session sharing it sends the details of the session it has created, via multicast, to any other Swivel servers that may need this information. **NB** The use of multicast means that if, for example, the second Swivel server became unavailable for some reason the first Swivel server does not waste time trying to send messages to it. It is possible to set the session sharing to use unicast, but this is not advisable.
5. The second Swivel server, as it also configured for session sharing, receives this information and also creates and stores the session.
6. The authentication request them is sent to the second Swivel server.
7. As this second server has details of the session that was created for the user, it can authenticate the user. Once the user has authenticated the session is no- longer valid, so it is deleted.
8. The second Swivel server therefore sends a message, again via multi-cast, informing any Swivel servers that the session is no longer valid.



## 12 Session Sharing Implementation

For Swivel Appliances see [Enabling Appliance Session Sharing](#)

For software installations and further assistance see the below.

### 12.1 Network Configuration

The Swivel servers need to be connected to a network that supports multicast traffic. Most switches are capable of supporting multicast although some switches / firewalls may be configured to block or disallow multicast.

### 12.2 Firewalls Settings

Note: When changing the firewall settings verify the following [Known Issues](#) before changing the firewall rules.

Any firewalls between the two Swivel servers needs to be opened to allow the messages to be passed between the servers. This includes the Linux firewalls on the appliances themselves. A rule must be added to the chain to accept traffic from the other appliance and to allow traffic to be sent to the other appliance.

<input type="checkbox"/>	Accept	If protocol is UDP and destination is 224.0.0.25
<input type="checkbox"/>	Accept	If source is 192.168.0.212
<input type="checkbox"/>	Accept	If destination is 192.168.0.212
<input type="checkbox"/>	Accept	If protocol is UDP and destination port is 631

Additional Firewall rules,

assuming other server is on 192.168.0.212

**The above rule means all traffic from the other server will be allowed. A better rule would just open up the multicast port being used. This is 4446 by default and is defined within the cache.xml file multicastGroupPort=4446**

#### 12.2.1 Primary Appliance Firewall Configuration

On the Primary Swivel appliance edit /etc/sysconfig/iptables with WinSCP:

After the line "-A RH-Firewall-1-INPUT -i lo -j ACCEPT" add the line:

```
-A RH-Firewall-1-INPUT -i eth0 -s 192.168.0.37 -j ACCEPT
```

(where 192.168.0.37 is the IP of the Standby machine. Replace this example IP address with your Standby IP)

#### 12.2.2 Standby Appliance Configuration

On the Standby Swivel appliance edit /etc/sysconfig/iptables with WinSCP:

After the line "-A RH-Firewall-1-INPUT -i lo -j ACCEPT" add the line:

```
-A RH-Firewall-1-INPUT -i eth0 -s 192.168.0.36 -j ACCEPT
```

(where 192.168.0.36 is the IP of the Primary machine. Replace this example IP address with your Primary IP)

#### 12.2.3 Appliance Firewall Restart

Once you've made these changes, on BOTH machines restart ipchains:

```
service iptables restart
```

#### 12.2.4 Appliance connectivity and listener

You should find that you return a result when running the following netstat command:

```
[admin@primary ~]# netstat -an | grep 4446
udp        1520      0 0.0.0.0:4446          0.0.0.0:*
udp         0        0 0.0.0.0:4446          0.0.0.0:*
```

## 12.3 Time Synchronisation

Sessions are timestamped by the server that creates them, sessions are only valid for finite time, so the clocks on the servers must be synchronised. As clocks drift, the use of an NTP server is essential. NTP servers can be configured from the Appliance menus.

Login to the command line to check and start the NTP service if necessary:

```
[admin@primary ~]# service ntpd status
ntpd is stopped
[admin@primary ~]# service ntpd start
Starting ntpd: [ OK ]
[admin@primary ~]#
```

See the [NTP servers](#) article for further information.

## 12.4 Swivel Settings

There are two elements to the Swivel configuration. First you need to configure Swivel to use session sharing, then select multicast session sharing.

To select session sharing you need to edit the config.properties file, the default settings for this file for Swivel 3.6 onwards are:

```
SECURITY_STRING_GENERATOR = com.swiveltechnologies.pinsafe.server.utility.SecurityString
SESSION_MANAGER = com.swiveltechnologies.pinsafe.server.session.LocalSessionManager
```

The file can be found under one of the following locations:

- Swivel 3.9.2 onwards on appliance: /home/swivel/.swivel/conf
- Swivel 3.9.2 onwards on software: USER\_HOME/.swivel/conf. Example Windows 7 c:/users/<username>/.swivel
- Swivel 3.5 to 3.9.1 or earlier: /usr/local/tomcat/webapps/pinsafe/WEB-INF/conf
- Swivel 3.5 to 3.9.1 or earlier on software: <path to Tomcat>/webapps/pinsafe/WEB-INF/conf

Note many of the transitory data settings were moved in 3.9.1, but the config.properties was moved in version 3.9.2

### Take a backup up copy of config.properties

This file includes a setting for Session Manager, the default value for Version 3.5 is;

com.swiveltechnologies.pinsafe.session.LocalSessionManager and needs to be changed to com.swiveltechnologies.pinsafe.session.DistributedCacheSessionManager as shown below:

```
SECURITY_STRING_GENERATOR = com.swiveltechnologies.pinsafe.utility.SecurityString
SESSION_MANAGER = com.swiveltechnologies.pinsafe.session.DistributedCacheSessionManager
```

for Version 3.6 and later the SESSION\_MANAGER line needs to be changed from com.swiveltechnologies.pinsafe.server.session.LocalSessionManager to com.swiveltechnologies.pinsafe.server.session.DistributedCacheSessionManager

```
SECURITY_STRING_GENERATOR = com.swiveltechnologies.pinsafe.server.utility.SecurityString
SESSION_MANAGER = com.swiveltechnologies.pinsafe.server.session.DistributedCacheSessionManager
```

To specify multicast session sharing navigate to the pinsafe/WEB-INF/classes folder. By default there will be two files related to session caching; multicast-cache.xml and peer-cache.xml.

To specify the use of multicast you need to copy the multicast-cache.xml file to cache.xml

### check file permissions and ownership

Tomcat must then be restarted for these changes to take effect

Repeat this for all Swivel servers.

Once these process have completed on all servers, they should all be sharing a common session-cache.

## 13 Testing

With session sharing enabled you can request a security string from one server and authenticate to another (assuming the user exists on both servers). Therefore the easiest way to test is via the Swivel Administration Console. Enter the username in the first login page, start the session, then enter the same username and the resultant one-time code on the second Swivel server. This should succeed. Entering the same one-time code in the first Swivel server should then fail, as the session should have been deleted from both servers.

## 14 Considerations

- The protocols use for the multicast session-sharing are not encrypted. Whereas just sniffing the network will not tell an attacker any useful information, an attacker with the time, expertise and access to the network would be able determine the details of the sessions passed between the servers. However this would only reveal the details of the security strings, this attack would have be combined with an attack that captured the submitted credentials.
- You may need to enable PIM (Protocol Independent Multicast) on your switch or VLAN for Multicast to work.

Swivel Multicast is configured using the RFC implementation with Multicast and IGMP joins for a specific group or groups.

# 15 Troubleshooting

**ERROR 127.0.0.1:Exception on replication of putNotification. RemoteException occurred in server thread; nested exception is: java.rmi.UnmarshalException: error unmarshalling arguments; nested exception is: java.io.InvalidClassException: net.sf.ehcache.Element; local class incompatible: stream classdesc serialVersionUID = 1098572221246444544, local class serialVersionUID = 3343087714201120157. Continuing...**

**127.0.0.1:Session started for user: xyz.**

This error has been seen in Swivel version 3.9.3 due to control codes in the config.properties file, care must be taken when editing such files such as using Windows third party tools.

**java.lang.ClassNotFoundException:**

**com.swiveltechnologies.pinsafe.server.session.DistributedCacheSessionManager**

```
org.apache.catalina.core.StandardContext loadOnStartup
SEVERE: Servlet /pinsafe threw load() exception
java.lang.ClassNotFoundException:
com.swiveltechnologies.pinsafe.server.session.DistributedCacheSessionManager
    at org.apache.catalina.loader.WebappClassLoader.loadClass(WebappClassLoader.java:1386)
    at org.apache.catalina.loader.WebappClassLoader.loadClass(WebappClassLoader.java:1232)
    at java.lang.Class.forName0(Native Method)
    at java.lang.Class.forName(Unknown Source)
    at com.swiveltechnologies.pinsafe.server.config.Config.load(Config.java:87)
    at com.swiveltechnologies.pinsafe.server.config.Startup.init(Startup.java:126)
    at javax.servlet.GenericServlet.init(GenericServlet.java:212)
    at org.apache.catalina.core.StandardWrapper.loadServlet(StandardWrapper.java:1139)
    at org.apache.catalina.core.StandardWrapper.load(StandardWrapper.java:966)
    at org.apache.catalina.core.StandardContext.loadOnStartup(StandardContext.java:3996)
    at org.apache.catalina.core.StandardContext.start(StandardContext.java:4266)
    at org.apache.catalina.core.ContainerBase.addChildInternal(ContainerBase.java:760)
    at org.apache.catalina.core.ContainerBase.addChild(ContainerBase.java:740)
    at org.apache.catalina.core.StandardHost.addChild(StandardHost.java:544)
    at org.apache.catalina.startup.HostConfig.deployDescriptor(HostConfig.java:626)
    at org.apache.catalina.startup.HostConfig.deployDescriptors(HostConfig.java:553)
    at org.apache.catalina.startup.HostConfig.deployApps(HostConfig.java:488)
    at org.apache.catalina.startup.HostConfig.start(HostConfig.java:1150)
    at org.apache.catalina.startup.HostConfig.lifecycleEvent(HostConfig.java:311)
    at org.apache.catalina.util.LifecycleSupport.fireLifecycleEvent(LifecycleSupport.java:120)
    at org.apache.catalina.core.ContainerBase.start(ContainerBase.java:1022)
    at org.apache.catalina.core.StandardHost.start(StandardHost.java:736)
    at org.apache.catalina.core.ContainerBase.start(ContainerBase.java:1014)
    at org.apache.catalina.core.StandardEngine.start(StandardEngine.java:443)
    at org.apache.catalina.core.StandardService.start(StandardService.java:448)
    at org.apache.catalina.core.StandardServer.start(StandardServer.java:700)
    at org.apache.catalina.startup.Catalina.start(Catalina.java:552)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
    at java.lang.reflect.Method.invoke(Unknown Source)
    at org.apache.catalina.startup.Bootstrap.start(Bootstrap.java:295)
    at org.apache.catalina.startup.Bootstrap.main(Bootstrap.java:433)
```

This error has been seen in Swivel 3.8 due to control codes in the config.properties file, care must be taken when editing such files such as using Windows third party tools.

# 16 Swivel Combined Client

## 16.1 Overview

This document describes a combined authentication and administration client for Swivel Server APIs which can be used in web servers using ASP.Net. It is a convenient wrapper around the XML-based APIs described [here](#).

The client can also be used in non-web applications using the .Net Framework. See configuration below.

## 16.2 Prerequisites

The client DLL is available from [here](#). This is version 1.3 of the client, which includes support for user attributes and for TLS protocol versions 1.1 and 1.2. This requires version 3.9.7 or later of the Swivel server. For TLS 1.1/1.2 support, version 3 of the Swivel appliance is required.

An example website demonstrating some of the features of the client is available from [here](#). This download includes the DLL above, so you don't need to download both. Again, this test server uses the new client.

This version of the client requires Microsoft.Net Framework version 4.5 or later.

## 16.3 Using the Client

### 16.3.1 Configuring The Swivel Server

In order to use the client with a particular Swivel Server, you must define the computer running your web application as an Agent on that server. To do this:

- Log into the Swivel Server Administration Console
- Go to Server -> Agents
- In version 3.8 or later, expand the entry at the bottom labelled "New Entry". In earlier versions, this will already be visible.
- Enter a name for the new Agent. Note that any users created using the API will be added to a repository with this name, so the name should not be the same as an existing repository if you intend to manage users through this Agent.
- Enter the IP address of the web server under Host/IP.
- If you will be managing users through this Agent, set "Can act as Repository" to Yes.
- The remaining entries can be left as default for now. Click Apply.

Depending on how you intend to use the client, you may like to create a new Repository Group for users created by the Agent, or you may prefer to use an existing Group. You can create a new Group under Repository -> Groups. All you need to add is a name in the blank entry at the bottom. There is no need to add definitions for other repositories, or to specify rights for this Group, as you will be specifying user rights explicitly when the user is created. You may also like to go back to the Agent definition and set the Group for this Agent to the Group you have just created. If you do this, only users created by the Agent will be able to authenticate through the Agent.

If you have created a new Group, you should also set up Transports for this Group. You should at least set up an Alert Transport, and if you are using dual channel authentication, a Strings Transport as well. Note that if you want to use an existing Transport, you will need to copy the class name to a new Transport and give it a new name. Select the Group you have just created as the Strings or Alert Group as appropriate.

### 16.3.2 Configuring Your ASP.Net Application

In order to use the client, you need to add certain entries to the web.config file in your ASP.Net application. The following is an example:

```
<appSettings>

  <add key="PINsafeServer" value="myserver"/>
  <add key="PINsafePort" value="8080"/>
  <add key="PINsafeContext" value="sentry"/>
  <add key="PINsafeSecret" value="secret"/>
  <add key="PINsafeSecure" value="True"/>
  <add key="PINsafeAcceptSelfSigned" value="False"/>
  <add key="AllowNonPINsafeUsers" value="False"/>
  <add key="IgnoreDomainPrefix" value="True"/>
  <add key="IgnoreDomainSuffix" value="False"/>
  <add key="PINsafeAgentVersion" value="3.97" />
  <add key="PINsafeTlsProtocols" value="Tls12" />

</appSettings>
```

The comments at the start and end are not necessary - they are for clarity - but there is a method in the client to remove these settings which requires these comments. The <appSettings> element should be contained within the main <configuration> element, and may already exist, depending on your application.

If you are using the client in an executable (i.e. non-web) application, you can use exactly the same settings, but these must be in a file named *Application.exe.config*, where *Application* is the name of the executable application. This must be in the same directory as the program executable.

The meanings of the key names are shown below:

- **PINsafeServer** - The host name or IP address of the Swivel server
- **PINsafePort** - The port used by the Swivel server, normally 8080
- **PINsafeContext** - The context (application URL) of the Swivel server, normally "pinsafe"
- **PINsafeSecret** - The shared secret as configured in the Agent entry previously
- **PINsafeSecure** - "True" if HTTPS is to be used for communication with, "False" otherwise
- **PINsafeAcceptSelfSigned** - "True" if SSL certificate errors should be ignored
- **AllowNonPINsafeUsers** - "True" if users unknown to PINsafe should be authenticated automatically, "False" if they should be rejected
- **IgnoreDomainPrefix** - "True" if the domain prefix in usernames such as domain\user should be stripped off before checking with Swivel server
- **IgnoreDomainSuffix** - "True" if the domain suffix in usernames such as user@domain should be stripped off before checking with Swivel server
- **PINsafeAgentVersion** - Sets the *Version* attribute sent with the API request. The default is "3.4". To support user attributes, this should be set to "3.97".

- **PINsafeTlsProtocols** - Specifies which TLS protocols are supported. You can specify Ssl3, Tls1, Tls11 or Tls12. Separate supported protocols with commas. The default is "Tls1,Tls11,Tls12".

NOTE: on an appliance, you should not use the proxy application (and port 8443), as functionality in the proxy is deliberately limited. You must use the pinsafe application to get the full functionality out of the client.

If it is not practical to provide the settings in a configuration file, you can create an instance of the SwivelSettings class, and initialise it as follows:

```
SwivelSettings swivelSettings = SwivelSettings.EmptySettings;  
swivelSettings.Server = "myserver";  
swivelSettings.Port = 8080;  
swivelSettings.Context = "sentry";  
swivelSettings.Ssl = true;  
swivelSettings.Secret = "secret";  
swivelSettings.AcceptSelfSigned = false;  
swivelSettings.TlsProtocols = SecurityProtocolType.Tls12;
```

You can then use this instance when creating a request - see the class documentation for details.

### 16.3.3 Implementing the Swivel Client in ASP.Net Applications

In order to use the client DLL, you need to copy it to the Bin folder of your ASP.Net application.

Technical documentation for the client is in progress. Please see the example application.

The namespace for all classes in the client is **swivelsecure.client**.

Documentation of the authentication class can be found [here](#)

The user management classes fall into two categories: [administrator](#) and [helpdesk](#). The administrator methods are more extensive, allowing you to create, update and delete users. However, they are limited to users belonging to a single repository, named after the Agent corresponding to the computer the application is running on. The helpdesk methods are more limited, typically read-only methods, but with some update functionality. However, these methods can apply to users in other repositories. See the two documents for details.

## 17 Category:Symptom



## 18 Category: Whitepaper