Cisco SocialMiner
Developers Guide — Release 8.5(2)

Contents:

I. API Development

[contents]

This part describes the SocialMiner API.

1. About the SocialMiner APIs

[contents]

The SocialMiner API's provide all the functionality that is available in the SocialMiner web interfaces. Using the API's, you can create your own client applications for configuring and using SocialMiner.

2. General Usage

[contents]

The API's are accessed over HTTP using REST through POST, GET, PUT, and DELETE requests. The input format is **XML** for all API calls other than GET and DELETE. All output is provided as XML when there is a response other than HTTP headers. The descriptions of the commands detail which commands accept which format.

Example commands are provided using cURL. cURL is an open source command-line application (and library) that makes it easy to demonstrate how the API commands are sent to SocialMiner, and what data is coming back from the server. cURL is available for all major operating systems. The examples use the following cURL options:

- **-I** : Include the headers in the returned output
- **-H** : Header to send with the request
- **-X** : Request method to use (POST, PUT, DELETE, etc.)
- **-d** : Data to send in the request

XML is case sensitive. When sending XML data to the server, the tag names must match the same case as defined below. <Name> and <name> are two different XML elements.

2.1. HTTP Responses

[contents]

All errors are returned as HTTP 1.1 Status Codes. The common codes used by the SocialMiner API are:

- **200 OK**: Success
- **201 Created**: The requested item was created.
- **202 Accepted**: The request was accepted. Generally, a URL is provided to obtain additional details, for example, fo polling the OAuth status.
- **400 Bad Request**: The request is invalid. See the information returned in the ApiErrors message for more details.
- **401 Unauthorized**: The authentication credentials were not supplied or were incorrect.
- **404 Not Found**: The URI requested does not exist on the server.
- **500 Internal Server Error**: Something broke on the server. Submit a post to the SocialMiner Forums explaining what you did and the server's response.

Additionally, field specific and database errors are provided in an XML error message with the format:

```
<ApiErrors>
       <ApiError>
              <ErrorType>Type of Error</ErrorType>
              <ErrorData>Field Error Occurred</ErrorData>
              <ErrorMessage>A Description of the Error</ErrorMessage>
       </ApiError>
</ApiErrors>
```

2.2. API Authentication

[contents]

Authentication is required to use the API's. Authentication is HTTP Basic and you must use the username and password for a SocialMiner Administrator.

When viewing an API call through a web browser, for example *http://server:port/ccp-webapp/ccp/campaign/*, you are prompted by the browser for the username and password.

When accessing the API through an application such as cURL, you must pass the username and password with the request, for example:

```
curl -i -X GET http://username:password@server:port/ccp-webapp/ccp/campaign/
```

If you do not provide a username or password, or provide incorrect credentials, then a **401 Unauthorized** error is returned.

3. The Feed API

[contents]

The SocialMiner Feed API allows you to create, delete, update, and list feeds. SocialMiner feeds can be RSS feeds, Twitter Accounts, Twitter Streams, or Facebook Fan Pages. The feed object contains data about the feed, such as the URL of the feed, how often the feed is to be read (pollingInterval), if SocialMiner needs to access the feed through a proxy (useProxy), and the type of feed that it is (RSS, Facebook Fan Page, etc.).

One or more feeds are assigned to Campaigns.

The URL to access the Feed API is **http://server:port/ccp-webapp/ccp/feed**.

Note: The Shindig OpenSocial container in which SocialMiner runs requires that REST requests complete within five seconds. However, communication with the Twitter servers can exceed five seconds. This limitation means you must poll after making calls to the Feed API for creating Twitter Account feeds to verify the status returned. A diagram is provided to illustrate the API calls and expected poll responses.

Commands:

- Feed API Commands
- Authorizing Against Twitter

3.1. Feed API Commands

[contents]

You can perform the following commands:

- create
- delete
- list
- get
- update

3.1.1. create

[contents]

Creates a feed.

URL:	http://server:port/ccp-webapp/ccp/feed
HTTP Method:	POST

Parameters:	• name - string (unique)
	• description - string (optional)
	• url - string (required for types 1, 4, 5. Ignored otherwise.) (if type = 4, then you cannot use an IP address, you must use a hostname).
	• type - integer :
	  ◦ RSS = 1
	  ◦ Twitter Stream = 3
	  ◦ Facebook Fan Page = 4
	  ◦ Authenticated RSS = 5
	  ◦ Twitter Account = 6, for more details on creating a Twitter Account feed, see Authorizing Against Twitter
	  ◦ Push = 7
	• pollingInterval - integer, seconds between each request for new data from the feed server (required for types 1, 4, 5, 6. Ignored otherwise.).
	• minAge - integer (optional); the minimum age of a post, in seconds, before it is included in results. This allows you to exclude recent posts. (Ignored for type 3).
	• authenticationUsername - string, the username for an authenticated RSS feed (required for type 3 and 5).
	• authenticationPassword - string, the password for an authenticated RSS feed (optional).
	• keywords - comma separated list of keywords to search on. Up to 200 keywords can be defined for a total limit of 2000 bytes. Each keyword must be between 1-60 bytes. At least one keyword must be defined. (required for type 3, ignored otherwise). Any spaces in this field are interpreted as an "And" modifier for search. Commas in this field are interpreted as an "Or" modifier for search.
	• replyTemplateRefUrl - string (optional); The URL of the reply template used to respond to Social Contacts obtained from this feed. If this field is blank then no reply template is used.

Example XML Request Payload:	`<Feed>`
	`  <name>TestName</name>`
	`  <description>TestDescription</description>`
	`  <url>http://example.com</url>`
	`  <type>1</type>`
	`  <pollingInterval>1</pollingInterval>`
	`  <minAge>5</minAge>`
	`  <authenticationUsername>user</authenticationUsername>`
	`  <authenticationPassword>nobody</authenticationPassword>`
	`</Feed>`

Example using cURL:	`curl -i -H "Content-type: application/xml" -X POST -d "<Feed><name>TestName</name><description>TestDescription</description><url>http://www.example.com</url><type>1</type><pollingInterval>1</pollingInterval><minAge>0</minAge></Feed>" http://user:password@192.168.0.1/ccp-webapp/ccp/feed`

HTTP Response Headers:	The response contains the **URL** for the newly created feed.
	`HTTP/1.1 201 Created`
	`Location: http://192.168.0.1/ccp-webapp/ccp/feed/100162`

```
Content-Type: text/plain
Content-Length: 0
Date: Tue, 12 Jan 2010 16:15:04 GMT
```

3.1.2. delete

[contents]

Delete a feed.

URL:	http://server:port/ccp-webapp/ccp/feed/<id>
HTTP Method:	DELETE
Example using cURL:	`curl -i -X DELETE http://user:password@192.168.0.1/ccp-webapp/ccp/feed/100169`

HTTP Response Headers:	`HTTP/1.1 200 OK`
	`Content-Type: text/plain`
	`Content-Length: 0`
	`Date: Thu, 12 Jan 2010 16:44:35 GMT`

3.1.3. list

[contents]

List feeds.

URL:	http://server:port/ccp-webapp/ccp/feed
HTTP Method:	GET
Parameter:	summary - true/false (optional, defaults to "false"). When "true", only a subset of the XML elements are returned.
Example:	`http://192.168.0.1:80/ccp-webapp/ccp/feed/?summary=false`

Example XML Response:	`<Feeds>`
	`<Feed>`
	`<refURL>`
	`http://localhost:8080/ccp-webapp/ccp/feed/101246`
	`</refURL>`
	`<name>Example1</name>`
	`<description>Description1</description>`
	`<url>http://www.google.com</url>`
	`<type>1</type>`
	`<pollingInterval>5</pollingInterval>`
	`<minAge>0</minAge>`
	`<changeStamp>0</changeStamp>`
	`<replyTemplateRefUrl>http://192.168.0.1/ccp-webapp/ccp/template/reply/300</replyTemplateRefUrl>`
	`</Feed>`
	`<Feed>`
	`<refURL>`
	`http://localhost:8080/ccp-webapp/ccp/feed/101247`
	`</refURL>`

```
                <name>Example2</name>
                <description>Description2</description>
                <url>http://www.google.com</url>
                <type>1</type>
                <pollingInterval>5</pollingInterval>
                <minAge>0</minAge>
                <changeStamp>0</changeStamp>>
                <replyTemplateRefUrl>http://192.168.0.1/ccp-webapp/ccp/template/reply/301</replyTemplateRefUrl>
        </Feed>
</Feeds>
```

HTTP Response Headers:

```
HTTP/1.1 200 OK
Content-Type: application/xml
Transfer-Encoding: chunked
Date: Tue, 12 Jan 2010 16:47:58 GMT
```

3.1.4. get

[contents]

Return the data for a single feed. Note, for security, passwords are not returned for feeds. Any password elements will contain a masked password.

URL: http://server:port/ccp-webapp/ccp/feed/<id>

HTTP Method: GET

Example:

```
http://user:password@192.168.0.1:80/ccp-webapp/ccp/feed/100036
```

Example XML Response:

```
<Feed>
        <changeStamp>2</changeStamp>
        <name>Chevy Volt Twitter</name>
        <pollingInterval>60</pollingInterval>
        <refURL>http://192.168.0.1/ccp-webapp/ccp/feed/100002</refURL>
        <replyTemplateRefUrl>http://192.168.0.1/ccp-webapp/ccp/template/reply/300</replyTemplateRefUrl>
        <type>1</type>
        <url>http://search.twitter.com/search.atom?q=chevy+volt</url>
</Feed>
```

HTTP Response Headers:

```
HTTP/1.1 200 OK
Content-Type: application/xml
Transfer-Encoding: chunked
Date: Tue, 12 Jan 2010 16:50:46 GMT
```

3.1.5. update

[contents]

Update an existing feed.

URL: http://server:port/ccp-webapp/ccp/feed/<id>

HTTP Method: http://server:port/ccp-webapp/ccp/feed/<id>

Parameters:	• name - string (unique)
	• url - string (required) (if type = 4, then you cannot use an IP address, you must use a hostname).
	• type - integer :
	  ◦ RSS = 1
	  ◦ Twitter Stream = 3
	  ◦ Facebook Fan Page = 4
	  ◦ Authenticated RSS = 5
	  ◦ Twitter Account = 6, If changing username, you must re-authorize against twitter, see Authorizing Against Twitter.
	• pollingInterval - integer, seconds between each request for new data from the feed server (required).
	• minAge - integer (optional); the minimum age of a post, in seconds, before it is included in results. This allows you to exclude recent posts.
	• changeStamp - integer, required
	• authenticationUsername - string, the username for an authenticated RSS feed (required if type=5).
	• authenticationPassword - string, the password for; an authenticated RSS feed (optional) and, Twitter Stream feed (required).
	• replyTemplateRefUrl - string (optional); The URL of the reply template used to respond to Social Contacts obtained from this feed. If this field is blank then no reply template is used.

The changeStamp:	**Important:** You must provide the current changeStamp of the feed when you perform an update. If you do not provide the current changestamp then the update fails. This mechanism is in place so that two clients cannot edit the feed at the same time. The changeStamp increments by 1 if the update is successful.

Example XML Request Payload:	`<Feed>`
	`  <name>String</name>`
	`  <description>String</description>`
	`  <url>String</url>`
	`  <type>short</type>`
	`  <pollingInterval>Integer</pollingInterval>`
	`  <minAge>0</minAge>`
	`  <changeStamp>0</changeStamp>`
	`</Feed>`

Example using cURL:	`curl -i -H "Content-type: application/xml" -X PUT -d "<Feed><name>TestName5</name><description>Edited Description</description><url>http://example.com</url><type>1</type><pollingInterval>1</pollingInterval><minAge>0</minAge><changeStamp>0</changeStamp></Feed>" http://user:password@192.168.0.1/ccp-webapp/ccp/feed/103114`

HTTP Response Headers:	`HTTP/1.1 200 OK`
	`Content-Type: text/plain`
	`Content-Length: 0`
	`Date: Thu, 14 Jan 2010 15:49:17 GMT`

3.2. Authorizing Against Twitter

[contents]

You must obtain authorization for SocialMiner to access a twitter account for the *Twitter Account* feed type.

Twitter uses OAuth to provide secure communication between twitter and third-party application such as SocialMiner.

There are several additional steps to creating a *Twitter Account* feed type:

1. Use the Feed create API to create a type 6 feed.

2. Poll the Feed FeedOauthStatus API until the *status* is WAITING-FOR-AUTH-CALLBACK. The authUrl field is populated with the twitter authorization URL.
3. Access the authUrl URL through a browser session and allow authorization for SocialMiner.
4. Twitter returns a PIN code through the callback API. This is handled on the SocialMiner server and does not require any API calls from your application.
5. Poll the Feed FeedOauthStatus API until the status is *SUCCEEDED*. The feed has been created.

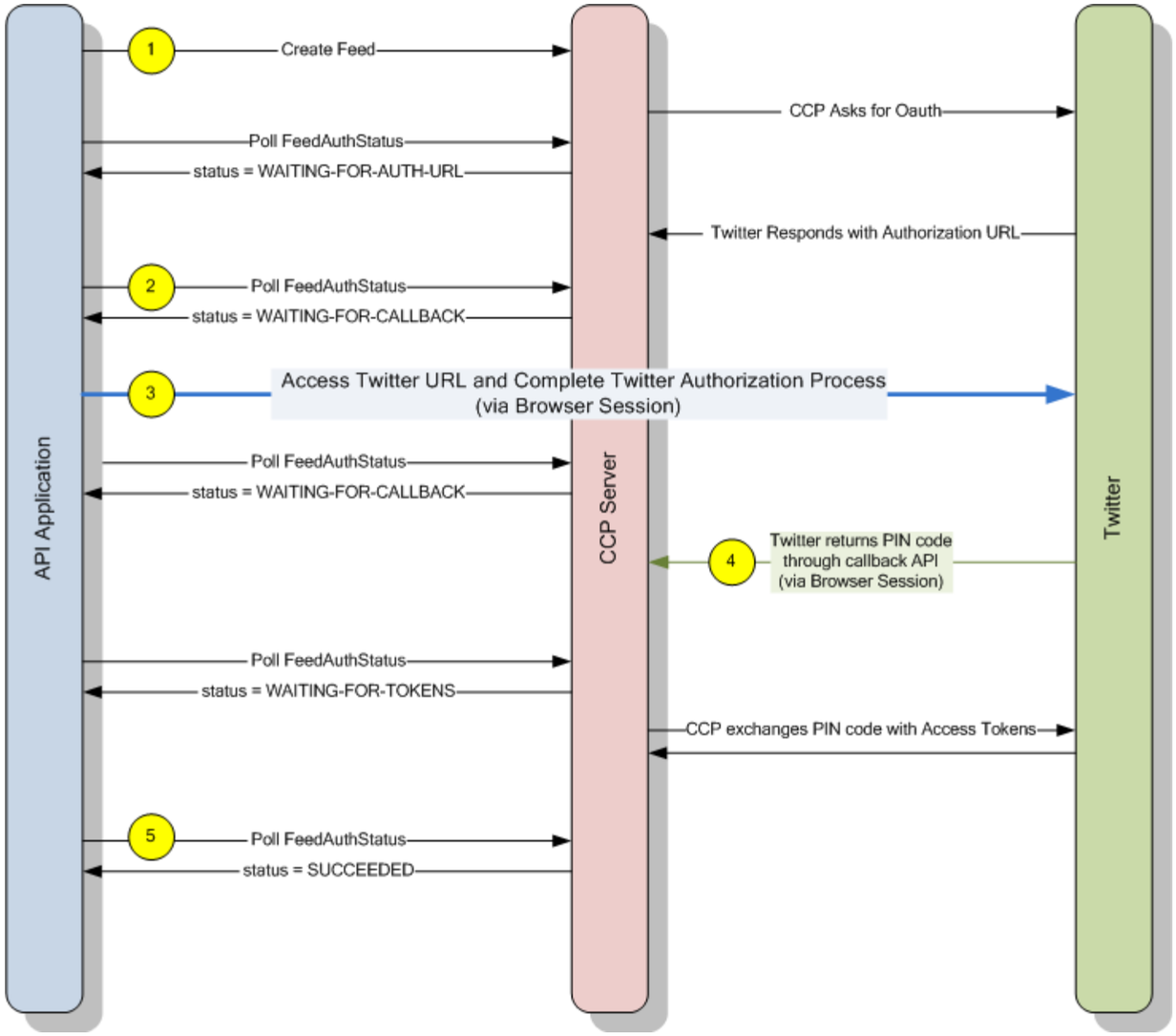

Figure 1: Twitter Account Authorization API Flow Diagram

3.2.1. FeedOauthStatus

[contents]

Used only for the *Twitter Account* feed type. Check the status of the authorization with twitter. Your application should poll this URL until status is "SUCCEEDED"

URL:	http://server:port/ccp-webapp/ccp/feed/oauth/<username>
HTTP Method:	GET
Example XML Response:	<FeedOAuthStatus> <authUrl>AUTH_URL</authUrl> <status>STATUS</status>

```
</FeedOAuthStatus>
```

- authURL is the authorization URL from twitter.
- status is one of:
  - SUCCEEDED - The application/user account has been authorized for use with twitter
  - FAILED - The application/user is not authorized for use with twitter. This could be due to a failed login or a timeout. The authorization request times out in 10 minutes.
  - FAILED-USER-MISMATCH - The username entered into the twitter authorization page does not match the username on the feed.
  - IN-PROGRESS - The authorization is in progress.

3.2.2. oathCancel

[contents]

Used only for the *Twitter Account* feed type. Cancel a pending OAuth request.

Note: If the OAuth request to twitter is completed before the call to oauthCancel is made, the configuration is not deleted.

URL:	http://server:port/ccp-webapp/ccp/feed/oauth/<username>
HTTP Method:	DELETE
Response:	Response is contained in the HTTP response code.

3.2.3. callback

[contents]

Used only for the *Twitter Account* feed type. This URL is the callback URL invoked by twitter after a user approves or denies the authorization request. Details are provided here only for reference. Do not call this API in your application.

URL:	http://server:port/ccp-webapp/ccp/feed/oauth/<username>/callback
HTTP Method:	GET
Responses:	*SUCCEEDED* or *Failed*.

4. The Proxy API

[contents]

The proxy API allows you to update and read proxy server settings. All feeds use the proxy if the proxy is enabled.

The URL to access the API is **http://server:port/ccp-webapp/ccp/proxy/default**

4.1. Proxy API Commands

[contents]

4.1.1. update

[contents]

Update the proxy configuration. By default, the configuration is blank and disabled.

URL:	http://server:port/ccp-webapp/ccp/proxy/default
HTTP Method:	PUT

Parameters:	• host - The hostname or IP address of the proxy server (required if enabled parameter is true)
	• port - the HTTP port for the proxy server (required if enabled parameter is true)
	• exclusions - a list of hostnames to exclude from being used by the proxy. (optional) Wildcards can be used. Examples:
	  ◦ localhost
	  ◦ *.cisco.com
	  ◦ 161.44.*
	  ◦ 192.168.1.1
	  ◦ **Note:** The exclusion list is limited to 255 total characters (not including the `<exclusion>` tags). There is an additional character per item in the list that acts as a separator.
	• enabled - true/false. Whether proxy use is enabled.

Example XML Payload:	`<Proxy>`
	`  <host>10.86.141.293</host>`
	`  <port>8080</port>`
	`  <exclusions>`
	`  <exclusion>localhost</exclusion>`
	`  <exclusion>*.cisco.com</exclusion>`
	`  <exclusion>161.44.*</exclusion>`
	`  <exclusion>192.168.1.1</exclusion>`
	`  </exclusions>`
	`  <enabled>true</enabled>`
	`</Proxy>`

Example Using cURL:	`curl -i -H "Content-type: application/xml" -X PUT -d "<Proxy><host>proxy.esl.cisco.com</host><port>80</port><exclusions><exclusion>*.cisco.com</exclusion></exclusions><enabled>true</enabled></Proxy>" http://admin:password@192.168.0.1/ccp-webapp/ccp/proxy/default`

4.1.2. get

[contents]

Get the proxy configuration.

URL:	http://server:port/ccp-webapp/ccp/proxy/default
HTTP Method:	GET

Example XML Response:	`<Proxy>`
	`  <enabled>true</enabled>`
	`  <exclusions>`
	`  <exclusion>*.cisco.com</exclusion>`
	`  </exclusions>`
	`  <host>proxy.cisco.com</host>`

```
        <port>80</port>
        <refURL>http://192.168.0.1/ccp-webapp/ccp/proxy/default</refURL>
 </Proxy>
```

5. The Campaign API

[contents]

The SocialMiner Campaign API allows you to create, update, delete, get, and list campaigns in the SocialMiner system.

Campaigns are collections of feeds and filterss. Campaigns generate lists of results matching the criteria defined in the campaign.

The URL to access the API is **http://server:port/ccp-webapp/ccp/campaign/**.

5.1. Campaign API Commands

[contents]

The Campaign API Supports the following commands:

- create
- update
- delete
- get
- list
- suggestedtags

5.1.1. create

[contents]

Create a campaign.

The publicId	When a campaign is created, SocialMiner generates a unique publicId for the campaign, based on the name you provide when you use the **create** command. The publicId is then used with the other methods for campaign. You can generate your own publicId rather than have SocialMiner generate one based on the campaign name. Spaces, slashes, and backslashes are not allowed in the publicId, it cannot be blank, and the publicId must conform to RFC 3986 for URI syntax. When SocialMiner creates a publicId from the provided name element it replaces spaces with underscores and slashes or backslashes with hyphens. SocialMiner then formats the resulting string according to RFC 3986. The publicId is not editable after it is created and does not change if you change the name of the Campaign.
URL:	http://server:port/ccp-webapp/ccp/campaign
HTTP Method:	POST
Parameters:	• name - string (unique, required) • description - string (optional) • publicId - string (optional, but must follow publicId format if specified) • includeExpr - string (optional). Include results with this search expression. • excludeExpr - string (optional). Exclude results with this search expression. • feeds - A list of feed URLs that are used in this campaign (optional) • filters - list of filter URLs that are used in this campaign (optional)

Example XML Request Payload:	<Campaign> <name>MyTestCampaign</name> <description>This is my test campaign</description> <publicId>MyTestCampaign</publicId> <feeds> <feed>http://192.168.0.1:80/ccp-webapp/ccp/feed/5000</feed> <feed>http://192.168.0.1:80/ccp-webapp/ccp/feed/5001</feed> </feeds> <filters> <filter>http://192.168.0.1:80/ccp-webapp/ccp/filter/6000</filter> <filter>http://192.168.0.1:80/ccp-webapp/ccp/filter/6001</filter> </filters> </Campaign>
Example using cURL:	curl -i -H "Content-type: application/xml" -X POST -d "<Campaign><name>MyTestCampaign</name><publicId>MyTestCampaign</publicId><description>This is my test campaign</description><include>Cisco Expert Advisor</include><exclude>ICM</exclude><feeds><feed>http://192.168.0.1/ccp-webapp/ccp/feed/100162</feed><feed>http://192.168.0.1/ccp-webapp/ccp/feed/100165</feed></feeds><filters><filter>http://192.168.0.1/ccp-webapp/ccp/feed/100167</filter><filter>http://192.168.0.1/ccp-webapp/ccp/feed/100168</filter></filters></Campaign>" http://user:password@192.168.0.1/ccp-webapp/ccp/campaign
HTTP Response Headers:	The URI of the created resource is returned if the creation if the campaign succeeded. HTTP response: HTTP/1.1 201 Created Location: http://192.168.0.1/ccp-webapp/ccp/campaign/MyTestCampaign Content-Type: text/plain Content-Length: 0 Date: Tue, 12 Jan 2010 16:41:14 GMT

5.1.2. update

[contents]

Update an existing campaign.

URL:	http://server:port/ccp-webapp/ccp/campaign/<publicId>
HTTP Method:	PUT
Parameters:	• name - string (unique, required) • description - string (optional) • publicId - string (optional, not editable) • includeExpr - string (optional). Include results with this search expression. • excludeExpr - string (optional). Exclude results with this search expression. • feeds - A list of feed URLs that are used in this campaign (optional) • filters - list of filter URLs that are used in this campaign (optional) • changeStamp - integer (required)
The changeStamp:	**Important**: You must provide the current *changeStamp* of the feed when you perform an update. If you do not provide the current changestamp then the update fails. This mechanism is in place so that two clients cannot edit the feed at the same time. The changeStamp increments by 1 if the update is successful.
Example	<Campaign>

XML Request Payload:	<name>MyTestCampaign</name> <description>This is my test campaign</description> <includeExpr>Cisco Expert Advisor</includeExpr> <excludeExpr>ICM</excludeExpr> <feeds> <feed>http://192.168.0.1:80/ccp-webapp/ccp/feed/5000</feed> <feed>http://192.168.0.1:80/ccp-webapp/ccp/feed/5001</feed> </feeds> <filters> <filter>http://192.168.0.1:80/ccp-webapp/ccp/filter/6000</filter> <filter>http://192.168.0.1:80/ccp-webapp/ccp/filter/6001</filter> </filters> <changeStamp>8</changeStamp> </Campaign>
Example using cURL:	curl -i -H "Content-type: application/xml" -X PUT -d "<Campaign><name>MyTestCampaign 5 Edited</name><description>Edited Description</description><include>Cisco Expert Advisor</include><exclude>ICM</exclude><feeds><feed>http://192.168.0.1/ccp-webapp/ccp/feed/103111</feed><feed>http://192.168.0.1/ccp-webapp/ccp/feed/103112</feed></feeds><filters><filter>http://192.168.0.1/ccp-webapp/ccp/feed/100167</filter><filter>http://192.168.0.1/ccp-webapp/ccp/feed/100168</filter></filters></Campaign>" http://user:password@192.168.0.1/ccp-webapp/ccp/campaign/103115

5.1.3. delete

[contents]

Delete an existing campaign.

URL:	http://server:port/ccp-webapp/ccp/campaign/<publicId>
HTTP Method:	DELETE
Example using cURL:	curl -i -X DELETE http://user:password@192.168.0.1:80/ccp-webapp/ccp/campaign/MyTestCampaign
HTTP Response Headers:	HTTP/1.1 200 OK Content-Type: text/plain Content-Length: 0 Date: Tue, 12 Jan 2010 17:03:54 GMT

5.1.4. get

[contents]

Return the data for a single campaign.

URL:	http://server:port/ccp-webapp/ccp/campaign/<publicId>
HTTP Method:	GET
Example XML Response:	Response includes a *metrics/socialContactCount* element that contains the number of results included in the campaign. <Campaign> <changeStamp>1</changeStamp> <refURL> http://192.168.0.1:80/ccp-webapp/ccp/campaign/MyTestCampaign </refURL> <link> http://192.168.0.1:80/ccp-webapp/ccp/campaign/MyTestCampaign/results

```
        </link>
        <name>MyTestCampaign</name>
        <description>This is my test campaign</description>
        <publicId>MyTestCampaign</publicId>
        <includeExpr>Cisco Expert Advisor</includeExpr>
        <excludeExpr>ICM</excludeExpr>
        <feeds>
               <feed>http://192.168.0.1:80/ccp-webapp/ccp/feed/5000</feed>
               <feed>http://192.168.0.1:80/ccp-webapp/ccp/feed/5001</feed>
        </feeds>
        <filters>
               <filter>http://192.168.0.1:80/ccp-webapp/ccp/feed/6000</filter>
               <filter>http://192.168.0.1:80/ccp-webapp/ccp/feed/6001</filter>
        </filters>
        <metrics>
               <socialContactCount>12</socialContactCount>
        </metrics>
        <suggestedTagsURL>
               http://192.168.0.1/ccp-webapp/ccp/campaign/MyTestCampaign/suggestedtags
        </suggestedTagsURL>
 </Campaign>
```

HTTP Response Headers:

```
HTTP/1.1 200 OK
Content-Type: application/xml
Transfer-Encoding: chunked
Date: Tue, 12 Jan 2010 16:55:05 GMT
```

5.1.5. list

[contents]

List all campaigns.

URL:	http://server:port/ccp-webapp/ccp/campaign/
HTTP Method:	GET
URL Parameter:	• summary - true/false (optional, defaults to "false"). When "true", only a subset of the XML elements are returned.
Example XML Response:	Response includes a *metrics/socialContactCount* element that contains the number of results included in the campaign.

```
<Campaigns>
       <Campaign>
              <refURL>
                     http://192.168.0.1:80/ccp-webapp/ccp/campaign/100004
              </refURL>
              <name>MyTestCampaign</name>
              <description>This is my test campaign</description>
              <publicId>MyTestCampaign</publicId>
              <changeStamp>0</changeStamp>
              <feeds>
                     <feed>http://192.168.0.1:80/ccp-webapp/ccp/feed/100000</feed>
                     <feed>http://192.168.0.1:80/ccp-webapp/ccp/feed/100001</feed>
              </feeds>
              <metrics>
                     <socialContactCount>12</socialContactCount>
              </metrics>
       </Campaign>
       <Campaign>
              <refURL>http://192.168.0.1:80/ccp-webapp/ccp/campaign/100005</refURL>
              <name>MyTestCampaign2</name>
              <description>This is my test campaign</description>
              <publicId>MyTestCampaign2</publicId>
              <changeStamp>0</changeStamp>
              <feeds>
                     <feed>http://192.168.0.1:80/ccp-webapp/ccp/feed/100000</feed>
                     <feed>http://192.168.0.1:80/ccp-webapp/ccp/feed/100001</feed>
              </feeds>
              <metrics>
                     <socialContactCount>345</socialContactCount>
              </metrics>
```

```
        </Campaign>
</Campaigns>
```

HTTP Response Headers:

```
HTTP/1.1 200 OK
Content-Type: application/xml
Transfer-Encoding: chunked
Date: Tue, 12 Jan 2010 16:59:22 GMT
```

5.1.6. suggestedtags

[contents]

Retrieve the suggested tags for social contacts in this campaign. Up to ten tags are returned based on how recent and how often a tag has been used in this campaign.

URL:	http://server:port/ccp-webapp/ccp/campaign/<publicId>/suggestedtags
HTTP Method:	get
Example XML Response:	The first 10 suggested tags are returned

```
<SuggestedTags>
      <tags>
            <tag>fresh</tag>
            <tag>cool</tag>
            <tag>slide</tag>
      </tags>
</SuggestedTags>
```

6. The Campaign Results API

[contents]

The SocialMiner Campaign Results API allows you to get the results for a campaign.

The URL to access the API is **http://server:port/ccp-webapp/campaign/results/**.

6.1. Commands

[contents]

You can perform the following commands:

6.1.1. get

[contents]

Get results for the specified campaign.

URL:	http://server:port/ccp-webapp/ccp/campaign/<publicId>/results
HTTP Method:	GET
URL Parameters:	• **postId**: String (optional) If provided, results are displayed started after the provided postId. Cannot be used if timestamp or firstResultIndex is specified. • **numberOfResults**: Integer (optional) The maximum number of results to be returned. Default is 50 and maximum is 200.

- **timestamp**: Integer (optional) Display results older results than this timestamp. Defaults to the time of request if not provided. If firstResultIndex is not specified then timestamp assumes firstResultIndex = 0.
- **firstResultIndex:** Integer (optional) The number of results to skip based on the timestamp. Used for pagination. Assuming numberOfResults is set at the default of 50, then you could create a "page 2" link by using the timestamp provided in the href of the `feed/link rel="self"` and a firstResultIndex of 50. Page 3 would use the same timestamp and a firstResultIndex of 100, etc. See the code in Example Response for an example.
- **filterStatus**: String (optional) . Display Social Contacts whose status matches a status within this field. Defaults to all if the parameter is not specified. You must specify a value for the parameter if the parameter is included or no social contacts are returned. Can be one or more of (case-insensitive):
  - RESERVED
  - HANDLED
  - UNREAD
  - DISCARDED
  - Multiple status example: http://192.168.0.1/ccp-webapp/ccp/campaign/Business_News/results**?filterStatus=RESERVED&filterStatus=HANDLED**
- **filterTag**: String (optional). Display Social Contacts whose filter(s) matches one or more of the tags in this field. Defaults to all tags if not specified. Example: http://192.168.0.1/ccp-webapp/ccp/campaign/Business_News/results**?filterTag=tag1&filterTag=tag2**
- **Notes:**
  - If timestamp is provided and firstResultIndex is not provided, then the results are displayed up to the"numberOfResults" with a creation date older than "timestamp", starting at index 0.
  - If timestamp is not provided and firstResultIndex is provided, then the results are displayed up to the "numberOfResults" with a creation date older than "now" starting at firstResultIndex.
  - If postId is provided, then the contact identified by the postId is used as the basis for the search. The social contact for the provided postId does not appear in the results.

Example Response:

Results are returned as an ATOM 1.0 feed with the following elements:

- **feed/title**: The name of the Campaign.
- **feed/link rel = self & feed/id**: The URL of the results.
- **feed/link rel = countsince:** The URL for the API call for the number of new social contacts since this result was retrieved.
- **feed/link rel = next:** The URL to the next 50 results. Present only when there are more than 50 results left in the campaign.
- **entry/title**: The title of the social contact.
- **entry/link rel = alternate & entry/id**: The URL to the social contact.
- **entry/link rel = socialcontact:** The URL for the API call of this social contact.
- **entry/summary**: The content of the social contact.
- **entry/published**: The date and time that the social contact was published in the form of *YYYY-MM-DD*T*HH:MM:SS*Z. If the social contact did not contain a published date then this is the date when the social contact was read by SocialMiner.
- **entry/ccp:statustimestamp:** The timestamp of the last change to the status of the social contact.
- **entry/ccp:status:** The current status of the social contact.
- **entry/ccp:sctags/ccp:sctag** One or more tags associated with this social contact.
- **entry/ccp:scstatususerid** The last user to change the status of this social contact. If blank and the status is unread then this social contact has never had a status change.

```
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:ccp="http://www.cisco.com/ccbu/ccp/xml/socialcontact/1.0/" xmlns:dc="http://purl.org/dc/elements/1.1/">
       <title>Business News</title>
       <link rel="self" href="http://192.168.0.1/ccp-webapp/ccp/campaign/Business_News/results?timestamp=1278696169003" />
       <link rel="countsince" href="http://192.168.0.1/ccp-webapp/ccp/campaign/Business_News/count?postId=22E00F5310000129460A1EB40A568DDE" />
       <link rel="next" href="http://192.168.0.1/ccp-webapp/ccp/campaign/JingCampaign1/results?timestamp=1276525157775&amp;firstResultIndex=50" />
       <subtitle>This feed has been created by Cisco Customer Collaboration Platform</subtitle>
       <id>http://192.168.0.1/ccp-webapp/ccp/campaign/Business_News/results</id>
       <updated>2010-06-10T17:20:46Z</updated>
       <dc:date>2010-06-10T17:20:46Z</dc:date>
       <entry>
```

```
                <title>SEC OKs 'flash crash' fix</title>
                <link rel="alternate" href="http://rss.cnn.com/~r/rss/money_latest/~3/h0MxRXFew9I/index.htm" />
                <link rel="socialcontact" href="http://192.168.0.1/ccp-webapp/ccp/socialcontact/22E00F5310000129460A1EB40A568DDE" />
                <author>
                        <name />
                </author>
                <id>http://rss.cnn.com/~r/rss/money_latest/~3/h0MxRXFew9I/index.htm</id>
                <updated>2010-06-10T17:10:50Z</updated>
                <published>2010-06-10T17:10:50Z</published>
                <summary type="html">The Securities and Exchange Commission approved new rules Thursday that will halt trading uniformly across all U.S. markets for stocks experiencing wild price swings to prevent a repeat of last
month's "flash crash."&lt;img src="http://feeds.feedburner.com/~r/rss/money_latest/~4/h0MxRXFew9I" height="1" width="1"/&gt;</summary>
                <dc:creator />
                <dc:date>2010-06-10T17:10:50Z</dc:date>
                <ccp:scstatustimestamp>1283819058417</ccp:scstatustimestamp>
                <ccp:scstatus>reserved</ccp:scstatus>
                <ccp:scstatususerid>admin</ccp:scstatususerid>
                <ccp:sctags>
                        <ccp:sctag>sometag</ccp:sctag>
                        <ccp:sctag>anothertag</ccp:sctag>
                        <ccp:sctag>yetanothertag</ccp:sctag>
                </ccp:sctags>
        </entry>
        <entry> ... </entry>
        <entry> ... </entry>
        <entry> ... </entry>
</feed>
```

7. The Campaign Results Count API

[contents]

The SocialMiner Campaign Results Count API allows you to get a count of the results for a specified campaign. You can get a count of the results for the entire campaign or the count of results since a given post.

The URL to access the Campaign Results Count API is **http://server:port/ccp-webapp/ccp/campaign/<publicId>/count**.

7.1. Campaign Results Count API Commands

[contents]

You can perform the following command:

- get

7.1.1. get

[contents]

Get the number of results in an entire campaign or since a given postId.

URL:	http://server:port/ccp-webapp/ccp/campaign/<publicId>/count
HTTP Method:	GET
URL Parameter:	• **postId**: (optional) The postId of the last post seen. This id is the unique id of the campaign results. A link to this API with the appropriate id in the url is included in the results atom feed. The count displays the number of results published after the referenced campaign result. If no postId is provided, the full number of results in the campaign is returned. The postId can be found in the campaign results, for example: `<link rel="countsince" href="http://192.168.0.100/ccp-webapp/ccp/campaign/MyTestCampaign/count?postId=92517B6610000128295CEBB40A568DDE" />`
Example XML Response:	`<count>44</count>`

8. The Reply Template API

[contents]

The SocialMiner Reply Template API allows you to add, edit, and delete the name a location of reply templates. Reply templates are used to respond to Twitter Direct Messages sent to a Twitter Account feed type. Reply templates allow you to respond to the Tweet from within SocialMiner, instead of having to open a new browser window or use a third party application to access Twitter.

The URL to access the Reply Template API is **http://server:port/ccp-webapp/ccp/campaign/template/reply**

8.1. Reply Template API Commands

[contents]

8.1.1. create

[contents]

Create a new template definition.

URL:	http://server:port/ccp-webapp/ccp/campaign/template/reply
HTTP Method:	POST
Parameters:	• **name** - The name for your template. • **templateURL** - The URL of your template. The URL must reference an OpenSocial gadget to be displayed in SocialMiner. Additional information on OpenSocial is available at http://code.google.com/apis/opensocial/.

Example XML Request Payload:	`<Template>`
	`<name>My  Template</name>`
	`<templateURL>http://this.is.my.template.url/template.html</templateURL>`
	`</Template>`

Example using cURL:	`curl -i -H "Content-type: application/xml" -X POST -d "<Template><name>Fake 2</name><templateURL>http://www.example.com/reply-template-url.html</templateURL></Template>" http://user:password@192.168.0.1/ccp-webapp/ccp/template/reply`
Response:	Status: 201 Created

8.1.2. delete

[contents]

Delete a reply template definition.

URL:	http://server:port/ccp-webapp/ccp/campaign/template/reply/<id>
HTTP Method:	DELETE

HTTP Response Headers:	Status 200: OK

8.1.3. list

[contents]

List all reply templates stored on this system.

URL:	http://server:port/ccp-webapp/ccp/campaign/template/reply/
HTTP Method:	GET

Example Response:	`<Templates>`
	`  <Template>`
	`  <changeStamp>0</changeStamp>`
	`  <name>Cisco Twitter</name>`
	`  <refURL>http://192.168.0.1/ccp-webapp/ccp/template/reply/300</refURL>`
	`  <systemDefined>true</systemDefined>`
	`  <templateURL>/gadgets/files/ccp/templates/reply/cisco_twitter.jsp</templateURL>`
	`  </Template>`
	`  <Template>`
	`  <changeStamp>0</changeStamp>`
	`  <name>My Test Template</name>`
	`  <refURL>http://192.168.0.1/ccp-webapp/ccp/template/reply/100024</refURL>`
	`  <systemDefined>false</systemDefined>`
	`  <templateURL>http://this.is.my.template.url/template.html</templateURL>`
	`  </Template>`
	`</Templates>`

8.1.4. get

[contents]

Get the details for one reply template.

URL:	http://server:port/ccp-webapp/ccp/campaign/template/reply/<id>
HTTP Method:	GET

Example XML Response:	`<Template>`
	`  <changeStamp>1</changeStamp>`
	`  <name>Remote Custom Template</name>`
	`  <refURL>http://192.168.0.1/ccp-webapp/ccp/template/reply/105673</refURL>`
	`  <systemDefined>false</systemDefined>`
	`  <templateURL>http://192.168.0.1/ccp-reply/twitter-reply-gadget.jsp</templateURL>`
	`</Template>`
	• **changeStamp**: The change stamp of the reply template.
	• **name**: The name of the reply template.
	• **refURL**: The reference to the reply template.
	• **systemDefined**: True if the template was pre-installed on SocialMiner. systemDefined templates cannot be deleted.
	• **templateURL**: The URL of the reply template.

8.1.5. update

[contents]

Update an existing reply template definition.

URL:	http://server:port/ccp-webapp/ccp/campaign/template/reply/<id>
HTTP Method:	PUT
Parameters:	• **name** - string, required. The name for your template. • **templateURL** - string, required. The URL of your template. The URL must reference an OpenSocial gadget to be displayed in SocialMiner. Additional information on OpenSocial is available at http://code.google.com/apis/opensocial/. • **changeStamp** - integer, required.
The changeStamp:	**Important:** You must provide the current changeStamp of the reply template when you perform an update. If you do not provide the current changestamp then the update fails. This mechanism is in place so that two clients cannot edit the reply template at the same time. The changeStamp increments by 1 if the update is successful.

Example XML Request Payload:	`<Template>`
	`  <changeStamp>1</changeStamp>`
	`  <name>Remote Custom Template</name>`
	`  <templateURL>http://192.168.0.1/ccp-reply/twitter-reply-gadget.jsp</templateURL>`
	`</Template>`

Example using cURL:	`curl -i -H "Content-type: application/xml" -X PUT -d "<Template><name>Fake 2</name><templateURL>http://www.example.com/reply-template-url.html</templateURL></Template>" http://user:password@192.168.0.1/ccp-webapp/ccp/template/reply/110472`

8.2. Creating Reply Templates

[contents]

This section describes how to create a reply template for use with Twitter Account feeds.

9. Twitter Reply API

[contents]

The Twitter Reply API allows you to respond to Tweets or Twitter Direct Messages. You must first configured a Twitter Account Feed before you can use this API.

Note: The Shindig OpenSocial container in which SocialMiner runs requires that REST requests complete within five seconds. However, communication with the Twitter servers can exceed five seconds. This limitation means you must poll after making calls to the Twitter Reply API to verify the status returned. A diagram is provided to illustrate the API calls and expected poll responses.

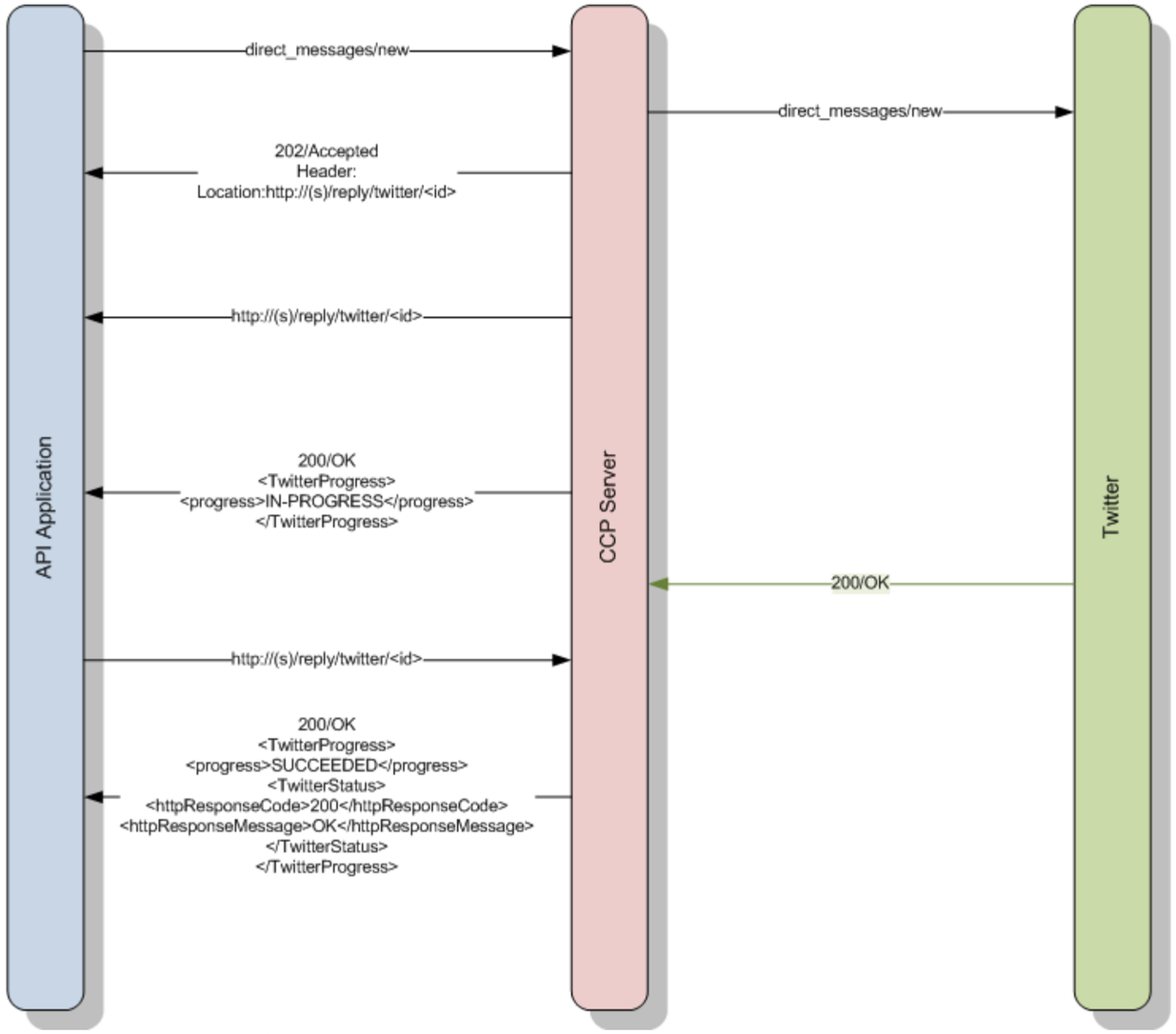

Figure 1: Twitter Follow API Flow Diagram

9.1. Twitter Reply API Commands

[contents]

9.1.1. get

[contents]

Get the status of a Twitter Reply API call.

URL:	http://server:port/ccp-webapp/ccp/reply/twitter/<id>

HTTP Method:	GET

Example Response:	<TwitterProgress>
	  <progress>SUCCEEDED|FAILED|IN-PROGRESS</progress>
	  <TwitterStatus></TwitterStatus>
	  <httpResponseCode>responseCode</httpResponseCode>
	  <httpResponseMessage>responseMessage</httpResponseMessage>
	</TwitterProgress>
	
	The response includes the following fields:
	• **progress**: one of the following
	  ◦ **SUCCEEDED**: The operation succeeded.
	  ◦ **FAILED**: The Twitter operation failed. Use `httpResponseCode` and `httpResponseMessage` to determine why the operation failed.
	  ◦ **IN-PROGRESS**: SocialMiner is waiting for a response from Twitter.
	• **TwitterStatus**: Varies depending on the API called. See examples for the individual Twitter Reply API methods.
	• **httpResponseCode**: the response code received from Twitter.
	• **httpResponseMessage**: the response message received from Twitter.

9.1.2. get user

[contents]

Retrieve the profile information of a given Twitter user name. Works similar to get.

URL:	http://server:port/ccp-webapp/ccp/reply/twitter/users/show
HTTP Method:	GET
Parameters:	• screenName: The user name of the Twitter account.
Example HTTP Request:	http://192.168.0.1/ccp-webapp/ccp/reply/twitter/users/show?screenName=ccpdoctest
Example XML Response:	If the fields in the initial request are valid, then the response header's location field contains the URL to poll for the status of the operation. Otherwise, SocialMiner returns an error. See HTTP errors for error responses. Pragma No-cache Cache-Control no-cache Expires Wed, 31 Dec 1969 19:00:00 EST Location http://192.168.0.1/ccp-webapp/ccp/reply/twitter/8 Content-Type application/xml Content-Length 0 Date Mon, 14 Feb 2011 22:32:59 GMT If the operation succeeds, the polling URL (`http://192.168.0.1/ccp-webapp/ccp/reply/twitter/8`) contains the following XML. <TwitterProgress> <progress>SUCCEEDED</progress> <twitterUser> <screenName>twitter_user</screenName> <id>1234567890</id> <name>A. Twitter User</name> <description></description>

```
    <profileImageUrl>http://a2.twimg.com/profile_images/60201051/WileECoyote_normal.jpg</profileImageUrl>
    <url>http://www.mycompany.com/</url>
    <followersCount>10</followersCount>
  </twitterUser>
  <httpResponseCode>responseCode</httpResponseCode>
  <httpResponseMessage>responseMessage</httpResponseMessage>
</TwitterProgress>
```

9.1.3. create status (tweet)

[contents]

Send a twitter status message (tweet) from a configured twitter account.

URL:	http://server:port/ccp-webapp/ccp/reply/twitter/statuses/new
HTTP Method:	POST
Parameters:	• username: The username of a configured twitter account. The twitter account must be configured on the system. • message: The text of the message. Limited to 140 characters.

Example XML Request Payload:	`<Status>`
	`  <username>twitterScreenName</username>`
	`  <message>Tweet text</message>`
	`</Status>`

Example using cURL:	`curl -i -H "Content-type: application/xml" -X POST -d "<Status><username>UserName_From_Twitter_Account_Feed</username><message>tweet test api</message></Status>" http://user:password@192.168.0.1/ccp-webapp/ccp/reply/twitter/statuses/update`

Response:	If the fields in the initial request are valid, then the response header's location field contains the URL to poll for the status of the operation. Otherwise, SocialMiner returns an error. See HTTP errors for error responses.
	If the operation succeeds, the polling URL contains the following XML.
	`<TwitterProgress>`
	`  <progress>SUCCEEDED</progress>`
	`  <TwitterStatus>`
	`  <id>idAssignedByTwitter</id>`
	`  <text>textOfTheTweet</text>`
	`  <user>`
	`  <screenName>usersScreenName</screenName>`
	`  <id>userIdAssignedByTwitter</id>`
	`  </user>`
	`  <createdAt>tweetCreateDate</createdAt>`
	`  <inReplyToScreenName>screenNameOfReplyRecipient</inReplyToScreenName>`
	`  <inReplyToStatusId>idOfOriginalTweet</inReplyToStatusId>`
	`  <inReplyToUserId>idOfReplyRecipient</inReplyToUserId>`
	`  </TwitterStatus>`
	`  <httpResponseCode>responseCode</httpResponseCode>`
	`  <httpResponseMessage>responseMessage</httpResponseMessage>`
	`</TwitterProgress>`
	If the operation fails, the httpResponseCode and httpResponseMessage fields contains the code and message returned by Twitter.

9.1.4. create direct message

[contents]

Send a Twitter Direct Message (DM) from a configured twitter account.

URL:	http://server:port/ccp-webapp/ccp/reply/twitter/direct_messages/new
HTTP Method:	POST
Parameters:	• fromUsername: The configured Twitter Account username for the user sending the message. • toUsername: The twitter username of the recipient of the DM. • message: The text of the message. Limited to 140 characters.

Example XML Request Payload:	`<DirectMessage>`
	`  <fromUsername>TwitterAccountUsernam</fromUsername>`
	`  <toUsername>someTwitterAccount</toUsername>`
	`  <message>This is a test of the create direct message API.</message>`
	`</DirectMessage>`

Example using cURL:	`curl -i -H "Content-type: application/xml" -X POST -d "<DirectMessage><fromUsername>UserName_From_Twitter_Account_Feed</fromUsername><toUsername>TwitterUser</toUsername><message>DM Test API</message></DirectMessage>" http://user:password@192.168.0.1/ccp-webapp/ccp/reply/twitter/direct_messages/new`

Response:	If the fields in the initial request are valid, then the response header's location field contains the URL to poll for the status of the operation. Otherwise, SocialMiner returns an error. See HTTP errors for error responses.
	If the operation succeeds, the polling URL contains the following XML.
	`<TwitterProgress>`
	`  <progress>SUCCEEDED</progress>`
	`  <TwitterStatus>`
	`  <id>idAssignedByTwitter</id>`
	`  <text>textOfTheTweet</text>`
	`  <createdAt>tweetCreateDate</createdAt>`
	`  <senderScreenName>senderScreenName</senderScreenName>`
	`  <recipientScreenName>recipientScreenName</recipientScreenName>`
	`  </TwitterStatus>`
	`  <httpResponseCode>responseCode</httpResponseCode>`
	`  <httpResponseMessage>responseMessage</httpResponseMessage>`
	`</TwitterProgress>`
	If the operation fails, the httpResponseCode and httpResponseMessage fields contains the code and message returned by Twitter.

9.1.5. create retweet

[contents]

Retweet a Twitter Social Contact from a configured Twitter account.

URL:	http://server:port/ccp-webapp/ccp/reply/twitter/statuses/retweet
HTTP Method:	POST
Parameters:	• username: The username of the configured Twitter Account sending the Retweet. • tweetId: the Twitter ID of the tweet.

Example XML Request Payload:	<Status> <username>twitterScreenName</username> <tweetId>Tweet text</tweetId> </Status>
Example using cURL:	curl -i -H "Content-type: application/xml" -X POST -d "<Status><username>twitterScreenName</username><tweetId>Tweet text</tweetId></Status>" http://user:password@192.168.0.1/ccp-webapp/ccp/reply/twitter/statuses/retweet
Response:	If the fields in the initial request are valid, then the response header's location field contains the URL to poll for the status of the operation. Otherwise, SocialMiner returns an error. See HTTP errors for error responses. If the operation succeeds, the polling URL contains the following XML. <TwitterProgress> <progress>SUCCEEDED</progress> <TwitterStatus> <id>idAssignedByTwitter</id> <text>textOfTheTweet</text> <user> <screenName>usersScreenName</screenName> <id>userIdAssignedByTwitter</id> </user> <createdAt>tweetCreateDate</createdAt> <inReplyToScreenName>screenNameOfReplyRecipient</inReplyToScreenName> <inReplyToStatusId>idOfOriginalTweet</inReplyToStatusId> <inReplyToUserId>idOf ReplyRecipient</inReplyToUserId> </TwitterStatus> <httpResponseCode>responseCode</httpResponseCode> <httpResponseMessage>responseMessage</httpResponseMessage> </TwitterProgress> If the operation fails, the httpResponseCode and httpResponseMessage fields contains the code and message returned by Twitter.

9.1.6. get follow

[contents]

Determine if one Twitter user is following another Twitter user.

URL:	http://server:port/ccp-webapp/ccp/reply/twitter/follow
HTTP Method:	GET
Parameters:	• **user_a**: the screen name of the user who may or may not be following user_b • **user_b**: the screen name of the user who may or may not be followed by user_a • **account_user**: this parameter is optional. If specified, the account_user must be associated with a Twitter Account feed configured on SocialMiner. If this is the case, the 'get follow' call is made to Twitter using the account_user's oAuth credentials.
Example:	http://server:port/ccp-webapp/ccp/reply/twitter/follow?user_a=USERNAME_A&user_b=USERNAME_B
Response:	If the fields in the initial request are valid, then the response header's location field contains the URL to poll for the status of the operation. Otherwise, SocialMiner returns an error. See HTTP errors for error response. If the operation succeeds, the polling URL contains the following XML.

```
<TwitterProgress>
       <progress>SUCCEEDED</progress>
       <TwitterStatus>
              <friends>true|false</friends>
       </TwitterStatus>
       <httpResponseCode>responseCode</httpResponseCode>
       <httpResponseMessage>responseMessage</httpResponseMessage>
</TwitterProgress>
```

If the operation fails, the httpResponseCode and httpResponseMessage fields contains the code and message returned by Twitter.

9.1.7. create follow

[contents]

Follow a user from a configured Twitter Account feed.

URL:	http://server:port/ccp-webapp/ccp/reply/twitter/follow
HTTP Method:	POST
Parameters:	This method uses post but uses query parameters to specify who to follow and who will follow: • **account_user**: the twitter account feed user. This username must match one of the user names in a SocialMiner Twitter Account feed. • **user_to_follow**: the screen name of the Twitter user to follow

Response:	If the fields in the initial request are valid, then the response header's location field contains the URL to poll for the status of the operation. Otherwise, SocialMiner returns an error. See HTTP errors for error response.
	If the operation succeeds, the polling URL contains the following XML.
	`<TwitterProgress>`
	`  <progress>SUCCEEDED</progress>`
	`  <httpResponseCode>responseCode</httpResponseCode>`
	`  <httpResponseMessage>responseMessage</httpResponseMessage>`
	`</TwitterProgress>`
	If the operation fails, the httpResponseCode and httpResponseMessage fields contains the code and message returned by Twitter.

9.1.8. create unfollow

[contents]

Stop following a user from a configured Twitter Account feed.

URL:	http://server:port/ccp-webapp/ccp/reply/twitter/unfollow
HTTP Method:	POST
Parameters:	This method uses post but uses query parameters to specify who to follow and who will follow: • **account_user**: the twitter account feed user. This username must match one of the user names in a SocialMiner Twitter Account feed. • **user_to_unfollow**: the screen name of the Twitter user to stop following.

Response:	If the fields in the initial request are valid, then the response header's location field contains the URL to poll for the status of the operation. Otherwise, SocialMiner returns an error. See HTTP errors for error response.
	If the operation succeeds, the polling URL contains the following XML.
	`<TwitterProgress>`
	`  <progress>SUCCEEDED</progress>`
	`  <httpResponseCode>responseCode</httpResponseCode>`
	`  <httpResponseMessage>responseMessage</httpResponseMessage>`
	`</TwitterProgress>`
	If the operation fails, the httpResponseCode and httpResponseMessage fields contains the code and message returned by Twitter.

10. The Conversation API

[contents]

The Conversation API allows you to view the conversational context of a specific Twitter social contact. If a Twitter social contact is in reply-to another tweet stored in SocialMiner, then this conversation can be retrieved. The API extracts all Twitter social contacts in a reply "chain".

Note: Twitter Direct Messages (DMs) do are not linked to each other by Twitter, so the conversation aspect of DMs can not be retrieved.

10.1. Conversation API Commands

[contents]

You can perform the following commands:

- list

10.1.1. list

[contents]

List the Twitter conversation stored in SocialMiner.

URL:	http://server:port/ccp-webapp/ccp/conversation/<SocialContactID>/?replies=xxx Where: • <SocialContactID> is the ID of the social contact, and • xxx is the number of replies to the Social Contact to return, for example 100.
HTTP Method:	GET
URL Parameters:	• SocialContactID: string, the ID of the Social Contact • replies: integer
Example HTTP Request:	`https://192.168.0.1/ccp-webapp/ccp/conversation/25FD59151000012E001FFC6B0A568DF5?replies=3`

Example XML Response:	`<SocialContacts>`
	`  <SocialContact>`
	`  <author>userA</author>`
	`  <description>@userB Tell me more about lolz</description>`
	`  <id>25FD59151000012E001FFC6B0A568DF5</id>`
	`  <link>http://twitter.com/userA/statuses/37255608721735680</link>`
	`  <publishedDate>1297717403000</publishedDate>`
	`  <refURL>https://192.168.0.1/ccp-webapp/ccp/socialcontact/25FD59151000012E001FFC6B0A568DF5</refURL>`
	`  <replyToId>37255525456412673</replyToId>`
	`  <status>reserved</status>`
	`  <statusTimestamp>1297717634793</statusTimestamp>`

```
    <statusUserId>admin</statusUserId>
    <tags/>

    <title>Tweet: userA reply to userB</title>
  </SocialContact>
  <SocialContact>
    <author>userB</author>
    <description>@userA lolz</description>
    <id>25FD5B4F1000012E001FFC710A568DF5</id>
    <link>http://twitter.com/userB/statuses/37255525456412673</link>

    <publishedDate>1297717384000</publishedDate>
    <refURL>https://192.168.0.1/ccp-webapp/ccp/socialcontact/25FD5B4F1000012E001FFC710A568DF5</refURL>
    <replyToId>37255476928323584</replyToId>
    <status>unread</status>
    <statusTimestamp>1297717486783</statusTimestamp>
    <statusUserId></statusUserId>

    <tags/>
    <title>Tweet: userB reply to userA</title>
  </SocialContact>
  <SocialContact>
    <author>userA</author>
    <description>@userB Hahaha, I had you dude! You thought I was a bot!</description>
    <id>25FD59151000012E001FFC6C0A568DF5</id>

    <link>http://twitter.com/userA/statuses/37255476928323584</link>
    <publishedDate>1297717372000</publishedDate>
    <refURL>https://192.168.0.1/ccp-webapp/ccp/socialcontact/25FD59151000012E001FFC6C0A568DF5</refURL>
    <replyToId>37255377833832448</replyToId>
    <status>unread</status>
    <statusTimestamp>1297717486749</statusTimestamp>

    <statusUserId></statusUserId>
    <tags/>
    <title>Tweet: userA reply to userB</title>
  </SocialContact>
</SocialContacts>
```

HTTP Response Headers:	A *200 OK* HTTP header is returned on success.

11. The Filter API

[contents]

The SocialMiner Filter API allows you create, update, and delete filters.

11.1. Filter API Commands

[contents]

You can perform the following commands:

11.1.1. create

[contents]

Create a new filter.

URL:	http://server:port/ccp-webapp/ccp/filter
HTTP Method:	POST
Parameters:	• name: string (unique, required)

	• description: string (optional)
	• type: integer (required) The type of filter. Must be one of:
	  ◦ 2 = Bayesian - use a trainable bayesian filter.
	  ◦ 3 = Author - filter by the author of the social contact.
	• keywords/keyword: string, required when type = 3, one or more keyword elements that contain the author information (for example, twitter username) that is to be filtered.
	• rule: integer, required when type = 3, values can be:
	  ◦ 1 = exclude: exclude this author from the campaign results.

Example XML Request Payload:	`<Filter>`
	`  <name>String</name>`
	`  <description>String</description>`
	`  <type>3</type>`
	`<rule>1</rule>`
	`<keywords>`
	`  <keyword>Author to Exclude</keyword>`
	`</keywords>`
	`</Filter>`

HTTP Response Headers:	The response contains the URL for the newly created filter.
	`HTTP/1.1 201 Created`
	`Location: http://192.168.0.1:80/ccp-webapp/ccp/filter/1266345862276`
	`Content-Type: text/plain`
	`Content-Length: 0`
	`Date: Tue, 16 Feb 2010 19:35:56 GMT`

11.1.2. delete

[contents]

Delete an existing filter.

URL:	http://server:port/ccp-webapp/ccp/filter/<id>
HTTP Method:	DELETE
Example using cURL:	`curl -i -X DELETE http://user:password@192.168.0.1:80/ccp-webapp/ccp/filter/100171`

HTTP Response Headers:	`HTTP/1.1 200 OK`
	`Content-Type: text/plain`
	`Content-Length: 0`
	`Date: Tue, 12 Jan 2010 17:03:54 GMT`

11.1.3. list

[contents]

List all filters.

URL:	http://server:port/ccp-webapp/ccp/filter
HTTP Method:	GET
URL Parameter:	summary - true/false (optional, defaults to "false"). When "true", only a subset of the XML elements are returned.
Example:	`http://192.168.0.1:80/ccp-webapp/ccp/filter?summary=false`

Example XML Response:	`<Filters>`
	`<Filterref>`
	`<url>`
	`http://localhost:8080/ccp-webapp/rest/ccp/filter/101246`
	`</url>`
	`<Filter>`
	`<name>wayne10</name>`
	`<description>xyz</description>`
	`<type>1</type>`
	`<changeStamp>12345</changeStamp>`
	`</Filter>`
	`</Filterref>`
	`<Filterref>`
	`<url>`
	`http://localhost:8080/ccp-webapp/rest/ccp/filter/101247`
	`</url>`
	`<Filter>`
	`<name>wayne11</name>`
	`<description>xyz</description>`
	`<type>1</type>`
	`<changeStamp>12345</changeStamp>`
	`</Filter>`
	`</Filterref>`
	`</Filters>`

HTTP Response Headers:	`HTTP/1.1 200 OK`
	`Content-Type: application/xml`
	`Transfer-Encoding: chunked`
	`Date: Tue, 12 Jan 2010 16:47:58 GMT`

11.1.4. get

[contents]

Return the data for a single filter.

URL:	http://server:port/ccp-webapp/ccp/filter/<id>
HTTP Method:	GET
Example:	`http://192.168.0.1:80/ccp-webapp/ccp/filter/100036`

Example XML Response:	`<Filter>`
	`<name>wayne10</name>`
	`<description>xyz</description>`
	`<type>1</type>`

	<changeStamp>12345</changeStamp> </Filter>
HTTP Response Headers:	HTTP/1.1 200 OK Content-Type: application/xml Transfer-Encoding: chunked Date: Tue, 12 Jan 2010 16:50:46 GMT

11.1.5. update

[contents]

Update an existing filter.

URL:	http://server:port/ccp-webapp/ccp/filter/<id>
HTTP Method:	PUT

Parameters:	• name: (unique, required) The name of the filter.
	• description: (optional) A description of the filter
	• type: (optional) The type of filter. Must be one of:
	  ◦ 2 = Bayesian
	  ◦ 3 = Author
	• keywords/keyword: string, required when type = 3, one or more keyword elements that contain the author information (for example, twitter username) that is to be filtered.
	• rule: integer, required when type = 3, values can be:
	  ◦ 1 = exclude: exclude this author from the campaign results.

The changeStamp:	**Important**: You must provide the current changeStamp of the filter when you perform an update. If you do not provide the current changestamp then the update fails. This mechanism is in place so that two clients cannot edit the filter at the same time. The changeStamp increments by 1 if the update is successful.
Example XML Request Payload:	<Filter> <name>String</name> <description>String</description> <type>1</type> <changeStamp>12346</changeStamp> </Filter>
Example using cURL:	curl -i -H "Content-type: application/xml" -X PUT -d "<Filter><name>TestName5</name><description>Edited Description</description><type>1</type><changeStamp>12346</changeStamp></Filter>" http://user:password@192.168.0.1/ccp-webapp/ccp/filter/103114
HTTP Response Headers:	HTTP/1.1 200 OK Content-Type: text/plain Content-Length: 0 Date: Thu, 14 Jan 2010 15:49:17 GMT

12. The Filter Results API

[contents]

The SocialMiner Filter Results API allows you to get the results for a filter.

The URL to access the API is **http://server:port/ccp-webapp/ccp/filter/<id>/results**.

12.1. Filter Results API Commands

[contents]

- get

12.1.1. get

[contents]

Get results for the specified filter.

URL:	http://server:port/ccp-webapp/ccp/filter/<id>/results
HTTP Method:	GET
URL Parameter:	**document**: (optional) String. The text for the filter to analyze.

Example XML Response:	Results are returned as XML with the following elements:
	• FilterResult: The container for the result.
	• refURL: The URL of the filter results request.
	• result: The result of the filter analysis expressed as an integer from 1 - 100.
	• document: The text passed to the filter for analysis.
	`<FilterResult>`
	`<refURL>http://192.168.0.10/ccp-webapp/ccp/filter/1001234/results</refURL>`
	`<result>88</result>`
	`<document>The text that was passed to the filter to analyze</document>`
	`</FilterResult>`

13. The Social Contact API

[contents]

Social contacts are the individual results obtained by campaigns. The SocialMiner Social Contact API allows you to get and update an individual social contact.

The URL to access the API is **http://server:port/ccp-webapp/ccp/socialcontact/<id>** .

The status of a social contact is global across all campaigns.

13.1. Social Contact API Commands

[contents]

- create (push using POST)
- create (push using GET)
- get
- update

13.1.1. create (push using POST)

[contents]

Create a new Social Contact.

URL:	http://server:port/ccp-webapp/ccp/socialcontact/
HTTP Method:	POST
Parameters:	• **feedRefURL** : string, required - the Ref URL of a push feed to which this social contact is associated. • **title** : string, required - The title of the social contact. • **publishedDate** : string - the social contact published date. Leave blank to use the current timestamp or provide a valid Unix timestamp. • **author** : string, required - the social contact author name. • **description** : string - the body of the social contact. • **tag/tags** : string, one or more tags to assign to this social contact. The tags can be new or existing tags.

Example XML Request:	Results are returned as XML.
	`<SocialContact>`
	`<feedRefURL>http://192.168.0.1/ccp-webapp/ccp/feed/(id)</feedRefURL>`
	`<title>social contact title</title>`
	`<publishedDate></publishedDate>`
	`<author>Customer_Tony</author>`
	`<description type="html"> This is the content of the social contact. Perhaps it is a tweet or a blog post. Supports HTML encoding. </description>`
	`<tags>`
	`<tag>tag1</tag>`
	`<tag>tag2</tag>`
	`</tags>`
	`</SocialContact>`

Example using cUrl:	`curl -i -H "Content-type: application/xml" -X POST -d "<SocialContact><feedRefURL>http://192.168.0.1/ccp-webapp/ccp/feed/100006</feedRefURL><title>social contact title</title><publishedDate></publishedDate><author>Customer_Tony</author><description type='html'>This is the content of the social contact. Perhaps it was a tweet or a blog post.</description><tags><tag>tag1</tag><tag>tag2</tag></tags></SocialContact>" http://admin:password@192.168.0.1/ccp-webapp/ccp/socialcontact`
HTTP Response:	A *201 Created* HTTP header is returned on success along with a URL to the newly created social contact.

13.1.2. create (push using GET)

[contents]

In order to prevent cross-site scripting (XSS) errors, a GET API call can be used to "push" a new social contact into the system. The push mechanism is only available for feeds that are of the Push type.

URL:	http://server:port/ccp-webapp/ccp/pushfeed/<feedId>/push
HTTP Method:	GET
Parameters:	• **title** : string, required - The title of the social contact.

	author : string, required - the social contact author name. • **description** : string - the body of the social contact. • **tags** : string, a comma=separated list of tags to apply to this social contact.
Example using a URL:	`http://192.168.0.1/ccp-webapp/ccp/pushfeed/100006/push?title=This+is+the+Title&author=John+Smith&description=This+is+the+description&tags=tag3,tag4`

13.1.3. get

[contents]

Get results for the specified social contact. The <id> attribute required for this command is found in the Campaign Results, in the feed/entry/link rel="socialcontact" element. For example:
<link rel="socialcontact" href="http://192.168.0.1/ccp-webapp/ccp/socialcontact/22E00F5310000129460A1EB40A568DDE" />

URL:	http://server:port/ccp-webapp/ccp/socialcontact/<id>
HTTP Method:	GET

Example Response:	Results are returned as XML.
	`<SocialContact>`
	`<refURL>http://192.168.0.1/ccp-webapp/ccp/socialcontact/E98F880E1000012A4D1E0CAA0A568DDE</refURL>`
	`<status>reserved</status>`
	`<statusTimestamp>1283819058417</statusTimestamp>`
	`<tags>`
	`<tag>slide</tag>`
	`</tags>`
	`<statusUserId>admin</statusUserId>`
	`<publishedDate>DATE</publishedDate>`
	`<replyTemplateRefURL>http://192.168.0.1/ccp-webapp/ccp/template/reply/105678</replyTemplateRefURL>`
	`<replyTemplateURL>http://192.168.0.1/gadgets/files/ccp/templates/reply/cisco_twitter.jsp</replyTemplateURL>`
	`</SocialContact>`
	If *statusUserId* is blank and the status is *unread* then this social contact has never had a status change.
	If the social contact is associated with a feed that supports Reply Templates, then the *replyTemplateRefURL* and *replyTemplateURL* fields are included. These fields can not be changed by the Social Contact API.

13.1.4. update

[contents]

Update the status or tags of a specified social contact.

Social contacts can have the following statuses:

- **Unread** - The default state of a new social contact.
- **Reserved** - Reserved to be handled.
- **Handled** - This social contact has been handled and no further action is required.
- **Discarded** - This social contact does not require a response and is filed in the recycle bin.

You can also add, edit, or remove tags using the update command.

URL:	http://server:port/ccp-webapp/ccp/socialcontact/<id>
HTTP Method:	PUT

Parameters:	• **statusTimestamp**: long integer (required)
	• **status**: string (required, case-sensitive) One of:
	  ◦ **unread** - The default state of a new social contact.
	  ◦ **reserved** - Reserved to be handled.
	  ◦ **handled** - This social contact has been handled and no further action is required.
	  ◦ **discarded** - This social contact does not require a response and is filed in the recycle bin.
	• **tag/tags** - (optional) one or more *tags* to associate with this social contact. If you include the *tags* element, but do not include any existing *tag* elements, then tags are deleted during an update.
	• **statusUserId** - (optional) The user ID of the user modifying the status. This value cannot be changed using this element. The value changes to the user who is currently authenticated against the API.

The statusTimestamp:	**Important**: You must provide the current statusTimestamp of the social contact when you perform an update. If you do not provide the same statusTimestamp as returned from a social contact **get** request, then the update fails. This mechanism is in place so that two clients cannot update the same social contact at the same time. The statusTimestamp changes to the current timestamp if the update is successful.
Example XML Request Payload:	<SocialContact> <statusTimestamp>1276008213</statusTimestamp> <status>Reserved</status> <statusUserId>admin</statusUserId> <tags> <tag>cool</tag> <tag>fresh</tag> </tags> <replyTemplateRefURL>http://192.168.0.1/ccp-webapp/ccp/template/reply/105678</replyTemplateRefURL> <replyTemplateURL>http://192.168.0.1/gadgets/files/ccp/templates/reply/cisco_twitter.jsp</replyTemplateURL> </SocialContact> If the social contact is associated with a feed that supports Reply Templates, then the *replyTemplateRefURL* and *replyTemplateURL* fields are included. These fields can not be changed by the Social Contact API.
Example using cURL:	curl -i -H "Content-type: application/xml" -X PUT -d "<SocialContact><status>reserved</status><statusTimestamp>1276190396724</statusTimestamp><tags><tag>cool</tag><tag>fresh</tag></tags></SocialContact>" http://user:password@192.168.0.1/ccp-webapp/ccp/socialcontact/22E00F5310000129460A1EB40A568DDE
Example XML Response:	<SocialContact> <refURL>http://192.168.0.1/ccp-webapp/ccp/socialcontact/22E00F5310000129460A1EB40A568DDE</refURL> <status>reserved</status> <statusTimestamp>1276190792688</statusTimestamp> <tags> <tag>cool</tag> <tag>fresh</tag> </tags> <statusUserId>admin</statusUserId> </SocialContact>

14. The Bayesian Filter Training API

[contents]

The SocialMiner Bayesian Filter Training API allows you to train a specified Bayesian filter to indicate if a document is a match for the filter.

The URL to access the API is **http://server:port/ccp-webapp/ccp/filter/<id>/train**.

14.1. Bayesian Filter Training API Commands

[contents]

- train
- delete

14.1.1. train

[contents]

Train the specified filter ID with a document and a true/false indication of the document is a match for the filter.

URL:	http://server:port/ccp-webapp/ccp/filter/<id>/train
HTTP Method:	PUT
Parameters:	• **socialContact**: string (required if document is not specified) The URL of the social contact. • **document**: string (required if socialcontact is not specified) The text on which to train the filter. • **match**: boolean (required) "True" or "False", Indication whether the document is a match for the filter (required)

Example XML Request Payload using socialContact:	`<TrainingRequest>`
	`  <socialContact>http://192.168.0.1/ccp-webapp/ccp/socialcontact/B83B18F4100001292B3D088D0A568DDE</socialContact>`
	`  <match>True</match>`
	`</TrainingRequest>`

Example XML Request Payload using document:	`<TrainingRequest>`
	`  <document>This is very positive. I really like it. Performance was excellent. Great product</document>`
	`  <match>True</match>`
	`</TrainingRequest>`

Example using cURL with socialContact:	`curl -i -H "Content-type: application/xml" -X PUT -d "<TrainingRequest><socialContact>http://192.168.0.1/ccp-webapp/ccp/socialcontact/B83B18F4100001292B3D088D0A568DDE</socialContact><match>True</match></TrainingRequest>" http://admin:password@192.168.0.1/ccp-webapp/ccp/filter/106430/train`
Example using cURL with document:	`curl -i -H "Content-type: application/xml" -X PUT -d "<TrainingRequest><document>This is very positive. I really like it. Performance was excellent. Great product</document><match>True</match></TrainingRequest>" http://admin:password@192.168.0.1/ccp-webapp/ccp/filter/106430/train`

HTTP Response Headers:	`HTTP/1.1 200 OK`
	`Pragma: No-cache`
	`Cache-Control: no-cache`
	`Expires: Wed, 31 Dec 1969 19:00:00 EST`
	`Set-Cookie: JSESSIONIDSSO=58AEE69D45227D9FE1704D18F9C72913; Path=/`
	`Set-Cookie: JSESSIONID=98504C52667551FFF276F885628BC3B9; Path=/ccp-webapp`
	`Content-Type: text/plain`
	`Content-Length: 0`
	`Date: Mon, 14 Jun 2010 14:13:09 GMT`
	`Server:`

14.1.2. delete

[contents]

Delete all training data for a filter.

URL:	http://server:port/ccp-webapp/ccp/filter/<id>/trainingdata
HTTP Method:	DELETE
Example using cURL:	`curl -i -X DELETE http://user:password@192.168.0.1:80/ccp-webapp/ccp/filter/100171/trainingdata`

HTTP Response Headers:	`HTTP/1.1 200 OK`
	`Content-Type: text/plain`
	`Content-Length: 0`
	`Date: Mon, 14 Jun 2010 14:22:30 GMT`

15. The Tag API

[contents]

SocialMiner supports the labeling of social contacts with tags. Tags can be added/edited/removed to or from a social contact using the Social Contact API.

The URL to access the API is **http://server:port/ccp-webapp/ccp/tag**.

15.1. Tag API Commands

[contents]

- list

15.1.1. list

[contents]

List all tags that exist.

URL:	http://server:port/ccp-webapp/ccp/tag
HTTP Method:	GET

Example XML Response:	`<Tags>`
	`  <Tag>`
	`  <name>tagname1</name>`
	`  </Tag>`
	`  <Tag>`
	`  <name>tagname2</name>`
	`  </Tag>`
	`  ...`
	`</Tags>`

16. The Serviceability API

[contents]

The Serviceability API provides information on various SocialMiner service, feed statuses, and version information.

The URL to access the API is **http://server:port/ccp-webapp/ccp/serviceability/**.

16.1. Serviceability API Commands

[contents]

Serviceability API Commands:

- list
- get
- feed statusDescription Values
- serverState Values
- Notifier Status Values

16.1.1. list

[contents]

List all of the available Serviceability attributes and their current values. See the get method for details.

URL:	http://server:port/ccp-webapp/ccp/serviceability/
HTTP Method:	GET

Example XML Response:	`<Serviceability>`
	`<diskUsage>`
	`<activePartitionTotalBytes>13463810048</activePartitionTotalBytes>`
	`<activePartitionUsableBytes>2352652288</activePartitionUsableBytes>`
	`<commonDatastorePartitionTotalBytes>46835601408</commonDatastorePartitionTotalBytes>`
	`<commonDatastorePartitionUsableBytes>37941018624</commonDatastorePartitionUsableBytes>`
	`<inactivePartitionTotalBytes>13463781376</inactivePartitionTotalBytes>`
	`<inactivePartitionUsableBytes>2357100544</inactivePartitionUsableBytes>`
	`</diskUsage>`
	`<feedStatuses>`
	`<FeedStatus>`
	`<feedRefURL>http://10.86.141.251/ccp-webapp/ccp/feed/100000</feedRefURL>`
	`<lastFetchCount>20</lastFetchCount>`
	`<statusDescription>NORMAL</statusDescription>`
	`</FeedStatus>`
	`</feedStatuses>`
	`<serviceStates>`
	`<datastoreServerState>SERVER_STATE_IN_SERVICE</datastoreServerState>`
	`<indexerServerState>SERVER_STATE_IN_SERVICE</indexerServerState>`
	`<runtimeServerState>SERVER_STATE_IN_SERVICE</runtimeServerState>`
	`</serviceStates>`
	`<systemConditions/>`
	`<version>`
	`<buildDate>Sat Oct 30 18:38:41 EDT 2010</buildDate>`
	`<buildVersion>165</buildVersion>`
	`<esVersion>0</esVersion>`
	`<maintenanceVersion>1</maintenanceVersion>`
	`<majorVersion>8</majorVersion>`
	`<minorVersion>5</minorVersion>`
	`<srVersion>0</srVersion>`
	`<vosActiveVersion>8.5.0.97000-93</vosActiveVersion>`
	`<vosInactiveVersion>8.5.0.97000-92</vosInactiveVersion>`
	`</version>`
	`</Serviceability>`

16.1.2. get

[contents]

Get the value for a given Serviceability attribute.

URL:	http://server:port/ccp-webapp/ccp/serviceability?category=<*attributeName*>&category=<*attributeName*> <*attributeName*> can be one of: • **diskUsage** - display the amount of storage used. • **feedStatuses** - get the statuses of just feeds. See feed statusDescription Values. • **serviceStates** - get the state of services. See serverState Values • **systemConditions** - get a list of any system conditions.

	• **version** - get system version information. • **systemInfo** - get all available system parameters. **Important!** Accessing this category forces the system to dump all system parameters into a large XML file. System performance is greatly inhibited while the snapshot is created. • **systemPerformance** - gets the number of socialContactsPerHour. • **notifiers** - The status of the notification services for this SocialMiner server. See Notifier Status Values
HTTP Method:	GET

Example XML Response:	Example response for http://192.168.0.1/ccp-webapp/ccp/serviceability/?category=serviceStates
	`<serviceStates>`
	`<datastoreServerState>SERVER_STATE_IN_SERVICE</datastoreServerState>`
	`<indexerServerState>SERVER_STATE_IN_SERVICE</indexerServerState>`
	`<runtimeServerState>SERVER_STATE_IN_SERVICE</runtimeServerState>`
	`</serviceStates>`

16.1.3. feed statusDescription Values

[contents]

Values for a feed's statusDescription can be:

- **NORMAL**: The feed is operating normally.
- **SCHEDULED**: The feed has been scheduled but has not yet been executed. Feeds are in this state for a very short period of time and then either go to NORMAL or an error state.
- **NETWORK_TIMEOUT**: The remote server was reachable but did not respond to the request in a timely manner.
- **NETWORK_NOT_REACHABLE**: Could not connect to the remote server.
- **NETWORK_ERROR**: Unhandled network error.
- **UNKNOWN_ERROR**: An error occurred that does not have a specific exception.
- **DATASTORE_ERROR**: Error attempting to write the social contacts to the datastore.
- **UNSUPPORTED_FEED_CONTENT**: The content retrieved from the feed is not in a format that SocialMiner supports.
- **TWITTER_STREAM_INTERRUPTED**: The thread that runs that Twitter stream client was interrupted.
- **TWITTER_STREAM_CONNECTED**: The Twitter stream feed has connected to Twitter, but has not received any contacts yet.
- **TWITTER_STREAM_MALFORMED_URL**: An exception occurred when connecting to Twitter.
- **TWITTER_STREAM_CONNECT_ERROR**: There was an error while connecting to the Twitter stream. Requests to Twitter are automatically slowed exponentially to reestablish the connection.
- **TWITTER_STREAM_READ_ERROR**: There was an error while reading to the Twitter stream. Requests to Twitter are automatically slowed linearly.
- **TWITTER_STREAM_STREAM_DISCONNECTED**: A Twitter Stream Feed is subscribed, but not connected to Twitter.
- **TWITTER_STREAM_EOF**: Twitter sent an End Of File to close the stream.
- **TWITTER_STREAM_NO_CLIENT**: The Twitter feed checked to see if there were any contacts but no stream client was found for this feed.
- **AUTHENTICATION_FAILED**: An authenticated feed failed because of incorrect credentials.
- **FACEBOOK_PARSE_ERROR**: JSON returned from Facebook did not parse correctly. This may be due to an unknown change in the Facebook API.

16.1.4. serverState Values

[contents]

Values for serverState can be:

- **SERVER_STATE_API_INIT**: the Serviceability API is initializing. This is an ephemeral state when the API is first started.
- **SERVER_STATE_UNREACHABLE**: the API can't connect to the Runtime, Datastore, or Indexer Server to check the state, either because the service is down/stopped or because of other errors.
- **SERVER_STATE_IN_SERVICE**: the Runtime, Datastore, or Indexer server is in service.

- **SERVER_STATE_PARTIAL_SERVICE**: the Runtime, Datastore, or Indexer server is waiting for another component or sub-component to start or recover from an error. No new social contacts are returned when the service is in partial service.

16.1.5. Notifier Status Values

[contents]

Values for notifier status can be:

- **type**: The type of notification sent by this notifier. For example, *email* or *im*.
- **notificationsSent**: The number of notification successfully sent by this notifier.
- **notificationsFailed**: The number of failed notifications.
- **notificationsDropped**: The number of notifications that were dropped because the output queue was full.
- **outQueueDepth**: The number of items in the output queue.
- **outQueueWait**: The average amount of time in milliseconds between when a notification request is queued and when it is sent.
- **ConnectionStatus**: Can be one of:
  - *CONNECTED*: The notifier successfully connected to the configured server.
  - *DISCONNECTED*: The notifier is unable to connect to the configured server.
  - *DISABLED*: The notifier is not enabled.
  - *BAD_CONFIGURATION*: The notifier is not configured correctly and is unable to attempt to make a connection.

17. The Authentication API

[contents]

The authentication API allows you to configure a connection to a Microsoft Active Directory (AD) server. You can specify all users who exist on an AD to have access to SocialMiner or you can specify a single group of AD users.

The URL to access the API is **http://server:port/ccp-webapp/ccp/authentication/**.

17.1. Authentication API Commands

[contents]

17.1.1. update

[contents]

update the authentication information from SocialMiner.

URL:	http://server:port/ccp-webapp/ccp/authentication/
HTTP Method:	PUT

Example XML Payload:	<pre><Authentication>
	  <enabled>true</enabled>
	  <primaryHost>ad.server</primaryHost>
	  <primaryPort>3268</primaryPort>
	  <primaryUseSSL>false</primaryUseSSL>
	  <managerDistinguishedName>cn=admin,ou=users,dc=ad,dc=server</managerDistinguishedName>
	  <managerPassword>password</managerPassword>
	  <roleName>CCP_Users</roleName>
	</Authentication></pre> Parameters:
	• **enabled**: true/false. Indicates if the authentication settings are used when trying to authenticate a user. If this is disabled then only the application administrator will have

access to the system.

- **primaryHost**: Required if enabled. The host address of the AD server.
- **primaryPort**: Required if enabled. The host port.
- **primaryUseSSL**: true/false. Indicates if a secure connection should be established. This requires that a domain certificate be uploaded to the server and that the primaryPort allows secure connections..
- **managerDistinguishedName**: Required if enabled. The distinguished name of a user that has manager access to the AD server. For example CN=Administrator, CN=users, DC=MYSERVER, DC=COM.
- **managerPassword**: Required if enabled. The password of the user specified in the managerDistinguishedName field.
- **roleName**: Optional. The name of an AD role or group. All users in this AD role or group are provided access to SocialMiner. Users on the AD who are not members of this role or group are not provided access to SocialMiner. Blank or * indicates that all users in the AD are allowed to use the application.

17.1.2. get

[contents]

Get the authentication information from CCP.

URL:	http://server:port/ccp-webapp/ccp/authentication/
HTTP Method:	GET

Example XML Response:	`<Authentication>`
	`<enabled>true</enabled>`
	`<managerDistinguishedName>CN=administrator,CN=users,DC=ccbu-doc-ad,DC=cisco,DC=com</managerDistinguishedName>`
	`<managerPassword>********</managerPassword>`
	`<primaryHost>10.86.132.145</primaryHost>`
	`<primaryPort>3268</primaryPort>`
	`<primaryUseSSL>false</primaryUseSSL>`
	`<refURL>http://192.168.0.1/ccp-webapp/ccp/authentication</refURL>`
	`<roleName></roleName>`
	`</Authentication>`
	Parameters:
	• **enabled**: true/false. Indicates if the authentication settings are used when trying to authenticate a user. If this is disabled then only the application administrator will have access to the system.
	• **primaryHost**: Required if enabled. The host address of the AD server.
	• **primaryPort**: Required if enabled. The host port.
	• **primaryUseSSL**: true/false. Indicates if a secure connection should be established. This requires that a domain certificate be uploaded to the server and that the primaryPort allows secure connections. **If set to true, then you must follow the instructions in** Enabling SSL for Active Directory Authentication
	• **managerDistinguishedName**: Required if enabled. The distinguished name of a user that has manager access to the AD server. For example CN=Administrator, CN=users, DC=MYSERVER, DC=COM.
	• **managerPassword**: Required if enabled. The password of the user specified in the managerDistinguishedName field.
	• **roleName**: Optional. The name of an AD role or group. All users in this AD role or group are provided access to CCP. Users on the AD who are not members of this role or group are not provided access to CCP. Blank or * indicates that all users in the AD are allowed to use the application.

17.2. Enabling SSL for Active Directory Authentication

[contents]

You can enable secure authentication (SSL) against a Microsoft Active Directory server by exchanging the SocialMiner certificate with the AD server.

To enable SSL for the Active Directory connection:

On the Active Directory Server:

1. Verify that the Active Directory has the Certificate Services service installed.
2. Select **All Programs > Administrative Tools > Certificate Authority**.
3. Expand the domain node and select **Issued Certificates**.
4. Double click the certificate to open it
5. Open the Details tab and click **Copy to file**.
6. An Export wizard appears. In the wizard select DER encoded binary.
7. Using the wizard to select a location to save the file.
8. Click **Finish**.

On the SocialMiner Server:

1. Open the *Cisco Unified Operating System Administration* page. The link is available from the administration gadget.
2. Select **Security > Certificate Management**.
3. Click **Upload Certificate**.
4. For the Certificate Name, select **tomcat-trust**.
5. In the Upload File field, select the file to upload by clicking **Browse...** Select the certificate file you saved from the Active Directory server.
6. Click **Upload File**.
7. Restart the Cisco Tomcat service. Using the CLI, run the command `utils service restart Cisco Tomcat`.

18. The Reporting User API

[contents]

The reporting user API allows you to create the reporting user and set the reporting user password. The reporting user username is *reportinguser* and it not editable.

Note: Only the administrator created during install can use this API.

The URL to access the API is **http://server:port/ccp-webapp/ccp/reportinguser**.

18.1. Reporting User API Commands.

[contents]

Reporting User API Commands:

18.1.1. create

[contents]

Creates the reporting user.

Note: The userName of the reporting user must be *reportinguser* or an error is thrown.

URL:	http://server:port/ccp-webapp/ccp/reportinguser/
HTTP Method:	POST

Parameters:	• **userName**: string, (required) - must be **reportinguser**. • **password**: string - the password for the reporting user.

Example XML Request Payload:	`<ReportingUser>`
	`  <userName>reportinguser</userName>`
	`  <password>password</password>`
	`</ReportingUser>`

Example using cURL:	`curl -i -H "Content-type: application/xml" -X POST -d "<ReportingUser><username>reportinguser</username><password>password</password></ReportingUser>" http://admin:Samb123@10.86.141.251/ccp-webapp/ccp/reportinguser`
HTTP Response Headers:	A *201 Created* HTTP header is returned on success.

18.1.2. delete

[contents]

Delete a reporting user.

URL:	http://server:port/ccp-webapp/ccp/reportinguser/<id>
HTTP Method:	DELETE
Example using cURL:	`curl -i -X DELETE http://admin:Samb123@10.86.141.251/ccp-webapp/ccp/reportinguser/100000`
HTTP Response Headers:	A *200 OK* HTTP header is returned on success.

18.1.3. list

[contents]

List all reporting users.

Note: Currently only a single reporting user is supported.

URL:	http://server:port/ccp-webapp/ccp/reportinguser
HTTP Method:	GET

Example XML Response:	`<ReportingUsers>`
	`  <ReportingUser>`
	`  <refURL>http://192.168.0.1/ccp-webapp/ccp/reportinguser/100001</refURL>`
	`  <userName>reportinguser</userName>`
	`  </ReportingUser>`
	`</ReportingUsers>`

18.1.4. get

[contents]

Get a single reporting user.

URL:	http://server:port/ccp-webapp/ccp/reportinguser/<id>
HTTP Method:	GET
Example Response:	<ReportingUser> <refURL>http://10.86.141.251/ccp-webapp/ccp/reportinguser/100001</refURL> <userName>reportinguser</userName> </ReportingUser>

18.1.5. update

[contents]

Update a reporting user. Currently, only the password can be changed.

URL:	http://server:port/ccp-webapp/ccp/reportinguser/<id>
HTTP Method:	PUT
Parameters:	• **password**: string - the new password for the reporting user.
Example XML Request Payload:	<ReportingUser> <userName>reportinguser</userName> <password>newpassword</password> </ReportingUser>
Example using cURL:	curl -i -H "Content-type: application/xml" -X PUT -d "<ReportingUser><username>reportinguser</username><password>newpassword</password></ReportingUser>" http://admin:Samb123@10.86.141.251/ccp-webapp/ccp/reportinguser/1000001
HTTP Response Headers:	A *200 OK* HTTP header is returned on success.

19. The Reporting Server API

[contents]

The Reporting Server API returns the Reporting Server database connection information.

The URL to access the API is **http://server:port/ccp-webapp/ccp/reportingserver**.

19.1. Reporting Server API Commands

[contents]

Reporting Server API Commands:

- get

19.1.1. get

[contents]

Get the reporting server database connection information.

URL:	http://server:port/ccp-webapp/ccp/reportingserver (requires administrator privileges)
HTTP Method:	get

Example XML Response:	<pre><ReportingServer>
	  <databaseType>Informix</databaseType>
	  <reportingDatabase>mmca_data</reportingDatabase>
	  <reportingHost>ccp-223-rt</reportingHost>
	  <reportingHostIp>10.86.141.223</reportingHostIp>
	  <reportingPort>1526</reportingPort>
	  <reportingServer>ccp_223_rt_mmca</reportingServer>
	</ReportingServer></pre>
	• **databaseType**: The database server type. Returns "Informix".
	• **reportingDatabase**: The name of the reporting database.
	• **reportingHost**: The host name of the reporting server.
	• **reportingHostIp**: The IP address of the reporting server.
	• **reportingPort**: The connection port for the reporting server.
	• **reportingServer**: The reporting-server informix instance name.

20. The Purge API

[contents]

The Purge API allows you to change settings associated with the database purge feature. Routine database purging is necessary to prevent the file system from filling up.

The URL to access the API is **http://server:port/ccp-webapp/ccp/purge**.

20.1. Purge API Commands

[contents]

Purge API Commands:

- update
- list

20.1.1. update

[contents]

Update the purge settings.

URL:	http://server:port/ccp-webapp/ccp/purge
HTTP Method:	PUT
Parameters:	

	dataStorePurgeAge: (required) - The age of the social contacts that will be purged. Valid values are 1 to 30. • **dataStoreEmergencyPurgeDiskUsage**: (required) - The percent of disk usage that acts as a purge threshold. When this threshold is reached a purge starts. Social Contacts older than dataStorePurgeAge are removed first. If disk usage is still above the threshold for emergency purging, then the purge continues removing social contacts (one day at a time) until the disk usage is below the threshold for emergency purge. Valid values are 60 - 90. • **reportingPurgeTime**: (required) - The ages of reporting records that will be purged. Valid values are 1 to 550. • **reportingPurgeAge**: (required) - The time, in 24 hour format (HH:mm), when the purge is to start. Valid values are 00:00 to 23:59.

Example XML Request Payload:	`<PurgeConfig>`
	`<dataStoreEmergencyPurgeDiskUsage>80</dataStoreEmergencyPurgeDiskUsage>`
	`<dataStorePurgeAge>30</dataStorePurgeAge>`
	`<reportingPurgeAge>550</reportingPurgeAge>`
	`<reportingPurgeTime>01:00</reportingPurgeTime>`
	`</PurgeConfig>`

Example using cURL:	`curl -i -H "Content-type: application/xml" -X PUT -d "<PurgeConfig><dataStoreEmergencyPurgeDiskUsage>85</dataStoreEmergencyPurgeDiskUsage><dataStorePurgeAge>25</dataStorePurgeAge><reportingPurgeAge>500</reportingPurgeAge><reportingPurgeTime>04:00</reportingPurgeTime></PurgeConfig>" http://admin:Samb123@10.86.141.251/ccp-webapp/ccp/purge`
HTTP Response Headers:	A *200 OK* HTTP header is returned on success.

20.1.2. list

[contents]

List the current purge settings.

URL:	http://server:port/ccp-webapp/ccp/purge
HTTP Method:	GET

Example XML Response:	`<PurgeConfig>`
	`<dataStoreEmergencyPurgeDiskUsage>80</dataStoreEmergencyPurgeDiskUsage>`
	`<dataStorePurgeAge>30</dataStorePurgeAge>`
	`<reportingPurgeAge>550</reportingPurgeAge>`
	`<reportingPurgeTime>01:00</reportingPurgeTime>`
	`</PurgeConfig>`

21. The Email API

[contents]

The Email API allows you to configure a single SMTP server connection. An SMTP server connection is required to send email notifications.

The URL to access the API is **http://server:port/ccp-webapp/ccp/email/default**.

21.1. Email API Commands

[contents]

Email API Commands:

- get
- update

21.1.1. get

[contents]

Get the SMTP configuration.

URL:	http://server:port/ccp-webapp/ccp/email/default (requires administrator privileges)
HTTP Method:	GET

Example XML Response:	<pre><Email>
	  <smtpHost>10.86.141.293</smtpHost>
	  <smtpPort>587</smtpPort>
	  <smtpFromUser>FromUser@Here.net</smtpFromUser>
	  <smtpHostUserName>userNameForEmailServer</smtpHostUserName>
	  <smtpAuthenticationEnabled>true</smtpAuthenticationEnabled>
	  <smtpEnabled>true</smtpEnabled>
	  <refURL>http://10.86.141.223/ccp-webapp/ccp/email/default</refURL>
	</Email></pre>
	• **smtpHost**: The IP address or hostname of the SMTP server.
	• **smtpPort**: The SMTP port. The standard SMTP port is 25
	• **smtpFromUser**: The email (reply-to) address of email sent by this SocialMiner server.
	• **smtpHostUserName**: The username used to log into the SMTP server.
	• **smtpAuthenticationEnabled**: Whether SMTP authentication is required for the SMTP host.
	• **smtpEnabled**: Whether this SMTP configuration is enabled.

21.1.2. update

[contents]

Update the SMTP configuration.

URL:	http://server:port/ccp-webapp/ccp/email/default (requires administrator privileges)
HTTP Method:	PUT
Parameters:	• **smtpHost**: The fully qualified host address of the SMTP server. (required when smtpEnabled = true) • **smtpPort**: The SMTP port. Default is 587. (required when smtpEnabled = true) • **smtpFromUser**: The email (reply-to) address of email sent by this SocialMiner server. (required when smtpEnabled = true) • **smtpHostUserName**: The username used to log into the SMTP server.(required when smtpAuthenticationEnabled = true) • **smtpHostUserPassword**: The password used to log into the SMTP server.(required when smtpAuthenticationEnabled = true) • **smtpAuthenticationEnabled**: Whether SMTP authentication is required for the SMTP host. (optional, boolean) • **smtpEnabled**: Whether this SMTP configuration is enabled. (optional, boolean, defaults to false)

22. The XMPP API

[contents]

The XMPP API allows you to configure an XMPP Server connection. An XMPP server connection is required to send Instant Messaging (IM) notifications.

The URL to access the API is **http://server:port/ccp-webapp/ccp/xmpp/default**.

22.1. XMPP API Commands

[contents]

XMPP API Commands:

22.1.1. get

[contents]

Get the XMPP configuration.

URL:	http://server:port/ccp-webapp/ccp/xmpp/default (requires administrator privileges)
HTTP Method:	GET
Example XML Response:	<Xmpp> <refURL>http://10.86.141.251/ccp-webapp/ccp/xmpp/default</refURL> <xmppEnabled>true</xmppEnabled> <xmppHost></xmppHost> <xmppPort>5222</xmppPort> <xmppService>im.example.com</xmppService> <xmppServiceLookup>true</xmppServiceLookup> <xmppServicePassword>*******</xmppServicePassword> <xmppServiceUsername>fred</xmppServiceUsername> </Xmpp> • **xmppHost**: string - The IP address or hostname of the XMPP server. • **xmppPort**: integer - The XMPP Port. The default port is 5222. This field is not used if *xmppServiceLookup* is set to *true*. • **xmppService**: string - The host name of the XMPP service if using XMPP service lookup. • **xmppServiceLookup**: true/false - Whether this XMPP service is enabled for • **xmppServiceUserName**: string - The username used to log into the XMPP server. • **xmppEnabled**: true/false - Whether this XMPP configuration is enabled.

22.1.2. update

[contents]

Update the XMPP configuration.

URL:	http://server:port/ccp-webapp/ccp/xmpp/default (requires administrator privileges)
HTTP Method:	PUT
Parameters:	• **xmppHost**: string - The IP address or hostname of the XMPP server. • **xmppPort**: integer - The XMPP Port. The default port is 5222. This field is not used if *xmppServiceLookup* is set to *true*. • **xmppService**: string - The host name of the XMPP service if using XMPP service lookup. • **xmppServiceLookup**: true/false - Whether this XMPP service is enabled for

xmppServiceUserName: string - The username used to log into the XMPP server.

- **xmppEnabled**: true/false - Whether this XMPP configuration is enabled.

23. The Notification Rule API

[contents]

The Notification Rule API allows you to configure notifications that are sent when a specific tag is added to a social contact in a specific campaign.

Note: When using email notifications, you must first configure an Email (SMTP) Server before notification can be sent through email or configure an XMPP server before IM notifications can be sent.

The URL to access the API is **http://server:port/ccp-webapp/ccp/notificationrule**.

23.1. Notification API Commands

[contents]

Notification Rule API Commands:

23.1.1. create

[contents]

Create a new notification rule.

URL:

http://server:port/ccp-webapp/ccp/notificationrule

HTTP Method:

POST

Parameters:

- **name**: string, (required) - the name of the notification rule.
- **description**: string - the password for the reporting user.
- **campaignUrl**: The URL of the campaign.
- **tags/tag**: string - A single tag. When this tag is applied to a social contact in the specified campaign, the notification rule is activated and a notification is sent.
- **targets/target**: string - on or more targets, with a maximum of ten, to which the notification rule is sent.
- **type**: string - can be either *email* or *IM*.
- **subject**: string, max 255 characters - The subject of a notification rule message. Not used for *IM* notifications.
- **body**: string, max 2048 characters - The body of the notification rule message. The link to the social contact is automatically inserted after the body.

Example XML Request Payload:

```
 <NotificationRule>
   <body>New Contact:</body>
   <campaignUrl>http://1192.168.0.1/ccp-webapp/ccp/campaign/Pushed_Contacts</campaignUrl>
   <name>Push</name>
   <subject>Notification: New Push Tag applied to Pushed Contacts Campaign</subject>
   <tags>
       <tag>push</tag>
   </tags>
   <targets>
       <target>user@example.com</target>
       <target>user2@example.com</target>
   </targets>
   <type>
</NotificationRule>email</type>
```

Example using cURL:

```
curl -i -H "Content-type: application/xml" -X POST -d "<NotificationRule><body>New Contact:</body><campaignUrl>http://192.168.0.1/ccp-webapp/ccp/campaign/Pushed_Contacts</campaignUrl><name>Push 2</name><subject>Notification: New tag applied to
pushed contacts Campaign</subject><tags><tag>push2</tag></tags><targets><target>user@example.com</target></targets><type>email</type></NotificationRule>" http://admin:password@192.168.0.1/ccp-webapp/ccp/notificationrule
```

HTTP Response Headers:

A *201 Created* HTTP header is returned on success as well as the REST URL to the new notification rule.

23.1.2. delete

[contents]

Delete a notification rule.

URL:

http://server:port/ccp-webapp/ccp/notificationrule/<id>

HTTP Method:

DELETE

Example using cURL:

```
curl -i -X DELETE http://admin:password@192.168.0.1/ccp-webapp/ccp/notificationrule/100011
```

HTTP Response Headers:

A *200 OK* HTTP header is returned on success.

23.1.3. list

[contents]

List all notification rule.

URL:

http://server:port/ccp-webapp/ccp/notificationrule

HTTP Method:

GET

Example using cURL:

```
curl -i -X GET http://admin:password@192.168.0.1/ccp-webapp/ccp/notificationrule
```

Example XML Response:

```
<NotificationRules>
  <NotificationRule>
     <body>New Contact:</body>
     <campaignUrl>http://192.168.0.1/ccp-webapp/ccp/campaign/Pushed_Contacts</campaignUrl>
     <changeStamp>0</changeStamp>
     <name>Push</name>
     <refURL>http://10.86.141.251/ccp-webapp/ccp/notificationrule/100010</refURL>
     <subject>Notification: New Push Tag applied to Pushed Contacts Campaign</subject>
     <tags>
        <tag>push</tag>
     </tags>
     <targets>
        <target>user@example.com</target>
     </targets>
     <type>email</type>
  </NotificationRule>
  <NotificationRule>
    ....
  </NotifactionRule>
</NotificationRules>
```

HTTP Response Headers:

A *200 OK* HTTP header is returned on success.

23.1.4. get

[contents]

Get a specific notification rule.

URL:
http://server:port/ccp-webapp/ccp/notificationrule/<id>

HTTP Method:
GET

Example using cURL:

```
curl -i -X GET http://admin:password@192.168.0.1/ccp-webapp/ccp/notificationrule/100011
```

Example XML Response:

```
<NotificationRule>
   <body>New Contact:</body>
   <campaignUrl>http://192.168.0.1/ccp-webapp/ccp/campaign/Pushed_Contacts</campaignUrl>
   <changeStamp>0</changeStamp>
   <name>Push</name>
   <refURL>http://10.86.141.251/ccp-webapp/ccp/notificationrule/100010</refURL>
   <subject>Notification: New Push Tag applied to Pushed Contacts Campaign</subject>
   <tags>
       <tag>push</tag>
   </tags>
   <targets>
       <target>user@example.com</target>
   </targets>
   <type>email</type>
</NotificationRule>
```

HTTP Response Headers:
A *200 OK* HTTP header is returned on success.

23.1.5. update

[contents]

Update an existing notification rule.

URL:
http://server:port/ccp-webapp/ccp/notificationrule/<id>

HTTP Method:
PUT

Parameters:

- **name**: string, (required) - the name of the notification rule.
- **description**: string - the password for the reporting user.
- **campaignUrl**: The URL of the campaign.
- **tags/tag**: string - A single tag. When this tag is applied to a social contact in the specified campaign, the notification rule is activated and a notification is sent.
- **targets/target**: string - on or more targets, with a maximum of ten, to which the notification rule is sent.
- **type**: string - can be either *email* or *IM*.
- **subject**: string, max 255 characters - The subject of a notification rule message. Not used for *IM* notifications.
- **body**: string, max 2048 characters - The body of the notification rule message. The link to the social contact is automatically inserted after the body.

The changeStamp:
Important: You must provide the current changeStamp of the notification rule when you perform an update. If you do not provide the current changestamp then the update fails. This mechanism is in place so that two clients cannot edit the notification rule at the same time.

The changeStamp increments by 1 if the update is successful.

Example XML Request Payload:

```
<NotificationRule>
  <body>New Contact:</body>
  <campaignUrl>http://192.168.0.1/ccp-webapp/ccp/campaign/Pushed_Contacts</campaignUrl>
  <changeStamp>0</changeStamp>
  <name>Push</name>
  <subject>Notification: New Push Tag applied to Pushed Contacts Campaign</subject>
  <tags>
```

```
      <tag>push</tag>
   </tags>
   <targets>
      <target>user@example.com</target>
   </targets>
  <type>email</type>
</NotificationRule>
```

Example using cURL:

```
curl -i -H "Content-type: application/xml" -X PUT -d "<NotificationRule><body>New Contact:</body><campaignUrl>http://192.168.0.1/ccp-webapp/ccp/campaign/Pushed_Contacts</campaignUrl><name>Push 2</name><subject>Notification: New Push Tag applied to Pushed Contacts Campaign</subject><tags><tag>push2</tag></tags><targets><target>user@cisco.example</target></targets><type>email</type><changeStamp>1</changeStamp></NotificationRule>" http://admin:password@192.168.0.1/ccp-webapp/ccp/notificationrule/100010
```

HTTP Response Headers:

A *200 OK* HTTP header is returned on success.

II. Reporting Development

[contents]

This part of the Cisco SocialMiner Developer's Guide describes how the reporting database works in SocialMiner.

24. Connecting to the Reporting Database

[contents]

You can connect to the reporting database using JDBC. The reporting database runs on Informix,

1.1. Configuring the SQL Connection to the SocialMiner Reporting Database

[contents]

Connection to the SocialMiner Informix reporting database can be made through JDBC using the following format:

```
jdbc:informix-sqli://<hostname>:<port>/<databaseName>:INFORMIXSERVER=<informixserver>;
```

- # The reporting database <port> is 1526.
- # The <databaseName> is "mmca_data".
- # The <informixserver> name is based on the hostname of the server with _mmca append to the end of the hostname. Additional, any dashes ("-") in the hostname are replace by underscores ("_").

For example, if your server's hostname is my-server.mycompany.com, then the INFORMIXSERVER name is my_server_mmca. The complete JDBC URL would be:

```
jdbc:informix-sqli://my-server.mycompany.com:1526/mmca_data:INFORMIXSERVER=my_server_mmca;
```

Note: When authenticating, the username is always reportinguser and the password is the password you created in the Administration Gadget.

25. The Reporting Database Schema

[contents]

The reporting database schema for consists of the following tables:

- **mmca_report_campaign**
- **mmca_campaign_activity**
- **mmca_agent_campaign_activity**

The mmca_report_campaign table contains information used in reports. It is synchronized with campaigns in the Configuration Database when campaign synchronization jobs are run.

Table mmca_report_campaign

Field Name:	Description:	Data Type:	Keys and Null Option:
campaignid	Auto incrementing surrogate ID	serial(8)	Primary Key, Not Null
configcampaignid	The internal database ID.	int(8)	Not Null

campaignname	The Campaign Name as defined in the Campaign Gadget.	nvarchar	Not Null
lastupdated	Last time this row was updated.	datetime	Not Null

The mmca_campaign_activity table is an aggregate table used for reporting campaign statistics.

Table mmca_campaign_activity

Field Name:	Description:	Data Type:	Keys and Null Option:
recordid	Auto-incrementing ID	serial(8)	Primary Key, Not Null
interval	Reporting interval to which this record applies (currently only 15 minute interval is supported)	datetime	Not Null
campaignid	ID of the campaign from the mmca_reportcampaign table. Foreign key relation to....	int(8)	Not Null
screceived	The number of social contacts that were received for this campaign for this interval.	int(8)	Not Null
screserved	The number of social contacts that were reserved for this campaign for this interval.	int	Not Null
schandled	The number of social contacts that were handled for this campaign for this interval.	int	Not Null
scdiscarded	The number of social contacts that were discarded for this campaign for this interval.	int	Not Null
reservedtime	Cumulative reserved time for all social contacts reserved in this campaign for this interval. Reserved time is the time between when the contact was received and when the contact was marked as reserved.	bigint	Not Null
handledtime	Cumulative handled time for all social contacts handled in this campaign for this interval. Handled time is the time between when the contact was received and when the contact was marked as handled.	bigint	Not Null
discardedtime	Cumulative discard time for all social contacts discarded in this campaign for this interval. Discard time is the time between when the contact was received and when the contact was marked as discarded.	bigint	Not Null

The mmca_agent_campaign_activity table is an aggregate table used for reporting campaign statistics .

Table mmca_agent_campaign_activity

Field Name:	Description:	Data Type:	Keys and Null Option:
recordid	Auto-incrementing ID	serial(8)	Primary Key, Not Null

interval	Reporting interval to which this record applies (currently only 15 minute interval is supported)	datetime	Not Null
campaignid	ID of the campaign from the mmca_reportcampaign table. Foreign key relation to....	int(8)	Not Null
userid	String representing the login name of the user to who modified this record for this interval.	varchar	Not Null
schandled	Number of social contacts handled in this interval by the userid for this campaign, during this interval.	int	Not Null
scdiscarded	Number of social contacts discarded in this interval by the userid for this campaign.	int	Not Null
screserveddiscarded	Number of discarded social contacts in this interval that were previously reserved (at any time) by this user.	int	Not Null
screservedhandled	Number of handled social contacts in this interval that were previously reserved (at any time) by this user.	int	Not Null
handledtime	Cumulative Handled Time for all social contacts handled by this user in this interval for this campaign. Handled Time is defined as the time between when a contact was marked as reserved and the time the contact was marked handled.	int(8)	Not Null
discardedtime	Cumulative Discarded Time for all social contacts discarded by this user in this interval for this campaign. Discard Time is defined as the time between when a contact was marked as reserved and the time the contact was marked discarded	int(8)	Not Null