Cisco SocialMiner Developer Guide, Release 10.0(1)

First Published: December 12, 2013

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

CONTENTS

Overview

This document introduces Application Program Interface (API) use and conventions for SocialMiner and provides details about each API. It also describes other areas of concern to developers such as XMPP BOSH Eventing, the reporting database, and security configuration options.

Changes in SocialMiner release 10.0(1)

The following changes are introduced with this release of SocialMiner:

- When agents respond to Facebook and Twitter social contacts, they now have the capability to invite end customers to join them in a chat session by clicking on an 'Insert Chat Invitation' button. After a Chat Invitation Feed has been defined, the Insert Chat Invitation button appears in the Twitter and Facebook reply templates. Agents click this button to invite the customer into a chat session.
    - A Public URL Prefix for Chat Invitation API was added for the invitation to chat.
    - A URL shortener makes the public URL prefix more manageable in user interfaces.
    - The Chat feed API was restructured to distinguish between the creation of the chat feed and the three methods available for initiating a chat request.
- The CCE system-level Agent Request feature enables customers to provide a 'request a call from an agent' function to their end users. End-user requests are collected by SocialMiner and passed to CCE agent queues.
    - A Callback API was added to support agent requests.
    - A Callback feed type was added to the Feed API.
    - A "Connect to CCE notification" type was added to the Notification Rule API to support the submission of new tasks to the media routing peripheral gateway (MR PG).
    - An API was added for media routing (MR) configuration.
    - The Serviceability API was updated to show the status of the connection to the MR PG and the status of each notification rule.
- Other changes in this release:

- ◦ The Chat feed chatproxy url changed from /chatproxy/api/chat to /ccp/chat.
- ◦ A GET (user) API was added to the Facebook Reply template section that retrieves Facebook user profile information similar to the way the Twitter API does.
- ◦ Events for chat contacts have been added to XMPP Bosh eventing.

CHAPTER 1

API conventions

SocialMiner uses REST-based API functions accessed over http. Five API functions are supported; each is mapped to an http operation or command. Not all functions are used for all components.

The URL format is:

```
http://<ServerIP>:<Port>/ccp-webapp/ccp/<Component>
```

where <ServerIP> is the IP address or hostname of the SocialMiner server and the default port is 8080.

The functions are:

- create (http POST)—creates an object in the database and returns a response that contains the URL reference to the newly created object. This example response shows the URL reference returned for a newly-created feed:

  http/1.1 201 Created Location: http://192.168.0.1/ccp-webapp/ccp/feed/100162.

  The id for the feed is *100162*.

  This URL reference can be used to retrieve the object with an http GET.

  Note In some APIs (for example, the callback API), you can also create objects with an http GET (create).
  Although GET (create) does not take a payload, the API developer must supply the required parameters in the URL.

- delete (http DELETE)—deletes an object.
- get (http GET)—returns data for an object. For objects for which there are multiple records, GET takes an identifier variable <id> of some kind. For different APIs, the <id> can appear as a <publicid>, an <objectid>, or another form of id variable.
- update (http PUT)—modifies an object. For some objects, PUT must include a changeStamp, but all other parameters are optional. Some parameters cannot be modified with a PUT as the change would impact system integrity. For example, you cannot change Feed Type or Filter Type. A PUT with a modification to these read-only parameters generates an error.
- list (http GET)—for objects for which there can be multiple records, returns a list. For different components, GET (list) takes different optional URL parameters that modify the content of the returned list. The optional parameters are defined in the sections for each component.

The POST and PUT operations take a payload for which the input format is XML. GET and DELETE calls do not take a payload.

Other than http headers, all output is provided as XML.

XML is case sensitive, therefore all xml element names are case sensitive. For example, <Name> and <name> are two different XML elements.

Boolean values (true and false) are not case sensitive.

If a payload contains duplicate fields, only the first one is interpreted by the server.

Id variables

Different forms of identifier variables are used with the different API functions and components. This section provides a summary of some of the key <id> variables.

objectId

An *objectId* is the generic term for any identifier generated when using http POST to create objects. However, not all identifiers use this generic form. Campaigns use a *publicId* that has special characteristics described below.

Throughout this document, you will see references to different forms of objectIds (such as callbackFeedId or contactId) that are used to distinguish the different objects being addressed. Unless noted below, consider all <id>s found in this document as objectIds.

The DELETE, GET, and PUT operations are performed using the relevant id in the REST URL. For example:

- Use this URL to view results for a specified filter: *http://<ServerIP>:<Port>/ccp-webapp/ccp/filter/<id>/results*.
- Use this URL to delete a feed: *http://<ServerIP>:<Port>/ccp-webapp/ccp/feed/<id>*.
- Use this URL to retrieve data for a single callback contact: *https://<ServerIP>:<Port>/ccp/callback/contact/<id>*.

Use the list (GET) function to determine object identifiers. In this example, the id for the Facebook test feed is *111852*.

```
<Feeds>
 <Feed>
  <authToken>********</authToken>
  <changeStamp>4</changeStamp>
  <name>Facebook Test</name>
    <pollingInterval>300</pollingInterval>
  <refURL>http://[ServerIP]:[Port]/ccp-webapp/ccp/feed/111852</refURL>
  <status>0</status>
  <tags/>
  <type>4</type>
  <url>http://www.facebook.com/TestSocialMiner</url>
 </Feed>
</Feeds>
```

publicId

Using http POST to create a campaign object generates a *publicId* based on the campaign name.

Note The publicId is not editable after it is created and does not change if you change the name of the campaign.

You can generate your own publicId rather than have SocialMiner generate one based on the campaign name. The publicId must conform to RFC 3986 for URL syntax. Spaces, slashes, and backslashes are not allowed in the publicId, and it cannot be blank. When SocialMiner creates a publicId from the provided campaign name element, the string is formatted according to RFC 3986: spaces are replaced with underscores and slashes or backslashes are replaced with hyphens.

If the encoded name results in a collision with another object of the same type, integers starting at 1 are appended to the encoded name until a non-colliding ID is found. If the user changes the name of the campaign later, the publicId will not change.

DELETE, GET, and PUT operations for a campaign are performed using the <publicId> in the REST URL. For example:

- Use this URL to get the count of results for a specified campaign: *http://<ServerIP>:<Port>/ccp-webapp/ccp/campaign/<publicId>/count*).
- Use this URL to update the configuration of a given campaign: *http://<ServerIP>:<Port>/ccp-webapp/ccp/campaign/<publicId>*.

Use the list (GET) function with *summary=false* to see the publicId for a campaign.

progressId

Using http GET for Facebook or Twitter generates a *progressId* for the object. The progressId is used to retrieve the status of a Facebook or Twitter API call.

changeStamp

A changeStamp is a required parameter for the body of a PUT (update) operation for these objects:

- feed
- campaign
- filter
- reply template
- notification
- social contact

A changeStamp is returned as part of a read or creation operation. The changeStamp must be included in any modify or delete request (a method known as optimistic locking) to prevent clients from unintentionally overwriting each others data. If you do not provide a changeStamp or the changeStamp is out of date, the update fails.

If the update is successful, the database increments the changeStamp by 1.

Passwords

For security, the APIs do not return passwords in cleartext. Password elements are masked (******).

HTTP responses

All errors are returned as http 1.1 status codes. The common codes used by the APIs 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, for polling the OAuth status.
- **400 Bad Request**: The request is invalid. Information returned in the ApiErrors message (the example below) shows 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.
- **405 Method Not Allowed**: The method specified in the request line is not allowed for the resource identified by the Request-URI.
- **503 Service Unavailable Error**: The requested operation is unavailable at this time. This error is returned by a chat request when the chat limit has been reached.

Field-specific errors 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>
```

API authentication

SocialMiner APIs that require authentication are grouped under ccp-webapp. The SocialMiner public APIs that do not require authentication are grouped under ccp.

The username and password credentials that were configured for the administrator during installation are used for http basic authentication for those APIs that require it.

When you submit an API call through a web browser, for example *http://<ServerIP>:<Port>/ccp-webapp/ccp/campaign/*, the browser prompts for the username and password.

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

```
curl -I -X GET
http://username:password@<ServerIP>:<Port>/ccp-webapp/ccp/campaign/
```

Failing to provide a username and password or providing incorrect credentials returns **401 Unauthorized**.

If you forget the administrator credentials, refer to the Cisco Systems Command Line Interface (CLI) document for commands you can run to reset them.

Field constraints and limitations

All user-visible configuration objects (such as feeds, campaigns, and filters) have name and description fields. These fields share common constraints on size and the number of characters allowed to ensure a consistent user experience. For common fields, the characteristics are:

Field	Min length	Max length	Allowed characters
Name	1	85	All UTF-8 characters except non-printing ASCII (0-31 and 127).
Description	0	85	All UTF-8 characters except non-printing ASCII (0-31 and 127).

Symbols and special characters are allowed in these fields, but they must be handled carefully (and escaped as required).

See Provisioning in the *SocialMiner User Guide* for information on limitations.

CHAPTER 2

Authentication

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

This API is represented on the SocialMiner user interface by the System Administration panel on the Administration tab.

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

Authentication API commands

This section describes the supported commands for the authentication API and the parameters for those commands.

Related Topics

GET, on page 7
PUT, on page 8
Authentication API parameters, on page 8

GET

Retrieves the authentication information from SocialMiner.

URL:	http://<ServerIP>:<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.xx.yyy.zzz</primaryHost>`
	` <primaryPort>3268</primaryPort>`
	` <primaryUseSSL>false</primaryUseSSL>`
	` <refURL>`
	`   http://<ServerIP>:<Port>/ccp-webapp/ccp/`
	`   authentication`
	` </refURL>`
	` <roleName></roleName>`
	`</Authentication>`

Parameters:	See Authentication API parameters, on page 8.

PUT

Updates the authentication information from SocialMiner.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/authentication/
HTTP method:	PUT

Example XML payload:	`<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>`

Parameters:	See Authentication API parameters, on page 8.

Authentication API parameters

Parameters are optional unless otherwise noted.

Parameter	Description	Notes
enabled	True/False. Indicates if the authentication settings are used when trying to authenticate a user.	If this is disabled, only the application administrator will have access to the system.

Parameter	Description	Notes
managerDistinguishedName	The distinguished name of a user that has manager access to the AD server. For example CN=Administrator, CN=users, DC=MYSERVER, DC=COM.	Required if the enabled parameter is true.
managerPassword	The password of the user specified in the managerDistinguishedName field.	Required if enabled parameter is true.
primaryPort	The host port.	Required if enabled parameter is true.
primaryHost	The host address of the AD server.	Required if enabled parameter is true.
primaryUseSSL	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** Enable SSL for Active Directory authentication, on page 9.
roleName	The name of an AD role or group.	All users in this AD role or group are provided access to SocialMiner. Users in AD who are not members of this role or group are not provided access to SocialMiner. Blank or * indicates that all users in AD are allowed to use the application.

Enable SSL for Active Directory authentication

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

On the Active Directory Server:

Procedure

Step 1 Verify that the Active Directory has the Certificate Services service installed.

Step 2 Select **All Programs > Administrative Tools > Certificate Authority**.

Step 3 Expand the domain node and select **Issued Certificates**.

Step 4 Double click the certificate to open it.

Step 5 Open the Details tab and click **Copy to file**.

Step 6 An Export wizard appears. In the wizard, select DER encoded binary.

Step 7 Use the wizard to select a location to save the file.

Step 8 Click **Finish**.

Enable SSL for Active Directory on the SocialMiner server

On the SocialMiner server:

Procedure

Step 1 Open the *Cisco Unified Operating System Administration* by entering the URL `http://<servername>/cmplatform` or by using the Platform Administration link in the System Administration gadget.

Step 2 Select **Security > Certificate Management**.

Step 3 Click **Upload Certificate**.

Step 4 For the Certificate Name, select **tomcat-trust**.

Step 5 In the Upload File field, locate the file to upload by clicking **Browse...** and select the certificate file you saved from the Active Directory server.

Step 6 Click **Upload File**.

Step 7 Restart the Cisco Tomcat service by using the CLI to run the command `utils service restart Cisco Tomcat`.

CHAPTER 3

Bayesian filter training

The Bayesian filter training API allows you to define whether or not social contacts containing specific types of content should be included or excluded from campaigns to which the filter is applied.

Bayesian filter training API commands

This section describes the supported API commands for the Bayesian filter training API and the parameters for those commands.

Related Topics

PUT (train), on page 11
DELETE, on page 12

PUT (train)

Adds the content of a document (text in the REST call) or social contact to the specified filter (objectId) and indicates whether to include or exclude this type of content when the filter is run.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/filter/<objectId>/train
HTTP method:	PUT
Parameters:	• **document**: Required if socialcontact is not specified. String. The text on which to train the filter. • **socialContact**: Required if document is not specified. String. The URL of the social contact containing the content on which to train the filter. • **match**: Required. Boolean. ◦ Set to “True” to include social contacts with similar content in campaigns where the filter is applied.

	◦ Set to “False” to exclude social contacts with similar content in campaigns where the filter is applied

Example XML request payload using socialContact:	`<TrainingRequest>`
	` <socialContact>`
	`   http://[ServerIP]:[Port]/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>`

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:`

Related Topics

Id variables, on page 2

DELETE

Deletes all training data for a filter.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/filter/ <objectId>/trainingdata
HTTP method:	DELETE

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

Related Topics

Id variables, on page 2

CHAPTER 4

Callback

The Callback API allows callback applications to send a notification to CCE for an agent to make a voice call to a customer, at the customer's request The API works in conjunction with a callback feed, campaigns, and a Connection to CCE Notification.

You can use the API to poll the status of the request (including estimated wait time) and to cancel previous callback requests (see DELETE below).

Note Only the callback API can be used to **submit or cancel** callback requests. It cannot be done with the Social contact API.

Before creating a callback request, you must have a Callback feed (type 10) assigned to a campaign (see Feed, on page 63) and you must have a Connection to CCE notification set up. The notification is triggered by a specific tag that is automatically created in the callback social contact.

The response of the create request contains a URL in the location field that can be used by applications to retrieve the status of the callback request, including the estimated wait time.

The URL is available until the request has been cancelled or until there has been no polling detected for at least five minutes.

Note The estimated wait time is calculated only once CCE, so it is not necessary to update that value on each poll.

Callback API commands

This section describes the supported commands for the Callback API and the parameters for those commands.

POST

Sends the XML body of a contact to SocialMiner to make a callback request.

Alternatively, GET can be used to create a callback request. GET uses UTF-8 encoded URL parameters to provide the parameters required for the contact. See GET (create callback request), on page 16.

Note The variables cv_1 to cv_10 are included as call variables when SocialMiner initiates a callback request with CCE. All variables starting with "user_" are included as expanded call context (ECC) variables when SocialMiner initiates a callback request with CCE.

URL:	https://<ServerIP>:<Port>/ccp/callback/feed/<callbackFeedId> The <callbackFeedId> specifies the callback feed to target for the callback request.
HTTP method:	POST

Example XML payload:

```
<Contact>
  <name>name</name>
  <title>title</title>
  <description>description</description>
  <mediaAddress>phoneNumber</mediaAddress>
  <tags>
    <tag>tag1</tag>
    <tag>tag2</tag>
  </tags>
  <variables>
    <variable>
      <name>cv_[1-10]</name>
      <value>callVariableValue</value>
    </variable>
    <variable>
      <name>user_(eccVariableName)</name>
      <value>eccVariableValue</value>
    </variable>
    <variable>
      <name>anythingElseExtensionFieldName</name>
      <value>anythingElseExtensionFieldValue</value>
    </variable>
  </variables>
</Contact>
```

Note The contact name, title, and mediaAddress (the phone number to be called) are required to create the request.

A reference URL to the contact is returned in the location field in the header:	https://<ServerIP>:<Port>/ccp/callback/contact/6EEF968810000132000015F60A568DFB
Response codes:	201 Created 400 Bad Request See HTTP responses for more information about the response codes.

Extension fields, call variables, and ECC variables :

There is an associated callback contact in SocialMiner for every contact that is processed by the callback API. All of the variables associated with the contact in the create request are included as extension fields in the SocialMiner callback contact. For example, a contact with the following XML:

```
<Contact>
  <name>Customer</name>
  <title>Help</title>
  <mediaAddress>5551212</mediaAddress>
  <variables>
    <variable>
      <name>cv_7</name>
      <value>test7</value>
    </variable>
    <variable>
      <name>user_user.callback.test</name>
      <value>ct7</value>
    </variable>
    <variable>
      <name>location</name>
      <value>Boston, MA</value>
    </variable>
  </variables>
</Contact>
```

results in the following SocialMiner callback contact being created:

```
<SocialContact>
  <author>Customer</author>
  <title>Help</title>
  <description/>
  <extensionFields>
    <extensionField>
      <name>mediaAddress</name>
      <value>5551212</value>
    </extensionField>
    <extensionField>
      <name>location</name>
      <value>Boston, MA</value>
    </extensionField>
    <extensionField>
      <name>cv_7</name>
      <value>test7</value>
    </extensionField>
    <extensionField>
      <name>user_user.callback.test</name>
      <value>ct7</value>
    </extensionField>
    <extensionField>
      <name>ewt</name>
      <value>8</value>
    </extensionField>
  </extensionFields>
  <status>handled</status>
  <statusReason>externally_handled</statusReason>
</SocialContact>
```

Note Not all SocialMiner callback contact attributes have been included in this example.

All variables pulled from contact XML are stored as extension fields in the SocialMiner callback contact. The variables cv_1 to cv_10 are treated as call variables. All variables starting with "user_" are treated as ECC variables. The "cv_" and "user_" prefixes are not case sensitive.

This example shows how to set call variable 7 to "test7", ECC variable user_user.callback.test to "ct7", and the extension field location to Boston.

```
<Contact>
```

```
  <name>Customer</name>
  <title>Help</title>
  <mediaAddress>5551212</mediaAddress>
  <variables>
    <variable>
      <name>cv_7</name>
      <value>test7</value>
    </variable>
    <variable>
      <name>user_user.callback.test</name>
      <value>ct7</value>
    </variable>
    <variable>
      <name>location</name>
      <value>Boston</value>
    </variable>
  </variables>
</Contact>
```

GET (create callback request)

As an alternative to POST, GET (create callback request) uses UTF-8 encoded URL parameters to provide the parameters required for the callback contact.

Note The variables cv_1 to cv_10 are included as call variables when SocialMiner initiates a callback request with CCE. All variables starting with "user_" are included as ECC variables when SocialMiner initiates a callback request with CCE.

URL:	https://<ServerIP>:<Port>/ccp/callback/feed/<callbackFeedId> The <callbackFeedId> specifies the callback feed to target for the callback request.
HTTP method:	GET
Parameters:	See Callback API parameters.
The contact is returned in the location field in the header:	https://<ServerIP>:<Port>/ccp/callback/contact/ 6EEF968810000132000015F60A568DFB
Response codes:	201 Created 400 Bad Request See HTTP responses for more information about the response codes.

When using GET to create a callback, variables must be passed as query parameters. Variables are denoted by the "variable_" prefix. For example, use the following GET command to create a callback contact with

- call variable 7 set to "test7"
- ECC variable user_user.callback.test set to "ct7"

```
http://sample_server/ccp/callback/feed/12345?
name=Customer
&title=Help
```

```
&mediaAddress=5551212
&variable_cv_7=test7
&variable_user_user.callback.test=ct7
```

Note Sending an ECC variable that is not configured in CCE when creating a callback request will not result in the request failing. (CCE will ignore the variable when processing the request.)

Use the 'tags' parameter in the query string to include tags when creating a callback request using GET.

```
http://sample_host/ccp/callback/feed/12345
?name=Customer
&title=Help
&mediaAddress=5551212
&tags=tag1,tag2,tag3
```

GET

Returns a reference URL for a single callback contact.

URL:	https://<ServerIP>:<Port>/ccp/callback/contact/<ContactID>
HTTP method:	GET
Parameters:	See Callback API parameters.
A reference URL to the contact is returned in the location field in the header:	https://<ServerIP>:<Port>/ccp/callback/contact/6EEF96881000013200015F60A568DFB **Note** The URL returned in the GET call is available until the request is cancelled or until there has been no polling for at least five minutes.
Response codes:	200 OK 400 Bad Request For more information about the response codes, see HTTP responses, on page 4.

Note The customer callback application developer is responsible for the messages that their customer callback interface provides to the customer. The <status> and <statusReason> fields can be used to understand the state of the contact.

Contact transition states during callback flow are listed in the table below.

Contact state	State reason code	Notes
Queued	EXTERNALLY_HANDLED	The callback request was successfully submitted to the contact center for routing.

Contact state	State reason code	Notes
Handled	EXTERNALLY_HANDLED	The callback request was successfully routed to a contact center agent.

Contact state	State reason code	Notes
Discarded	**StatusReason for failure**	
	NOTIFICATION_INVALID_NEW_ TASK_MESSAGE	CCE routing error: Invalid message in route request.
	NOTIFICATION_MEDIA_ ROUTING_ DISABLED	CCE routing error: Routing is disabled.
	NOTIFICATION_NO_SCRIPT	CCE routing error: No available script to run. Mis-configuration of CCE. No script is available to run.
	NOTIFICATION_INVALID_ MRD_ID	CCE routing error: Invalid media routing domain. Mis-configuration of CCE. Invalid CCE Media Routing Domain. Ensure that the Media Routing Domain configured in the SocialMiner Notification exists in the CCE configuration.
	NOTIFICATION_INVALID_ SCRIPT_ SELECTOR	CCE routing error: Invalid dialed number or script selector. Mis-configuration of CCE. Invalid CCE Dialed Number/Script Selector. Ensure that the Dialed Number/Script Selector configured in the SocialMiner Notification exists in the CCE configuration.
	NOTIFICATION_ROUTER_ RELEASED_ TASK	CCE routing error: CCE released the call or task. The CCE script indicated that the callback request should be dropped. Possible script mis-configuration.
	NOTIFICATION_UNKNOWN_ ROUTING_PROBLEM	CCE routing error: Unknown problem.
	NOTIFICATION_CCE_ CONNECTION_LOST	The connection to CCE was lost or was not established.
	NOTIFICATION_CCE_ SOCIALMINER_SYSTEM_ FAILURE	SocialMiner failed (or restarted) while the callback was queued.
	NOTIFICATION_INVALID_ VARIABLE	

Contact state	State reason code	Notes
		SocialMiner could not submit the task to CCE because it contained invalid media address, call variable, or ECC variable values. The maximum field lengths for these fields are • media address: 39 bytes • call variable: 40 bytes • ECC variable name: 32 bytes • ECC variable value: 210 bytes.
	NOTIFICATION_RATE_LIMITED	The incoming rate of callback contacts has exceeded 40 contacts per minute.

DELETE

Cancels a callback request. Once the DELETE request is successfully processed (response code 200 is received by the application), the application cannot use GET to poll for the status of the contact.

Note It is possible for the customer to receive a callback after cancelling the callback request if the cancellation request arrives after an agent was already selected.

URL:	https://<ServerIP>:<Port>/ccp/callback/contact/<Id>
HTTP method:	DELETE
Example request XML payload:	None
Response codes:	200 OK 400 Bad Request See HTTP responses for more information about the response codes.

The following table shows the statusReason codes that can be triggered by the cancellation requests for callback contacts in different states.

Social contact state	StatusReason for cancel request	Meaning
QUEUED	NOTIFICATION_CCE_ CALLBACK_CANCEL_REQUESTED	Cancel callback request was initiated.

Social contact state	StatusReason for cancel request	Meaning
DISCARDED	NOTIFICATION_CCE_CALLBACK_ CANCEL_SUCCEEDED	CCE cancelled the callback successfully.
HANDLED	EXTERNALLY_ HANDLED	The cancellation request failed because the task was routed to an agent before the callback could be cancelled.

Callback API parameters

Parameters are optional unless otherwise noted.

Parameter	Description	Notes
title	The name of the callback request.	Required for POST and GET (create).
name	The person who generated the callback request.	Required for POST and GET (create). The name is limited to a maximum of 100 characters and a newline (\n) is not allowed in the name.
mediaAddress	The phone number to be called.	Required for POST and GET (create).
description	An optional description to accompany the callback request.	
tags	One or more tags or keywords associated with the contact.	To include tags when using GET to create a callback contact, use the parameter 'tags' in the query string.
variables	A set of custom name and value pairs.	When using GET to create a callback, variables must be passed as query parameters and start with 'variable_'.
status	The status of a callback request.	See "Contact State" in GET, on page 17.
estimatedWaitTime	An estimate of the amount of time (in seconds) until an agent's phone will be available to place the call.	

CHAPTER 5

Campaign

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

Campaigns are collections of feeds (see Feed) and filters (see Filter) that generate lists of results matching the criteria defined in the campaign.

This API is represented on the SocialMiner user interface by the Campaigns panel on the Configuration tab.

Campaign API commands

This section describes the supported commands for the Campaign API and the parameters for those commands.

Related Topics

POST
PUT
DELETE, on page 25
GET
GET (list)
GET (suggested tags), on page 28
Campaign API parameters, on page 28

POST

Creates a campaign.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/campaign
HTTP method:	POST
Parameters:	See Campaign API parameters, on page 28.

Example XML request payload:	`<Campaign>`
	`  <name>MyTestCampaign</name>`
	`  <publicId>MyTestCampaign</publicId>`
	`  <description>This is my test campaign</description>`
	`  <includeExpr>Cisco Expert Advisor</includeExpr>`
	`  <excludeExpr>ICM</excludeExpr>`
	`  <chatInvitationFeed>`
	`    http://[ServerIP]:[Port]/ccp-webapp/ccp/feed/5000`
	`  </chatInvitationFeed>`
	`  <feeds>`
	`    <feed>`
	`     http://[ServerIP]:[Port]/ccp-webapp/ccp/feed/5000`
	`    </feed>`
	`    <feed>`
	`     http://[ServerIP]:[Port]/ccp-webapp/ccp/feed/5001`
	`    </feed>`
	`  </feeds>`
	`  <filters>`
	`    <filter>`
	`     http://[ServerIP]:[Port]/ccp-webapp/ccp/filter/6000`
	`    </filter>`
	`    <filter>`
	`     http://[ServerIP]:[Port]/ccp-webapp/ccp/filter/6001`
	`    </filter>`
	`  </filters>`
	`</Campaign>`

HTTP response headers:	If the campaign is successfully created, the URL of the created resource is returned.
	`http/1.1 201 Created`
	`Location: http://<ServerIP>:<Port>/ccp-webapp/ccp/`
	`campaign/MyTestCampaign`
	`Content-Type: text/plain`
	`Content-Length: 0`
	`Date: Tue, 12 Jan 2010 16:41:14 GMT`
	See also HTTP responses.

PUT

Updates an existing campaign.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/campaign/<publicId> For more information about <publicId>, see Id variables, on page 2.
HTTP method:	PUT
Parameters:	See Campaign API parameters, on page 28.

Example XML request payload:	`<Campaign>`
	`  <name>MyTestCampaign</name>`
	`  <publicid>MyTestCampaign</publicid>`
	`  <description>This is my test campaign</description>`
	`  <includeExpr>Cisco Expert Advisor</includeExpr>`
	`  <excludeExpr>ICM</excludeExpr>`
	`  <chatInvitationFeed>`
	`    http://[ServerIP]:[Port]/ccp-webapp/ccp/feed/5000`
	`  </chatInvitationFeed>`
	`  <feeds>`
	`    <feed>`
	`     http://[ServerIP]:[Port]/ccp-webapp/ccp/feed/5000`
	`    </feed>`
	`    <feed>`
	`     http://[ServerIP]:[Port]/ccp-webapp/ccp/feed/5001`
	`    </feed>`
	`  </feeds>`
	`  <filters>`
	`    <filter>`
	`     http://[ServerIP]:[Port]/ccp-webapp/ccp/filter/6000`
	`    </filter>`
	`    <filter>`
	`     http://[ServerIP]:[Port]/ccp-webapp/ccp/filter/6001`
	`    </filter>`
	`  </filters>`
	`  <changeStamp>8</changeStamp>`
	`</Campaign>`

DELETE

Deletes an existing campaign.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/campaign/<publicId> For more information about <publicId>, see Id variables, on page 2.
HTTP method:	DELETE

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

GET

Returns the data for a single campaign.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/campaign/<publicId>?metrics=<true/false> For more information about <publicId>, see Id variables, on page 2.
HTTP method:	GET
Parameters:	See Campaign API parameters, on page 28.

Example XML response:	**Note** The" refURL "is a copy of the URL requested.
	```
	<Campaign>
	  <refURL>
	    http://[ServerIP]:[Port]/ccp-webapp/ccp/
	    campaign/MyTestCampaign
	  </refURL>
	  <resultsURL>
	    http://[ServerIP]:[Port]/ccp-webapp/ccp/
	    campaign/MyTestCampaign/results
	  </resultsURL>
	  <suggestedTagsURL>
	    http://[ServerIP]:[Port]/ccp-webapp/ccp/
	    campaign/MyTestCampaign/suggestedtags
	  </suggestedTagsURL>
	  <publicid>MyTestCampaign</publicid>
	  <description>This is my test campaign</description>
	  <includeExpr>Cisco Expert Advisor</includeExpr>
	  <excludeExpr>ICM</excludeExpr>
	  <chatInvitationFeed>
	    http://[ServerIP]:[Port]/ccp-webapp/ccp/feed/5000
	  </chatInvitationFeed>
	  <changeStamp>12345</changeStamp>
	  <feeds>
	    <feed>
	     http://[ServerIP]:[Port]/ccp-webapp/ccp/feed/5000
	    </feed>
	    <feed>
	     http://[ServerIP]:[Port]/ccp-webapp/ccp/feed/5001
	    </feed>
	  </feeds>
	  <filters>
	    <filter>
	     http://[ServerIP]:[Port]/ccp-webapp/ccp/filter/6000
	    </filter>
	    <filter>
	     http://[ServerIP]:[Port]/ccp-webapp/ccp/filter/6001
	    </filter>
	  </filters>
	  <metrics>
	    <socialContactCount>12</socialContactCount>
	  </metrics>
	</Campaign>
	```
	The metrics element contains <socialContactCount>, which is the number of social contacts associated with this 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
	```

GET (list)

Lists all configured campaigns.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/campaign?summary=<true/false> Where *summary* is an optional query parameter that is false by default. OR http://<ServerIP>:<Port>/ccp-webapp/ccp/campaign?metrics=<true/false>

	Where *metrics* is an optional query parameter that is false by default
HTTP method:	GET

Example XML responses:	When summary is true and metrics is false:
	
	`<Campaigns>`
	`  <Campaign>`
	`    <refURL>`
	`       http://[ServerIP]:[Port]/ccp-webapp/ccp/`
	`       campaign/MyTestCampaign`
	`    </refURL>`
	`  </Campaign>`
	`  <Campaign>`
	`    <refURL>`
	`      http://[ServerIP]:[Port]/ccp-webapp/ccp/`
	`      campaign/MyTestCampaign2`
	`    </refURL>`
	`  </Campaign>`
	`</Campaigns>`

	When summary is false and metrics is false:
	
	`<Campaigns>`
	`  <Campaign>`
	`    <refURL>`
	`      http://[ServerIP]:[Port]/ccp-webapp/ccp/`
	`      campaign/MyTestCampaign`
	`    </refURL>`
	`    <name>MyTestCampaign</name>`
	`    <publicid>MyTestCampaign</publicid>`
	`    <description>This is my test campaign</description>`
	`    ...`
	`  </Campaign>`
	`  <Campaign>`
	`    <refURL>`
	`      http://[ServerIP]:[Port]/ccp-webapp/ccp/`
	`      campaign/MyTestCampaign2`
	`    </refURL>`
	`    <name>MyTestCampaign2</name>`
	`    <publicid>MyTestCampaign2</publicid>`
	`    <description>This is my test campaign</description>`
	`    ...`
	`  </Campaign>`
	`</Campaigns>`

	When summary is false and metrics is true:
	
	`<Campaigns>`
	`  <Campaign>`
	`    <refURL>`
	`      http://[ServerIP]:[Port]/ccp-webapp/ccp/`
	`      campaign/MyTestCampaign`
	`    </refURL>`
	`    <name>MyTestCampaign</name>`
	`    <publicid>MyTestCampaign</publicid>`
	`    <description>This is my test campaign</description>`
	`    <metrics>`
	`      <socialContactCount>12</socialContactCount>`
	`    </metrics>`
	`    ...`
	`  </Campaign>`
	`  <Campaign>`
	`    <refURL>`
	`      http://[ServerIP]:[Port]/ccp-webapp/ccp/`

```
    campaign/MyTestCampaign2
  </refURL>
  <name>MyTestCampaign2</name>
  <publicid>MyTestCampaign2</publicid>
  <description>This is my test campaign</description>
  <metrics>
    <socialContactCount>12</socialContactCount>
  </metrics>
  ...
 </Campaign>
</Campaigns>
```

Summary set to true and metrics set to true is an invalid combination.

GET (suggested tags)

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

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/campaign/<publicId>/suggestedtags For more information about <publicId>, see Id variables, on page 2.
HTTP method:	GET

Example XML response:	The first 10 suggested tags are returned
	`<SuggestedTags>`
	` <tags>`
	`  <tag>tag1</tag>`
	`  <tag>tag2</tag>`
	`  <tag>tag3</tag>`
	` </tags>`
	`</SuggestedTags>`

Campaign API parameters

Parameters are optional unless otherwise noted.

Parameter	Description	Notes
changeStamp	The change stamp of the campaign record.	Integer. Defaults to 0. Required for PUT. For more information, see changeStamp.
chatInvitationFeed	The chat invitation feed for the campaign (must be a chat feed); must be set to a feed's reference URL.	A chat invitation feed is required to invite Twitter and Facebook users to chat from the SocialMiner reply templates.
description	The description of the campaign.	
excludeExpr	The searching expression to exclude.	

Parameter	Description	Notes
feeds	A list of feeds linked to the campaign.	
filters	A list of filters linked to the campaign.	
includeExpr	The searching expression to include.	
metrics (socialContactCount)	URL Parameter for GET and GET list.	True/false. Defaults to false. If “true”, a count of the social contacts in each campaign is returned. If “false”, no social contact count is returned.
name	The name of the campaign.	Required for POST.
publicId	URL-encoded version of name. Must be unique within object type.	
refURL	A copy of the URL requested.	
suggestedTags	Up to ten suggested tags for the campaign.	
summary	URL Parameter for List.	True/false. Defaults to false. If “true”, only the URLs of the objects are returned. If “false”, full object information is returned along with the URLs of the objects.

CHAPTER 6

Campaign results

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

This API is represented on the SocialMiner user interface in the Home tab.

Campaign results API commands

This section describes the supported command (GET) for the campaign results API and the parameters for that command.

Related Topics

GET, on page 31
Campaign results URL parameters, on page 33
Campaign results response parameters, on page 34

GET

Gets results for the specified campaign based on an optional index.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/campaign/<publicId>/results For more information about <publicId>, see publicId.
HTTP method:	GET
URL parameters:	See Campaign results URL parameters, on page 33.

Example response:

```
<?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://[ServerIP]:[Port]/
    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;startIndex=50" />

   <subtitle>This feed has been created by the Cisco
    Customer Collaboration Platform</subtitle>

<id>http://[ServerIP]:[Port]/ccp-webapp/ccp/campaign/
    Business_News/results</id>
   <updated>2010-06-10T17:20:46Z</updated>
   <dc:date>2010-06-10T17:20:46Z</dc:date>
   <stentry>
     <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>
   </stentry>
   <stentry> ... </stentry>
</feed>
```

For more information about the response parameters, see Campaign results response parameters, on page 34.

Campaign results URL parameters

Parameters are optional unless otherwise noted.

Parameter	Description	Notes
filterStatus	Display contacts whose status matches a status within this field.	String. Defaults to all if the parameter is not specified. You must specify a value for the parameter if the parameter is included or no contacts are returned. Can be one or more of (case-insensitive): • RESERVED • HANDLED • DRAFT • UNREAD • DISCARDED • QUEUED • Multiple status example: http://192.168.0.1/ccp-webapp/ccp / campaign/ Business_News/results **?filterStatus=RESERVED & filterStatus=HANDLED**
filterTag	Display contacts whose filters matches one or more of the tags in this field.	String. Defaults to all tags if not specified. Example: http://192.168.0.1/ccp-webapp/ccp /campaign/Business_News/results **?filterTag=tag1** **&filterTag=tag2**
includePostWithPostId	Specifies whether or not to display contacts with ID equal to postId.	This option is ignored if postId is not specified. If includePostWithPostId is not specified or is specified and is set to false, the contact with ID equal to postId is not included in the campaign results. If includePostWithPostId is specified and is set to true, the contact with ID equal to postId is included in the campaign results.

Parameter	Description	Notes
postId	The identifier of the userid that made the post.	String. If provided, results are displayed starting after the provided postId. It cannot be used if timestamp or startIndex is specified.
timestamp	A given time and date.	Integer. Displays results older than this timestamp. Defaults to the time of request if not provided. If startIndex is not specified, then timestamp assumes startIndex = 0.
resultsPerPage	The maximum number of results to be returned.	Integer. Default is 50 and maximum is 200.
startIndex	The number of results to skip based on the timestamp.	Integer. Used for pagination. Assuming resultsPerPage is set at the default of 50, you could create a "page 2" link by using the timestamp provided in the href of the `feed/link rel="self"` and a startIndex of 50. Page 3 would use the same timestamp and a startIndex of 100, and so on.

Note

- If timestamp is provided and startIndex is not provided, then the results are displayed up to the "resultsPerPage" with a creation date older than "timestamp", starting at index 0.
- If timestamp is not provided and startIndex is provided, then the results are displayed up to the "resultsPerPage" with a creation date older than "now" starting at startIndex.
- 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.

Campaign results response parameters

Results are returned as an ATOM 1.0 feed that can contain the following elements:

Parameter	Description	Notes
feed/title	The name of the campaign.	
feed/link rel = self & feed/id	The URL of the results that were requested.	

Parameter	Description	Notes
feed/link rel = countsince	The URL for the API call for the number of new contacts since this result was retrieved.	See Campaign results count, on page 37.
feed/link rel = next	The URL to the next 50 results.	Present only when there are more than 50 results left in the campaign. Represented in the SocialMiner user interface by the More button.
entry/title	The title of the contact.	
entry/link rel = alternate & entry/id	The URL to the contact.	
entry/link rel = socialcontact	The URL for the API call of this contact.	
entry/summary	The content of the contact.	
entry/published	The date and time that the contact was published. If the contact did not contain a published date, this is the date when the contact was read by SocialMiner.	Date and time form is *YYYY-MM-DD*T*HH:MM:SS*Z.
entry/ccp:statustimestamp	The timestamp of the last change to the status of the contact.	
entry/ccp:status	The current status of the contact.	
entry/ccp:scstatusreason	The reason the contact changed to its current status.	
entry/ccp:sctags/ccp:sctag	One or more tags associated with this contact.	
entry/ccp:scstatususerid	The last user to change the status of this contact. If blank and the status is unread, then this contact has never had a status change.	
entry/ccp:sourcetype	The type of feed that generated or fetched the contact.	

Parameter	Description	Notes
entry/ccp:scissoftlocked	Whether or not the contact state can be modified using the SocialMiner user interface.	

CHAPTER 7

Campaign results count

The 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.

Campaign results count API commands

This section describes the supported command (GET) for the campaign results API.

GET

Gets the number of results in a campaign.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/campaign/ <publicId>/count For more information about <publicId>, see publicId.
HTTP method:	GET
URL parameter:	**postId**: Optional. The postId of the last post seen. This id is the unique id of the campaign result. 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 result, for example: `<link rel="countsince" href="http://[ServerIP]:[Port]/ ccpwebapp/ccp/campaign/MyTestCampaign/count? postId=92517B6610000128295CEBB40A568DDE" />`
Example XML response:	`<count>44</count>`

CHAPTER 8

Chat feed

Chat feeds are used by the Chat REST API. The Chat REST API supports operations on the URL http://<ServerIP>:<Port>/ccp/chat..

For more information on feeds, see Feed, on page 63.

Create a chat feed

Before creating a chat request, you must have a chat feed assigned to a campaign. Once a chat feed is set up, three methods are used to create chat requests. Use the following steps to create the feed and assign it to a campaign:

Procedure

Step 1 Use the POST to create a type 8 chat feed.

Step 2 Confirm that the POST returned a 201 (created) response code, then look in the location field of the http response header for the reference URL (refURL) of the feed just created.

Step 3 Add the chat feed to a campaign. You can create a new campaign and then use the PUT API to add the feed to it or to any existing campaign.

POST

After creating the chat feed, POST the body of the contact to the chat proxy (with no re-direct to the customer chat UI).

URL:	http://<ServerIP>:<Port>/ccp/chat
HTTP method:	POST

Example XML request payload:	`<SocialContact>`
	`  <feedRefURL>http://[ServerIP]:[Port]/`
	`   ccp-webapp/ccp/feed/134268</feedRefURL>`
	`  <author>Test_author</author>`
	`  <title>Social contact title</title>`
	`</SocialContact>`

The contact is returned in the location field in the header:	http://<ServerIP><:<Port>/ccp-webapp/ccp/socialcontact/ 6EEF968810000132000015F60A568DFB

The chat room is in the Extension Fields of the content response:	`<SocialContact>`
	`  <author>Test_author</author>`
	`  <createdDate>1316121187977</createdDate>`
	`  <description/>`
	`    <extensionFields>`
	`      <extensionField>`
	`        <name>chatRoom</name>`
	`       <value>socialminer_chat.3@conference.127.0.0.1</value>`
	
	`      </extensionField>`
	`    </extensionFields>`
	`    <id>6EEF968810000132000015F60A568DFB</id>`
	`    <link>`
	`      http://[ServerIP]:[Port]/`
	`      ccp-webapp/ccp/socialcontact/`
	`      6EEF968810000132000015F60A568DFB`
	`    </link>`
	`    <publishedDate>1316121187976</publishedDate>`
	`    <refURL>`
	`      http://[ServerIP]:[Port]/`
	`      ccp-webapp/ccp/socialcontact/`
	`      6EEF968810000132000015F60A568DFB`
	`    </refURL>`
	`    <replyToId/>`
	`      <status>unread</status>`
	`      <statusTimestamp>1316121187976</statusTimestamp>`
	`      <statusUserId/>`
	`        <tags/>`
	`          <title>Social contact title</title>`
	`</SocialContact>`

Parameters:	See Chat feed API parameters (create chat request), on page 42.
HTTP response headers:	See HTTP responses.

Chat request form submission

Chat requests are created when a customer fills in and submits a chat request form. This method creates the contact.

If the contact is successfully created, the feed is redirected to the customer chat UI. The URL to submit the GET is http://<ServerIP>:<Port>/ccp/chat/<feedid>/redirect.

The code below is an example of how to submit a chat request through a form post. Users can customize this basic HTML to suit their needs.

```
<style type='text/css'>span { display: inline-block; width: 100px; }</style>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<form action='https://10.86.141.242/ccp/chat/form/100525' method='post'>
    <span>Title:</span><input id='title' type='text' name='title' /><br/>
    <span>Author:</span><input id='author' type='text' name='author' /><br/>
   <span>Description:</span><input id='description' type='text' name='description' /><br/>

    <span>Tags:</span><input id='tags' type='text' name='tags' /><br/>
    <span>Remarks:</span><input id='remarks' type='text' name='extensionField_remarks'
      value='sample value' /><br>
    <input id='submit' type='submit' value='Submit'/>
    <input type="hidden" name="extensionField_chatLogo" value="./img/ciscoLogoColor.png">
    <input type="hidden" name="extensionField_chatWaiting"
     value="Welcome, please wait while we connect you with a customer care representative.">

    <input type="hidden" name="extensionField_chatAgentJoinTimeOut"
      value="All customer care representatives are busy assisting other clients.
      Please continue to wait or try again later.">
    <input type="hidden" name="extensionField_chatError"
      value="Sorry, the chat service is currently not available. Please try again later.">
</form>
```

See Chat feed API parameters (create chat request), on page 42.

GET method

Alternately, a simple GET can be used to create a chat request where all the required parameters for the social contact and the chat session created are provided as UTF-8 encoded URL parameters.

If the contact is successfully created, the feed is redirected to the customer chat UI. The URL to submit the GET is http://<ServerIP>:<Port>/ccp/chat/<feedid>/redirect.

GET (create chat request)

Creates a social contact with the chat session for the particular chat feed id (<id>) and redirects the feed to the customer chat UI.

URL:	http://<ServerIP>:<Port>/ccp/chat/<id>/redirect
HTTP method:	GET
Parameters:	See Chat feed API parameters (create chat request), on page 42.
Response:	302 redirect.

Once submitted, the client needs to continue the chat for the current browser session using the GET/PUT and DELETE operations on URL **http://<ServerIP>:<Port>/ccp/chat** (without the feedid).

Chat feed API parameters (create chat request)

Parameters are optional unless otherwise noted.

Parameter	Description	Notes
title	The name of the contact.	Required
author	The author of the contact.	Required
description	A description of the content or context of the contact.	
tags	One or more tags that are associated with this contact.	
extensionField	A custom name/value pair.	The name is formatted (extensionField_<field name>). All parameters in the URL must be UTF-8 encoded.

GET (events)

The remaining APIs in this chapter are used after a chat session has been created.

This API gets events queued on the chat proxy starting from the specified eventid. GET returns an XML payload wrapped in a <ChatEvents> tag containing 0 or more event types returned in the order they were received. When first called, eventid should be set to 0. Subsequent calls should identify the id of the last processed event.

Note Because there is no identifier specifying which chat we are getting events for, API calls only work with a valid session cookie. The session cookie returned from the POST or GET should be provided on subsequent chat proxy API calls.

URL example:	http://<ServerIP>:<Port>/ccp/chat?eventid=0&all=false
HTTP method:	GET
Parameter:	See below.

Example response payload:	`<ChatEvents>`
	` <PresenceEvent>`
	`  <id>2</id>`
	`  <from>Steve</from>`
	`  <status>joined</status>`
	` </Presence>`
	` <MessageEvent>`
	`  <id>3</id>`
	`  <from>Steve</from>`
	`  <body>Hi There</body>`
	` </Message>`
	` <MessageEvent>`
	`  <id>4 </id>`
	`  <from>Steve</from>`
	`  <body>How can I help you?</body>`
	` </Message>`
	`</ChatEvents>`

Responses:	200 (Succeeded) or 400/500 (Failed). See HTTP responses.

GET (events) parameters

GET takes two parameters:

- all (default = false)—if true, all events since eventid are returned. If false, MessageEvents sent from the consumer (by the PUT call) are omitted from the returned events list.

  For example: `http://<ServerIP>:<Port>/ccp/chat?eventid=0&all=false`

- eventid (default = 0)— When first called, eventid is set to 0. Subsequent calls identify the eventid of the last processed event. The call will return events that occurred since the specified eventid. It is normal for there to be gaps in the eventids that are returned.

Event	Description and child events
MessageEvent	A chat message sent from the agent: • id: sequential id of the event. • from: sender of the message. • body: body of the message.
PresenceEvent	A user joined or left the session: • id: sequential id of the event. • from: user who joined/left the room. • status: joined|left.

Event	Description and child events
StatusEvent	Chat session status: • id: sequential id of the event. • status: status information. ◦ chat_finished: the chat is finished with no problems (not currently used). ◦ chat_finished_error: the chat is finished due to an error; this is not recoverable (not currently used). ◦ chat_issue: there is an issue with the chat session; this may be recoverable. ◦ chat_ok: the chat session is ok again. • detail: If the XMPP Server fails, a StatusEvent will be sent with status set to *chat_issue* and detail set to *XMPP Server is down*.

PUT (update)

This API sends a chat event to the chat room. Currently, the only chat event that can be sent from the client is a chat message of the form:

```
<Message>
 <body>body of message </body>
</Message>
```

Note Because there is no identifier to specify which chat we are getting events for, API calls only work with a valid session cookie. The session cookie returned from the POST or GET should be provided on subsequent chat proxy API calls.

URL:	http://<ServerIP>:<Port>/ccp/chat
HTTP method:	PUT

Example response payload:	`<ChatEvents>`
	` <PresenceEvent>`
	`  <id>1</id>`
	`  <from>Steve</from>`
	`  <status>joined</status>`
	` </Presence>`
	` <MessageEvent>`
	`  <id>2</id>`
	`  <from>Steve</from>`
	`  <body>Hi There</body>`
	` </Message>`
	` <MessageEvent>`
	`  <id>3</id>`
	`  <from>Steve</from>`
	`  <body>How can I help you?</body>`
	` </Message>`
	`</ChatEvents>`

Responses:	200 (Succeeded) or 400/500 (Failed). See HTTP responses.

DELETE

Stops the chat session.

Note Because there is no identifier to specify which chat we are getting events for, API calls only work with a valid session cookie. The session cookie returned from the POST or GET should be provided on subsequent chat proxy API calls.

URL:	http://<ServerIP>:<Port>/ccp/chat
HTTP method:	DELETE
HTTP response headers:	See HTTP responses.

Chat session timeout

A chat session will time out after five minutes of customer inactivity. If there are no UPDATES or GETS for five minutes, the session is terminated. This timeout value is configurable; five minutes is the default value.

CHAPTER 9

Configuration for multichannel routing

This API configures SocialMiner to listen for a connection from the media routing peripheral gateway (MR PG).

SocialMiner will only allow MR PG(s) from the configured list to connect.

Configuration for multichannel routing API commands

This section describes the supported commands for the configuration for multichannel routing API and the parameters for those commands.

Related Topics

GET, on page 47
PUT, on page 48

GET

Get the MR PG configuration.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/contactcenter/ mrconfig/default
HTTP method:	GET

Example XML response:	`<MRconfig>`
	` <enabled>true</enabled>`
	` <port>38001</port>`
	` <hostA>myhostname.abc.com</hostA>`
	` <hostB>ccx.host.xyz.com</hostB>`
	`<refURL>`
	`  http://socialminer.server.ip/ccp-webapp/`
	`  ccp/contactcenter mrconfig/default`
	`</refURL>`
	` </MRconfig>`
	See also API conventions, on page 1.

PUT

Edit the MR PG connection configuration.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/contactcenter/mrconfig/default
HTTP method:	PUT
Input and output format:	XML

Example XML request:	`<MRconfig>`
	` <enabled>false</enabled>`
	` <port>38001</port>`
	` <hostA>myhostname.abc.com</hostA>`
	` <hostB>ccx.host.xyz.com</hostB>`
	`</MRconfig>`

Parameters:	• **enabled**: a Boolean flag to enable or disable the MR configuration. ◦ If false: MR PG configuration is ignored and no connections are accepted. ◦ If true and no hostnames are set: any connection is accepted. ◦ If true and at least one hostname is specified: only connections matching the specified hosts are allowed. • **hostA, hostB**: host identifiers of the MR PG servers. Any identifier can be specified (for example, IPv4 or hostname), as long as it can be resolved to the actual IP address of the server(s). The combined length of the host strings is limited to 254 characters. • **port**: the port number that the MR PG uses to connect to SocialMiner. Defaults to 38001. The valid range is 10000 - 65535.

CHAPTER 10

Conversation (Twitter)

The Conversation API allows you to view the conversational context of a specific Twitter social contact. A conversation will be displayed starting at the selected tweet. The selected contact is checked for a reply-to ID and, if there is one, the contact corresponding to the reply-to ID is included in the conversation. This continues until a contact with no reply-to ID is reached or until the replies limit set in the API call is reached.

To stay within Twitter's rate limits, Twitter conversations for tweets are constructed using the reply-to IDs stored in SocialMiner's data store.

Reply-to IDs are stored only for contacts that are retrieved using a Twitter account or Twitter stream feed. (The reply-to ID field is only populated for contacts retrieved through a Twitter account or Twitter stream feed.)

Specifying an older tweet in the conversation will not show the newer tweets in the conversation. The conversation API lists only the replies to the selected contact. It does not list any replies that the selected contact was in-reply-to.

Note Twitter direct messages (DMs) are not linked to each other by Twitter, so the conversation aspect of DMs cannot be retrieved.

Conversation API commands

This section describes the supported command (GET) for the Twitter conversations API and the parameters for that command.

GET (list)

Run this API to list the Twitter conversations for tweets (using the reply-to IDs stored in the SocialMiner data store).

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/conversation/<scID>/?replies=xxx

HTTP method:	GET
URL parameters:	• **scID** Required, string. The ID of the social contact whose conversation will be retrieved. • **replies** Integer. The maximum number of replies to retrieve. Default is 32.

Example http request:

```
https://<ServerIP>:<Port>/ccp-webapp/ccp/conversation/
25FD59151000012E001FFC6B0A568DF5?replies=3
```

Example XML response:

```
<SocialContacts>
 <SocialContact>
  <author>ContactAuthor</author>
  <description>Text of the tweet</description>
  <id>F1217D711000012D000003AB0A568DF5</id>
  <link>
    http://twitter.com/ContactAuthor/statuses/
    33460956554076160
  </link>
  <publishedDate>1296827212000</publishedDate>
  <refURL>
   http://[ServerIP]:[port]/ccpwebapp/ccp/
   socialcontact/F1217D711000012D000003AB0A568DF5
  </refURL>
  <status>unread</status>
  <statusTimestamp>1296830659988</statusTimestamp>
  <statusUserId/>
  <tags/>
  <title>Tweet: ContactAuthor to Agent</title>
 </SocialContact>
 <SocialContact>
  <author>Agent</author>
  <description>Text of the tweet</description>
  <id>F1217D711000012D000003AB0A568DF5</id>
  <link>
   http://twitter.com/ContactAuthor/statuses/
   33460956554076160
  </link>
  <publishedDate>1296827212000</publishedDate>
  <refURL>
   http://[ServerIP]:[port]/ccpwebapp/
   ccp/socialcontact/F1217D711000012D000003AB0A568DF5
  </refURL>
  <status>unread</status>
  <statusTimestamp>1296830659988</statusTimestamp>
  <statusUserId/>
  <tags/>
  <title>Tweet: Agent to ContactAuthor</title>
 </SocialContact>
 <SocialContact>
  <author>ContactAuthor</author>
  <description>Text of the tweet</description>
  <id>F1217D711000012D000003AB0A568DF5</id>
  <link>
   http://twitter.com/ContactAuthor/statuses/
   33460956554076160
  </link>
  <publishedDate>1296827212000</publishedDate>
  <refURL>
   http://[ServerIP]:[port]/ccpwebapp/
   ccp/socialcontact/F1217D711000012D000003AB0A568DF5
  </refURL>
  <status>handled</status>
  <statusTimestamp>1296830659988</statusTimestamp>
  <statusUserId>Agent</statusUserId>
  <tags/>
  <title>Tweet: ContactAuthor</title>
 </SocialContact>
</SocialContacts>
```

HTTP response headers:

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

CHAPTER 11

Email

Use the Email API to retrieve the existing SMTP server configuration and to update it if necessary. An SMTP server connection is required to send email notifications.

This API is represented on the SocialMiner user interface in the System Administration panel.

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

Email API commands

This section describes the supported commands for Email API and the parameters for those commands.

Related Topics

GET, on page 53
PUT, on page 54
Email API parameters, on page 54

GET

Retrieves the SMTP configuration.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/email/default
HTTP method:	GET

Example XML response:	`<Email>`
	`  <smtpHost><ServerIP>:<Port></smtpHost>`
	`  <smtpPort>587</smtpPort>`
	`  <smtpFromUser>`
	`   FromUser@Here.net`
	`  </smtpFromUser>`
	`  <smtpHostUserName>`
	`   userNameForEmailServer`
	`  </smtpHostUserName>`
	`  <smtpAuthenticationEnabled>`
	`   true`
	`  </smtpAuthenticationEnabled>`
	`  <smtpEnabled>true</smtpEnabled>`
	`  <smtpSslEnabled>true</smtpSslEnabled>`
	`  <refURL>`
	`   http://<ServerIP>:<Port>/ccp-webapp/ccp/`
	`   email/default`
	`</refURL>`
	`</Email>`

Parameters:	See Email API parameters, on page 54.

PUT

Updates the SMTP configuration.

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

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/email/default
HTTP method:	PUT
Parameters:	See Email API parameters, on page 54.

Email API parameters

Parameters are optional unless otherwise noted.

Parameter	Description	Notes
smtpEnabled	Whether this SMTP configuration is enabled.	Boolean. Defaults to false.
smtpAuthenticationEnabled	Whether SMTP authentication is required for the SMTP host.	Boolean. Defaults to false.
smtpFromUser	The email (reply-to) address of email sent by this server.	Required when smtpEnabled = true.
smtpHost	The fully qualified host address of the SMTP server.	Required when smtpEnabled = true.

Parameter	Description	Notes
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.
smtpPort	The SMTP port. Default is 587.	Required when smtpEnabled = true.

CHAPTER 12

Facebook reply

The Facebook reply API works much like Twitter reply, on page 159.

See also Facebook account authorization, on page 77.

Facebook reply API commands

This section describes the supported commands for the Facebook reply API and the parameters for those commands.

Related Topics

GET, on page 57
POST (comment), on page 58
POST (like), on page 59
GET (like), on page 60
GET (user), on page 60

GET

Gets the status of a Facebook reply API call.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/reply/facebook/<id> For more information about the elements in the URL, see API conventions, on page 1. In this instance, <id> represents the ProgressID being requested.
HTTP method:	GET
Example XML response:	If the fields in the request are valid, the Location field in the response contains the URL for the social contact associated with the post being queried.

	If the operation succeeds, the response returns the following XML:
	
	`<RequestProgress>`
	` <apiErrors>`
	`      <apiError>`
	`        <ErrorType></ErrorType>`
	`         <ErrorData></ErrorData>`
	`        <ErrorMessage></ErrorMessage>`
	`      </apiError>`
	`    </apiErrors>`
	` <httpResponseCode>responseCode</httpResponseCode>`
	` <httpResponseMessage>responseMessage</httpResponseMessage>`
	` <progress>SUCCEEDED|FAILED|IN-PROGRESS</progress>`
	` <Facebook>`
	`   <postId>facebookPostId</PostId>`
	` </Facebook>`
	`</RequestProgress>`
	
	If the operation fails, the httpResponseCode and httpResponseMessage fields contain the code and message returned by Facebook. The apiErrors field can contain additional detailed information about the error.

Response payload:	• **httpResponseCode**: the response code received from Facebook. • **httpResponseMessage**: the response message received from Facebook. • **progress**: ◦ IN_PROGRESS if SocialMiner is waiting for a response from Facebook. ◦ SUCCEEDED if the Facebook operation succeeded. ◦ FAILED if the Facebook operation failed. Use httpResponseCode and httpResponseMessage to determine why the operation failed. • **facebookReplyPostId:** the Facebook Post ID of the comment. • **facebookErrorType:** the Facebook error type returned from Facebook if the httpResponseCode is NOT_FOUND. • **facebookErrorMessage:** the Facebook error message returned from Facebook if the httpResponseCode is NOT_FOUND.

POST (comment)

Sends a status message (reply or comment) for a Facebook post.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/reply/facebook/comment For more information about the elements in the URL, see API conventions, on page 1.
HTTP method:	POST
Parameters:	• **socialContact**: Is the RefURL of the social contact for the Facebook post (each Facebook post is represented by a different social contact in SocialMiner).

	• **message**: The text of the status message.

Request payload:	`<Comment>`
	` <socialContact>socialContactRefUrl</socialContact>`
	` <message>status text</message>`
	`</Comment>`

Response:	If the fields in the request are valid, the Location field in the response contains the URL for the social contact associated with the post.
	If the operation succeeds, the response returns the following XML:
	`<RequestProgress>`
	` <apiErrors>`
	`  <apiError>`
	`   <ErrorType></ErrorType>`
	`   <ErrorData></ErrorData>`
	`   <ErrorMessage></ErrorMessage>`
	`  </apiError>`
	` </apiErrors>`
	` <progress>SUCCEEDED</progress>`
	`</RequestProgress>`
	If the operation fails, the httpResponseCode and httpResponseMessage fields contain the code and message returned by Facebook. The apiErrors field can contain additional detailed information about the error.

POST (like)

Use this to 'like' a specific post on Facebook.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/reply/facebook/like
HTTP method:	POST
Parameters:	• **socialContact**: Is the RefURL of the social contact for the Facebook post being 'liked'. • **likes**: Like or Unlike a post. Boolean.

Request payload:	`<Like>`
	` <socialContact>socialContactRefUrl</socialContact>`
	` <likes>booleanValue</likes>`
	`</Like>`

Response:	If the fields in the request are valid, the Location field in the response contains the URL for the social contact associated with the post being 'liked'.
	If the operation succeeds, the response returns the following XML:
	`<RequestProgress>`
	` <apiErrors>`
	`  <apiError>`
	`   <ErrorType></ErrorType>`
	`   <ErrorData></ErrorData>`
	`   <ErrorMessage></ErrorMessage>`

	```
	  </apiError>
	 </apiErrors>
	 <progress>SUCCEEDED</progress>
	</RequestProgress>
	```
	If the operation fails, the httpResponseCode and httpResponseMessage fields contain the code and message returned by Facebook. The apiErrors field can contain additional detailed information about the error.

GET (like)

Use this to determine if the authorized user already likes a post on Facebook.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/reply/facebook/like
HTTP method:	GET
Parameters:	**socialContact**: the RefURL of the social contact being queried.

Response:	If the fields in the request are valid, the Location field in the response contains the URL for the social contact.
	If the operation succeeds, the response returns the following XML:
	`<RequestProgress>`
	` <apiErrors>`
	`  <apiError>`
	`   <ErrorType></ErrorType>`
	`   <ErrorData></ErrorData>`
	`   <ErrorMessage></ErrorMessage>`
	`  </apiError>`
	` </apiErrors>`
	` <progress>SUCCEEDED|FAILED|IN-PROGRESS</progress>`
	`  <Facebook>`
	`   <like>value</Like>`
	`  </Facebook>`
	`</RequestProgress>`
	Note `like` will contain a boolean value of *true* if the user already likes the post and *false* if the user does not already like the post .
	If the operation fails, the httpResponseCode and httpResponseMessage fields contain the code and message returned by Facebook. The apiErrors field can contain additional detailed information about the error.

GET (user)

Retrieves the profile information of the Facebook user.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/reply/facebook/user/
HTTP method:	GET
Parameters:	**socialContact** : the RefURL of the social contact for the user whose public Facebook profile data will be retrieved.

Example http request:	`http://<ServerIP>:<Port>/ccp-webapp/ccp/reply/facebook/user?socialContact=https://xx.yy.zzz.ggg/ccp-webapp/ccp/socialcontact/C077693810000140000081D0A568DDD`
	If the fields in the request are valid, the Location field of the response contains the URL to poll for the status of the operation, for example:
	`http://<ServerIP>:<Port>/ccp-webapp/ccp/reply/facebook/6`
	Otherwise, SocialMiner will return an error. See HTTP responses for error response.

Example XML response:	If the operation fails, the httpResponseCode and httpResponseMessage fields contain the code and message returned by Facebook. The apiErrors field can contain additional detailed information about the error.
	If the operation with the polling URL (http://<ServerIP>:<Port>/ccp-webapp/ccp/reply/facebook/6) succeeds, the response returns the following XML:
	`<RequestProgress>`
	`<apiErrors/>`
	` <httpResponseCode>200</httpResponseCode>`
	` <httpResponseMessage>OK</httpResponseMessage>`
	` <progress>SUCCEEDED</progress>`
	` <facebook/>`
	`  <facebookUser>`
	`   <id>100004025925472</id>`
	`   <name>Soso Cannons</name>`
	`   <username>soso.cannons</username>`
	`   <link>http://www.facebook.com/soso.cannons</link>`
	`   <profileImageUrl>http://profile.ak.fbcdn.net/hprofile-ak-prn2/`
	`    274695_100004025925472_1641583011_q.jpg</profileImageUrl>`
	`  </facebookUser>`
	`</RequestProgress>`

CHAPTER 13

Feed

The Feed API allows you to create, delete, update, and list feeds that retrieve contacts. SocialMiner feeds can be RSS feeds, Twitter accounts, Twitter streams, Twitter searches, Facebook fan pages, push feeds, chat feeds, or callback feeds. (Callback feeds support the callback API. See Callback, on page 13 for more information.)

The feed object contains data about the feed—such as the URL of the feed, how often the feed is to be read (the polling interval), whether SocialMiner needs to access the feed through a proxy, and the feed type.

This API is represented on the SocialMiner user interface by the Feeds panel.

Feeds are assigned to at least one Campaign. To acquire contacts, you must create a feed, create a campaign, and add the feed to the campaign.

Feed API commands

This section describes the commands supported for the Feed API and the parameters for those commands. Additional information about push feeds is documented in Push feed.

POST

Creates a feed to be stored in the database.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/feed
HTTP method:	POST
Parameters:	See Feed API parameters, on page 69.

Example XML request payload:	`<Feed>`
	`  <name>test</name>`
	`  <description>this is the description`
	`  </description>`
	`  <url>http://test.com</url>`
	`  <type>5</type>`
	`  <pollingInterval>60</pollingInterval>`
	`  <minAge>1</minAge>`
	`  <authenticationUsername>User1`
	`  </authenticationUsername>`
	`  <authenticationPassword>password1`
	` </authenticationPassword>`
	`  <replytemplateRefURL>`
	`   http://[ServerIP]:[Port]/`
	`   ccp-webapp/ccp/template/reply/105678`
	`  </replytemplateRefURL>`
	`  <tags>`
	`    <tag>tag1</tag>`
	`    <tag>tag2</tag>`
	`  </tags>`
	`</Feed>`

HTTP response headers:	The response contains the **URL** for the newly created feed. Note the id: *100162*.
	
	`http/1.1 201 Created`
	`Location: http://<ServerIP>:<Port>`
	` /ccp-webapp/ccp/feed/100162`
	`Content-Type: text/plain`
	`Content-Length: 0`
	`Date: Tue, 12 Jan 2010 16:15:04 GMT`
	See also HTTP responses.

For feeds that require authorization (such as Twitter feeds and Facebook fan page feeds), this API creates a feed pending on OAuth. The webapp initiates an OAuth session with Twitter or Facebook. SocialMiner allows for one feed per Twitter user account; therefore the webapp evaluates whether or not the username has already been used. Once the OAuth successfully completes, the pending feed is finalized and added to the system.

There are several extra steps for creating Twitter account feeds and Facebook fan page feeds. See Authorize against Twitter feeds and Facebook account authorization.

DELETE

Deletes a feed from the database.

For feeds that require authorization (Twitter account and Facebook fan page feeds), this function deletes the specified feed if it has passed OAuth, or cancels the pending authentication if it has not passed OAuth.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/feed/Id variables
HTTP method:	DELETE
HTTP response headers:	See HTTP responses.

GET (list)

Retrieves a list of all feeds in the system.

URL:	http://<ServerIP>:<Port>/<webapp>/ccp/feed?summary=<true/false>
HTTP method:	GET
Parameters:	See Feed API parameters, on page 69.

Example:	`http://<ServerIP>:<Port>/ccp-webapp/ccp/feed?summary=false`

Example XML response:	```<feeds>```
	
	```<Feed>```
	```<changeStamp>0</changeStamp>```
	```<name>Boston.com Most Popular</name>```
	```<pollingInterval>300</pollingInterval>```
	
	```<refURL>```
	```http://[ServerIP]:[Port]/ccp-webapp/ccp/feed/100347```
	```</refURL>```
	```<status>1</status>```
	```<tags/>```
	```<type>1</type>```
	```<url>http://feeds.boston.com/boston/mostpopular</url>```
	```</Feed>```
	
	```<Feed>```
	```<changeStamp>2</changeStamp>```
	```<name>Boston.com top stories</name>```
	```<pollingInterval>300</pollingInterval>```
	
	```<refURL>```
	```http://[ServerIP]:[Port]/ccp-webapp/ccp/feed/100346```
	```</refURL>```
	```<status>1</status>```
	```<tags/>```
	```<type>1</type>```
	```<url>http://feeds.boston.com/boston/topstories</url>```
	```</Feed>```
	
	```<Feed>```
	```<changeStamp>0</changeStamp>```
	```<name>Cisco Live</name>```
	```<pollingInterval>300</pollingInterval>```
	
	```<refURL>```
	```http://[ServerIP]:[Port]/ccp-webapp/ccp/feed/102638```
	```</refURL>```
	
	```<replyTemplateRefURL>```
	```http://[ServerIP]:[Port]/ccp-webapp/ccp/template/reply/301```
	```</replyTemplateRefURL>```
	```<status>1</status>```
	```<tags/>```
	```<type>4</type>```
	```<url>http://www.facebook.com/ciscoliveeurope</url>```
	```</Feed>```
	
	```<Feed>```
	```<changeStamp>0</changeStamp>```
	```<name>contacts</name>```
	
	```<pushFeedURL>```
	```http://[ServerIP]:[Port]/ccp-webapp/ccp/pushfeed/102619```
	```</pushFeedURL>```
	
	```<refURL>```
	```http://[ServerIP]:[Port]/ccp-webapp/ccp/feed/102619```
	```</refURL>```
	```<status>1</status>```
	```<tags/>```
	```<type>7</type>```
	```</Feed>```
	
	```<Feed>```
	```<changeStamp>2</changeStamp>```
	```<name>Twitter boston</name>```
	```<pollingInterval>180</pollingInterval>```

	`<refURL>`
	`http://[ServerIP]:[Port]/ccp-webapp/ccp/feed/102621`
	`</refURL>`
	
	`<replyTemplateRefURL>`
	`http://[ServerIP]:[Port]/ccp-webapp/ccp/template/reply/103184`
	`</replyTemplateRefURL>`
	`<status>1</status>`
	`<tags/>`
	`<type>1</type>`
	`<url>http://search.twitter.com/search.rss?q=boston</url>`
	`</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`
	See also HTTP responses.

GET

Returns the data for a single feed. For security, passwords are not returned for feeds. Password elements are masked (******).

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/feed/<id> For more information about <id>, see Id variables, on page 2.
HTTP method:	GET

Example XML response:	
	```
	<Feed>
	  <refURL>
	   http://[ServerIP]:[Port]/ccp-webapp/ccp/
	   feed/(id)
	  </refURL>
	  <name>test</name>
	  <description>this is the description</description>
	  <url>http://test.com</url>
	  <type>1</type>
	  <pollingInterval>60</pollingInterval>
	  <useProxy>false</useProxy>
	  <minAge>1</minAge>
	  <changeStamp>0</changeStamp>
	  <sessionToken>********</sessionToken>
	  <status>0</status>
	  <replytemplateRefURL>http://[ServerIP]:[Port]/
	   ccp-webapp/ccp/template/reply/105678
	  </replytemplateRefURL>
	  <tags>
	    <tag>tag1</tag>
	    <tag>tag2</tag>
	  </tags>
	</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
	```
	See also HTTP responses.

PUT

Updates an existing feed.

For feeds that require authorization (Twitter account and Facebook fan page feeds), this API updates an existing feed which has passed OAuth. If the username is changed, or if the input XML document includes "<status>3</status>", the create procedure is invoked for re-OAuth. Otherwise, it works in the same way as updating a feed without OAuth.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/feed/<id> For more information about <id>, see Id variables, on page 2.
HTTP method:	PUT
Parameters:	See Feed API parameters, on page 69.

Example XML request payload:	`<Feed>`
	`  <name>this is the new name</name>`
	`  <description>`
	`   this is an updated description`
	`  </description>`
	`  <url>http://test.com</url>`
	`  <type>1</type>`
	`  <pollingInterval>60</pollingInterval>`
	`  <useProxy>false</useProxy>`
	`  <minAge>1</minAge>`
	`  <changeStamp>0</changeStamp>`
	`  <authToken>WJFH837923</authToken>`
	`  <status>0</status>`
	`  <replytemplateRefURL>`
	`   http://[ServerIP]:[Port]/`
	`   ccp-webapp/ccp/template/reply/105678`
	`  </replytemplateRefURL>`
	`  <tags>`
	`    <tag>tag1</tag>`
	`    <tag>tag2</tag>`
	`  </tags>`
	`</Feed>`

HTTP response headers:	`http/1.1 200 OK`
	`Content-Type: text/plain`
	`Content-Length: 0`
	`Date: Thu, 14 Jan 2010 15:49:17 GMT`
	See also HTTP responses.

Feed API parameters

This table defines the parameters used by the feed API. The table below this one identifies which parameters apply to each feed type and whether they are required or optional.

Parameter name	Description	Notes
authenticationPassword	The password for the username provided for an account.	String
authenticationUsername	The username for an account.	String
authToken	This is the oAuth access token.	String
changeStamp	The change stamp of the feed record.	Integer. Defaults to 0. Required for PUT (update). Is returned in GET.
chatJoinTimeout	The amount of time (in seconds) that the agent has to join the chat room.	

Parameter name	Description	Notes
chatInactivityTimeout	The amount of idle time (in seconds) between chat messages. If a chat message is not sent in this amount of time, the chat session is taken down.	
description	The description of the feed.	String
keywords	The search criteria required by TWITTER_SEARCH, or the comma separated list of keywords to search for with the TWITTER_STREAM.	Restrictions apply but based on the restrictions by Twitter on their search or stream api respectively. There must be a least one keyword defined. For TWITTER_STREAM, up to 200 keywords can be defined for a total limit of 2000 bytes. Each keyword must be between 1-60 bytes. For TWITTER_SEARCH, up to 1000 characters are allowed to form the search query, including operators. Spaces are interpreted as an "and" modifier for search.
minAge	The minimum post age in seconds (defaults to 0).	If a post is newer than the *minAge*, it will not be stored by the feed.
name	The name of the feed.	String, must be unique. Required for creating (POST).
pollingInterval	The amount of time in seconds the system waits between attempts to read this feed.	Integer
pushFeedURL	The URL to which you push the entities that will become social contacts.	
refURL	A copy of the URL requested.	Response for GET.
replytemplateRefURL	The URL of the reply template used to respond to social contacts obtained from this feed.	String If this field is blank, no reply template is used.

Parameter name	Description	Notes
status	The authorization status for Facebook, Twitter search, and Twitter account.	Values are: • AUTHENTICATION_ SUCCEEDED=0 • AUTHENTICATION_ NONE=1 • AUTHENTICATION_ FAILED=2 • AUTHENTICATION_ PENDING = 3
summary	Determines whether full object information or URLs only are returned for the list.	Boolean. Defaults to false. When “true”, only the URLs of the objects are returned. If summary=false, full object information, along with the URL reference, is returned. URL Parameter. Used for List API only.
tags	Contacts coming in from this feed will automatically be tagged with these default tags.	String. A maximum of 10 tags are allowed.
type	The feed type.	Required for POST Integer, types are: • RSS = 1 • Twitter stream = 3 • Facebook fan page = 4 • Authenticated RSS = 5 • Twitter account = 6 • Push = 7 • Chat = 8 • Twitter search = 9 • Callback = 10
url	The location of the feed that you want to read.	String If type is FACEBOOK, you cannot use an IP address; you must use a hostname.

This table summarizes the fields used by each feed type as R (required), O (optional), or NA (not applicable).

Feed type/ Field name	RSS (1)	Twitter stream (3)	Facebook (4)	Auth RSS (5)	Twitter account (6)	Push (7)	Chat (8)	Twitter search (9)	Callback (10)
type	**R**	**R**	**R**	**R**	**R**	**R**	**R**	**R**	**R**
name	**R**	**R**	**R**	**R**	**R**	**R**	**R**	**R**	**R**
description	**O**	**O**	**O**	**O**	**O**	**O**	**O**	**O**	**O**
url	**R**	NA	**R**	**R**	NA	NA	NA	NA	NA
polling interval	**R**	NA	**R**	**R**	**R**	NA	NA	**R**	NA
minAge	**O**	NA	**O**	**O**	**O**	NA	NA	NA	NA
authentication Username	NA	**R**	NA	**R**	**R**	NA	NA	**R**	NA
authentication Password	NA	NA	NA	**O**	NA	NA	NA	NA	NA
keywords	NA	**R**	NA	NA	NA	NA	NA	**R**	NA
authToken	NA	NA	NA	NA	NA	NA	NA	NA	NA
chatJoin Timeout	NA	NA	NA	NA	NA	NA	**O**	NA	NA
chat Inactivity Timeout	NA	NA	NA	NA	NA	NA	**O**	NA	NA
reply Template RefUrl	**O**	**O**	**O**	**O**	**O**	**O**	**O**	**O**	**O**
tags	**O**	**O**	**O**	**O**	**O**	**O**	**O**	**O**	**O**

Authorize against Twitter feeds

Twitter uses OAuth to provide secure communication between Twitter and third-party applications such as SocialMiner. You must obtain authorization to access a Twitter account for any kind of Twitter feed type (account, stream, or search).
For this reason, there are several additional steps to creating a Twitter feed:

Procedure

Step 1 Create a Twitter feed.
See POST, on page 63 for more information about how to do this.

Step 2 Use oauthGetStatus, on page 74 to poll the feed status until the *status* is WAITING-FOR-AUTH-CALLBACK. The authUrl field is populated with the twitter authorization URL.

Step 3 Access the **authUrl** through a browser session and allow authorization for SocialMiner. Twitter returns a PIN code through oauthCallback, on page 75. This is handled on the server and does not require any API calls from your application.

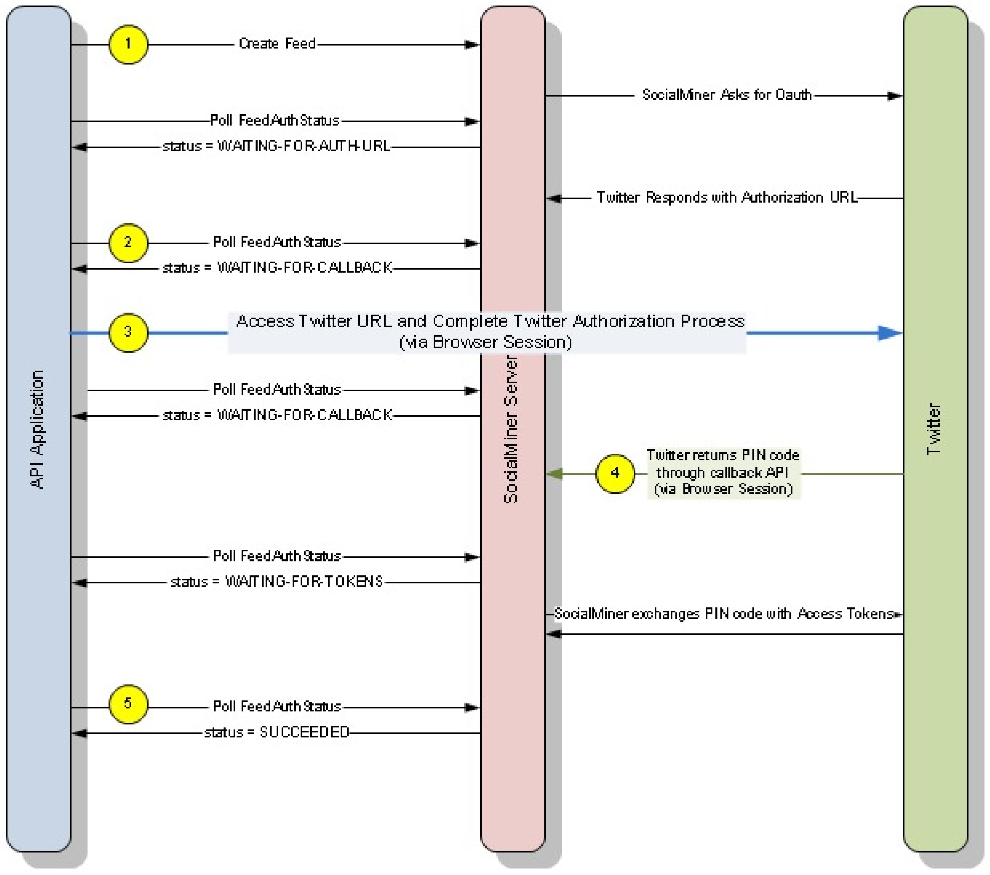
Step 4 Poll with oauthGetStatus, on page 74 until the status is *SUCCEEDED*. The feed has been created.

Note The Shindig OpenSocial container in which SocialMiner runs requires that REST requests complete within five seconds. Communication with the Twitter servers can exceed five seconds. This limitation means that after making calls to the Feed API for creating Twitter feeds, you must poll to verify the status returned.

Twitter account authorization communication

This diagram illustrates the API calls and expected poll responses for Twitter account authorization.

Figure 1: Twitter Account Authorization Communication diagram


API Application
SocialMiner Server
Twitter
1
Create Feed
SocialMiner Asks for Oauth
Poll FeedAuth Status
status = WAITING-FOR-AUTH-URL
Twitter Responds with Authorization URL
2
Poll FeedAuth Status
status = WAITING-FOR-CALLBACK
3
Access Twitter URL and Complete Twitter Authorization Process (via Browser Session)
Poll FeedAuth Status
status = WAITING-FOR-CALLBACK
4
Twitter returns PIN code through callback API (via Browser Session)
Poll FeedAuth Status
status = WAITING-FOR-TOKENS
SocialMiner exchanges PIN code with Access Tokens
5
Poll FeedAuth Status
status = SUCCEEDED

Twitter authorization APIs

oauthGetStatus

Used for Twitter account, search, and stream feeds. Checks the status of the authorization with Twitter. Your application should poll this URL until status is "SUCCEEDED."

The webapp first checks the hashmap using the input username as a key to see if the feed is there. If it is, the authorization is still pending; otherwise, it checks the database. If the feed is found in the database, the authorization is done successfully. The authorization fails if the feed is not found anywhere.

If the authenticating user is different from the configured user for the feed, the user is allowed to log into Twitter and click "Allow", but SocialMiner will fail the authentication and set the status code to "FAILED-USER-MISMATCH."

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/feed/oauth/<username>
HTTP method:	GET

Example XML response:	`<FeedOAuthStatus>`
	`    <status>STATUS</status>`
	`</FeedOAuthStatus>`
	• **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.

oathCancel

Used for Twitter account, search, and stream feeds. Cancels 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://<ServerIP>:<Port>/ccp-webapp/ccp/feed/oauth/<username>
HTTP method:	DELETE
Response:	Response is contained in the http response code. See also HTTP responses.

oauthCallback

Used for Twitter account, search, and stream feeds. 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://<ServerIP>:<Port>/ccp-webapp/ccp/feed/oauth/<username>/callback
HTTP method:	GET
Responses:	*SUCCEEDED* or *Failed*.

Authorize Facebook accounts

Facebook uses OAuth to provide secure communication and is similar to Twitter account authorization.

For Facebook fan page feeds, this API creates the feed and starts the OAuth process. The feed is saved in a hashmap indexed by the lookup key. The webapp then creates a future task which initiates an OAuth session with Facebook. This generates an authentication link and saves it with the original hashed feed configuration. Upon success, the function returns the URL for checking the OAuth status and sets the http response code to 202 (Accepted).

There are several additional steps to creating a Facebook fan page feed:

Procedure

Step 1 Create a type 4 feed.
See POST, on page 63 for more information about how to do this.

Step 2 Use FacebookOauthGetStatus, on page 77 to poll the feed status until the status is WAITING-FOR-AUTH-CALLBACK. The authUrl field is populated with the Facebook authorization URL.

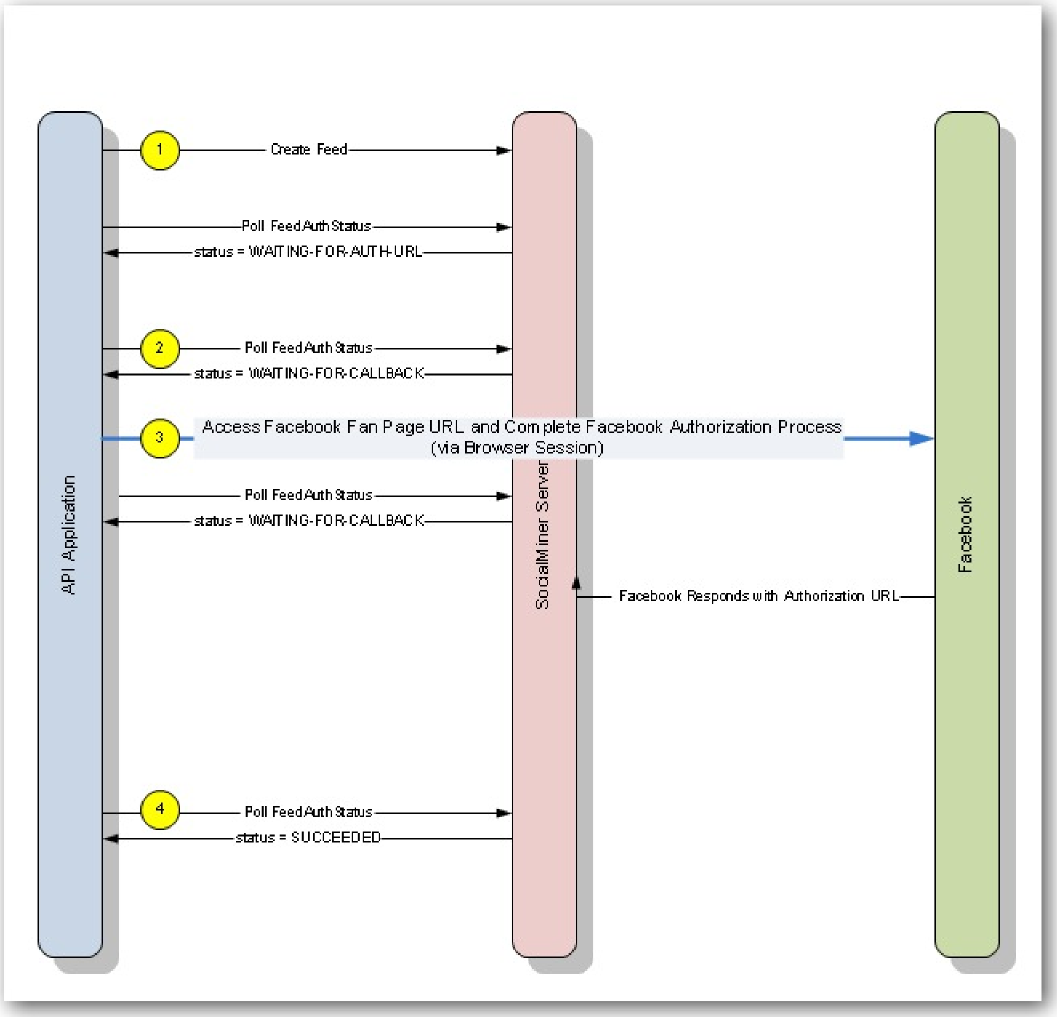
Step 3 Access the **authUrl** through a browser session and allow authorization for SocialMiner. Facebook returns a PIN code through FacebookCallback, on page 78. This is handled on the server and does not require any API calls from your application.

Step 4 Poll with FacebookOauthGetStatus, on page 77 until the status is "SUCCEEDED". The feed has been created.

Facebook account authorization

This diagram illustrates the API calls and expected poll responses for Facebook account authorization.

Figure 2: Facebook account authorization diagram


API Application
SocialMiner Server
Facebook
1
Create Feed
Poll FeedAuthStatus
status = WAITING-FOR-AUTH-URL
2
Poll FeedAuthStatus
status = WAITING-FOR-CALLBACK
3
Access Facebook Fan Page URL and Complete Facebook Authorization Process (via Browser Session)
Poll FeedAuthStatus
status = WAITING-FOR-CALLBACK
Facebook Responds with Authorization URL
4
Poll FeedAuthStatus
status = SUCCEEDED

Facebook authorization APIs

FacebookOauthGetStatus

Used only for the Facebook fan page feed type. Checks the status of the authorization with Facebook. Your application should poll this URL until status is "SUCCEEDED".

The webapp first checks the hashmap using the input lookup key to see if the feed is there. If it is, the authorization is still pending; otherwise, it checks the database. If the feed is found in the database, the authorization is successful. The authorization fails if the feed is not found anywhere.

If the lookup key is different from the one configured for the feed, the user is allowed to log into Facebook and click **Allow**, but SocialMiner will fail the authentication and set the status code to FAILED-LOOKUP-KEY-MISMATCH.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/feed/<lookup-key>/facebookOauth
HTTP method:	GET
Example XML response:	`<FeedOAuthStatus>` `<authUrl>AUTH_URL</authUrl>` `<status>STATUS</status>` `</FeedOAuthStatus>` • **authURL** is the authorization URL from Facebook. • **status** is one of: ◦ SUCCEEDED—The application/user account has been authorized for use with Facebook ◦ FAILED—The application/user is not authorized for use with Facebook. This could be due to a failed login or a timeout. The authorization request times out in 10 minutes. ◦ FAILED-LOOKUP-KEY-MISMATCH—The lookup key provided does not match the one for the user logged into Facebook. ◦ IN-PROGRESS—The authorization is in progress.

FacebookOauthCancel

This API cancels a pending OAuth and sets the status to failed. If the OAuth has already completed successfully when the cancel request is received, the saved configuration will not be deleted.

URL:	http://<ServerIP>:<Port>/<webapp>/ccp/feed/<id>/facebookOauthCancel For more information about <id>, see Id variables, on page 2.
HTTP method:	DELETE
Request payload:	None
Responses:	See HTTP responses, on page 4.

FacebookCallback

Used only for the Facebook account feed, this URL is the callback URL invoked by Facebook 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://<ServerIP>:<Port>/<webapp>/ccp/feed/<id>/facebookCallback For more information about <id>, see Id variables, on page 2.
HTTP method:	POST
Request payload:	The Facebook-assigned authorization PIN code carried in the “access_code” parameter or the error in parameter “error_description”.
Responses:	“Succeeded” or “Failed”.

CHAPTER 14

Filter

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

This API is represented on the SocialMiner user interface in the Filters panel.

Filter API commands

This section describes the supported commands for filter API and the parameters for those commands.

Note that for the POST and PUT request payloads for script filters (type = 4), you must enclose the script content in a CDATA wrapper. Failure to do this results in parser errors.

Related Topics

POST, on page 81
DELETE, on page 82
GET (list), on page 82
GET, on page 83
GET (script content), on page 84
PUT, on page 84
Filter API parameters, on page 85

POST

Creates a filter to be stored in the database.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/filter
HTTP method:	POST
Parameters:	See Filter API parameters, on page 85.

Example XML request payload for Bayesian filter:	`<Filter>`
	`   <name>Bayesian</name>`
	`   <description>Bayesian</description>`
	`   <type>2</type>`
	`</Filter>`

Example XML request payload for author filter:	`<Filter>`
	`   <name>String</name>`
	`   <description>String</description>`
	`   <type>3</type>`
	`   <keywords>`
	`     <keyword>author name</keyword>`
	`     <keyword>author userid</keyword>`
	`   </keywords>`
	`   <rule>1</rule>`
	`</Filter>`

Example XML request payload for script filter, showing the CDATA wrapper:	`<Filter>`
	`   <name>String</name>`
	`   <description>String</description>`
	`   <type>4</type>`
	`   <scriptFileName>me.groovy</scriptFileName>`
	`   <scriptContent>`
	`   <![CDATA[`
	`       put the text of your filter here.`
	`   ]]>`
	`   </scriptContent>`
	`</Filter>`

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

DELETE

Removes a filter from the database.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/filter/Id variables
HTTP method:	DELETE

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

GET (list)

Returns a list all filters.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/filter

HTTP method:	GET
URL parameter:	True or false. Defaults to false. When “true”, only the URLs of the objects are returned. When “false”, full object information is returned along with the URL reference.
Parameters:	See Filter API parameters, on page 85.
Example:	`http://<ServerIP>:<Port>/ccp-webapp/ccp/filter?summary=false`

Example XML response:

```
<Filters>
  <Filter>
    <changeStamp>0</changeStamp>
    <description>this is an AUTHOR filter!</description>
    <keywords>
      <keyword>author name</keyword>
      <keyword>author userid</keyword>
    </keywords>
    <name>filter1</name>
    <refURL>
     http://[ServerIP]:[Port]/ccp-webapp/ccp/filter/[id]
    </refURL>
    <rule>1</rule>
    <type>3</type>
  </Filter>
  <Filter>
    <changeStamp>0</changeStamp>
    <description>this is a SCRIPT filter!</description>
    <name>filter2</name>
    <ScriptFileName>me.groovy</ScriptFileName>
    <scriptContent>
      <scriptContentRefURL>
       http://[ServerIP]:[Port]/ccp-webapp/ccp/filter/
       [id]/scriptcontent
      </scriptContentRefURL>
    </scriptContent>
    <refURL>
     http://[ServerIP]:[Port]/ccp-webapp/ccp/filter/[id]
    </refURL>
    <type>4</type>
  </Filter>
</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
```

GET

Returns the data for a single filter.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/filter/<Id variables>
HTTP method:	GET

Example:	`http://<ServerIP>:<Port>/ccp-webapp/ccp/`
	`filter/100036`

Parameters:	See Filter API parameters, on page 85.

Example XML response for an author filter:	`<Filter>`
	`  <changeStamp>0</changeStamp>`
	`  <name>filter1</name>`
	`  <refURL>`
	`   http://[ServerIP]:[Port]/ccp-webapp/ccp/filter/[id]`
	`  </refURL>`
	`  <type>3</type>`
	`  <keywords>`
	`     <keyword>author name</keyword>`
	`     <keyword>author userid</keyword>`
	`   </keywords>`
	`   <rule>1</rule>`
	`</Filter>`

Example XML response for a script filter:	`<Filter>`
	` <changeStamp>0</changeStamp>`
	` <description>`
	`  this is a SCRIPT filter!`
	` </description>`
	` <name>filter2</name>`
	` <scriptFileName>me.groovy</scriptFileName>`
	` <scriptContentRefURL>`
	`  http://[ServerIP]:[Port]/ccp-webapp/ccp/filter/`
	`  [id]/scriptcontent`
	` </scriptContentRefURL>`
	` <refURL>`
	`  http://[ServerIP]:[Port]/ccp-webapp/ccp/filter/[id])`
	` </refURL>`
	` <type>4</type>`
	`</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`

GET (script content)

This request is valid for script filters only and returns the script content.

URL:	http://<ServerIP>:<Port>/<webapp>/ccp/filter/(id)/scriptcontent
HTTP method:	GET

Response for a script filter:	`function{`
	`if((a<b) && (c>d) ) {`
	`after=before;`
	`before=after;`
	`}`
	`}`

PUT

Updates an existing filter.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/filter/<Id variables>
HTTP method:	PUT
Parameters:	See Filter API parameters, on page 85.
Request payload for Bayesian filter:	`<Filter>` `<name>Bayesian2</name>` `<description>a different description</description>` `<changeStamp>0</changeStamp> <type>2</type>` `</Filter>`
Request payload for author filter:	`<Filter>` `<name>String</name>` `<description>String</description>` `<type>3</type>` `<changeStamp>12345</changeStamp>` `<keywords>` `<keyword>author name</keyword>` `<keyword>author userid</keyword>` `<keyword>alternate form on author name</keyword>` `</keywords>` `<rule>1</rule>` `</Filter>`
Request payload for script filter:	`<Filter>` `<changeStamp>12345</changeStamp>` `<name>String</name>` `<description>A new description</description>` `<type>4</type>` `<scriptFileName>me.groovy</scriptFileName>` `<scriptContent>` `<![CDATA[ put the updated text of the script here.]]>` `</scriptContent>` `</Filter>`
HTTP response headers:	`http/1.1 200 OK` `Content-Type: text/plain` `Content-Length: 0` `Date: Thu, 14 Jan 2010 15:49:17 GMT`

Filter API parameters

Parameters are optional unless otherwise noted.

Parameter	Description	Notes
name	The name of the filter. Must be unique.	Required for POST.
description	A description of the filter.	
keywords/keyword	List of keywords for include or exclude rule filtering.	Currently used only for author filters.
changeStamp	The change stamp of the filter record. Integer. Defaults to 0.	A changeStamp, on page 3 is required for PUT (update API). changeStamp is returned in GET.

Parameter	Description	Notes
refURL	The URL requested.	
rule	Integer. Sets the filter rule. Values can be: • 0 = UNKNOWN • 1 = EXCLUDE: exclude this author from the campaign results. • 2 = INCLUDE: include this author from the campaign results.	Required only for author filters.
scriptContent	Contains the text of script filter. Only one script filter is allowed per filter. Currently only supports groovy scripts.	Used only for script filters. Required for POST (create) for script filters.
scriptContentRefURL	Pointer to the script content.	Used only for script filters and is returned by GET for a script filter id. To review the script content, use GET (script content), on page 84.
scriptFileName	Should be the same as the uploaded script filter file name.	Used only for script filters. Required for POST (create) for script filters.
type	The type of filter. Must be one of: • 0 = UNKNOWN • 1 = WORD_COUNT_ LESS_THAN_SIX • 2 = Bayesian • 3 = Author • 4 = SCRIPT	Required for POST (create).

About script filters

Script filters are a special type of SocialMiner filter that can execute arbitrary code and modify a social contact. The campaign subsystem sends a filter request to the filter subsystem for a filter of this type. The configured script indicates what script is to be run to filter the social contact.

Related Topics

Script binding, on page 87
Develop and test script filters, on page 88
Script filter security, on page 90
Sample script filters, on page 91

Script binding

When it is run, a script filter has access to all of the objects in its binding. A binding is a map of variable names to objects that are passed to the script engine and that can be accessed and modified (by name) from the script. This defines the API available to the script.

The objects in the binding are:

- log—an object that can be called like a method and passed a string that will be logged by the filter. The output is logged in the application logs (with the name of the script name identifying the message) and is returned in the XML response to the filter results API call.
- restClient—an instance of groovyx.net.http.RESTClient that can be used by the script to make REST calls to third party APIs. This object is specific to Groovy; it is more of a convenience for making REST calls.
- socialContact—An object of type ScriptFilterSocialContact that has fields exposed. A copy of this object will be placed in the binding when the script is called. When the script exits, if the information in the copy has been modified, the SocialContact modifications are retained to the data store. The exposed fields are:
    - author
    - categories
    - description
    - * publishedDate
    - tags—Tags are of type List<String>. Duplicate tags are removed during the post-processing stages of the script engine.
    - title
    - * link
    - * sourceLink

Note that fields marked with * are read-only. Changes to these fields are not retained.

Filter execution is multi-threaded, the order is non-deterministic, and the campaign status is set after filtering is complete. For these reasons, the filterResults and campaignResults fields are omitted.

Develop and test script filters

You can test scripts as you development them by using the filter results API and passing it a social contact id. Doing this will run the filter on the social contact and return the results in XML, along with the output of any logs or exceptions output by the script.

Note Errors can result if an editor such as Notepad is used to edit a script written in a language containing multi-byte characters. Eclipse and Notepad++ are the most reliable editors to use for editing scripts.

The procedure to test script filters during development is as follows:

Procedure

Step 1 Run a Campaign results GET, on page 31 to obtain a social contact link. Each <entry> has a <link rel="socialcontact" href="socialcontactRefURL">.

Step 2 Run a Filters GET (list), on page 82 to obtain the refURL of the script filter you want to test.

Step 3 Run a Filter Results GET, on page 95 and add the refURL of the social contact as a parameter.

Example:

```
http://<ServerIP>:<Port>/ccp-webapp/ccp/filter/

103105/results?socialContact=

http://<ServerIP>:<Port>/ccp-webapp/ccp/socialcontact/

AB1C35141000013200000F450A568DD2.
```

Step 4 Enter that URL in the address bar of a browser to view the XML filter result.

Example:

The < logBuffer> is all the output from the script, including logging and exceptions.

```
 <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <FilterResult>
<logBuffer>
Bad script.
Security rules violation exception: startup failed:
General error during canonicalization:
Indirect import checks prevents
usage of expression java.lang.SecurityException:
Indirect import checks prevents usage of expression
at org.codehaus.groovy.control.customizers.
SecureASTCustomizer$SecuringCode
Visitor.assertExpressionAuthorized
(SecureASTCustomizer.java:682)
at org.codehaus.groovy.control.customizers.
SecureASTCustomizer
$SecuringCodeVisitor.visitConstructorCallExpression
SecureASTCustomizer.java:845)
at org.codehaus.groovy.ast.expr.
ConstructorCallExpression.visit
(ConstructorCallExpression.java:43)
at org.codehaus.groovy.control.customizers.
SecureASTCustomizer
$SecuringCodeVisitor.visitThrowStatement
(SecureASTCustomizer.java:804)
at org.codehaus.groovy.ast.stmt.ThrowStatement.visit
```

```
(ThrowStatement.java:41)
at org.codehaus.groovy.control.customizers.
SecureASTCustomizer
$SecuringCodeVisitor.visitBlockStatement
(SecureASTCustomizer.java:705)
at org.codehaus.groovy.ast.stmt.BlockStatement.
visit(BlockStatement.java:69)
at org.codehaus.groovy.control.customizers.
SecureASTCustomizer.call
(SecureASTCustomizer.java:549)
at org.codehaus.groovy.control.CompilationUnit.
applyToPrimaryClassNodes
(CompilationUnit.java:957)
at org.codehaus.groovy.control.CompilationUnit.
doPhaseOperation
(CompilationUnit.java:542)
at org.codehaus.groovy.control.CompilationUnit.
processPhaseOperations
(CompilationUnit.java:520)
at org.codehaus.groovy.control.CompilationUnit.
compile(CompilationUnit.java:497)
at groovy.lang.GroovyClassLoader.doParseClass
(GroovyClassLoader.java:306)
at groovy.lang.GroovyClassLoader.parseClass
(GroovyClassLoader.java:287)
at groovy.util.GroovyScriptEngine$ScriptClassLoader.
parseClass
(GroovyScriptEngine.java:197)
at groovy.lang.GroovyClassLoader.parseClass
(GroovyClassLoader.java:267)
at groovy.lang.GroovyClassLoader.parseClass
(GroovyClassLoader.java:214)
at groovy.util.GroovyScriptEngine.loadScriptByName
(GroovyScriptEngine.java:470)
at groovy.util.GroovyScriptEngine.createScript
(GroovyScriptEngine.java:539)
at groovy.util.GroovyScriptEngine.run
(GroovyScriptEngine.java:526)
at com.cisco.ccbu.ccp.filter.ScriptFilter.
executeFilterOnSocialContact
(ScriptFilter.java:148)
at com.cisco.ccbu.ccp.filter.FilterManager.
executeGenericFilter
(FilterManager.java:688)
at com.cisco.ccbu.ccp.filter.FilterManager.
applyFilterOnSocialContact
(FilterManager.java:497)
at com.cisco.ccbu.ccp.filter.FilterManager.
applyFilterOnSocialContact
(FilterManager.java:413)
at com.cisco.ccbu.ccp.filter.FilterSubsystem.
executeFilterOnSocialContact
(FilterSubsystem.java:356)
at com.cisco.ccbu.ccp.filter.FilterSubsystem.
handleFilterSocialContactRequest
(FilterSubsystem.java:334)
at com.cisco.ccbu.ccp.filter.FilterSubsystem.
handleMessage
(FilterSubsystem.java:130)
at com.cisco.ccbu.ccp.filter.messaging.FilterMsgHandler.
handleMessage
(FilterMsgHandler.java:22)
at com.cisco.ccbu.infra.msg.BaseMessage$Handler.
handleMessageInternal
(BaseMessage.java:1197)
at com.cisco.ccbu.infra.msg.BaseMessage$Handler.
handleMessage
(BaseMessage.java:1175)
at com.cisco.ccbu.infra.msg.MSGHolder.handleImpl
(MSGHolder.java:322)
at com.cisco.ccbu.infra.msg.MSGDispatcher$Hook.handle
(MSGDispatcher.java:2976)
```

```
at com.cisco.ccbu.infra.msg.MSGDispatcher$DispatchRunnable.
handleMessage
(MSGDispatcher.java:3232)
at com.cisco.ccbu.infra.msg.MSGDispatcher$DispatchRunnable.run
(MSGDispatcher.java:3262)
at com.cisco.ccbu.infra.threads.InstrumentedRunnable.run
(InstrumentedRunnable.java:88)
at java.util.concurrent.ThreadPoolExecutor$Worker.runTask
(ThreadPoolExecutor.java:886)
at java.util.concurrent.ThreadPoolExecutor$Worker.run
(ThreadPoolExecutor.java:908)
at java.lang.Thread.run(Thread.java:619)
at com.cisco.ccbu.infra.threads.ThreadPoolThread.run
(ThreadPoolThread.java:164)
Caused by: java.lang.SecurityException: Importing
[java.lang.NullPointerException] is not allowed
at org.codehaus.groovy.control.customizers.SecureASTCustomizer.
assertImportIsAllowed
(SecureASTCustomizer.java:574)
at org.codehaus.groovy.control.customizers.SecureASTCustomizer.
access$800
(SecureASTCustomizer.java:121)
at org.codehaus.groovy.control.customizers.SecureASTCustomizer
$SecuringCodeVisitor.assertExpressionAuthorized
(SecureASTCustomizer.java:664) ... 38 more 1 error
</logBuffer>

<refURL>
 http://[ServerIP]:[Port]/ccp-webapp/ccp/
 filter/103105/results</refURL>
<result>100</result>
<socialContact>
 http://[ServerIP]:[Port]/ccp-webapp/ccp/socialcontact/
 AB1C3514100001320000F450A568DD2
</socialContact>
</FilterResult>
```

Script filter security

SocialMiner imposes restrictions on Groovy script code to ensure the security and integrity of the system and data.

For example, scripts are restricted from:

- Shutting down the system (system.exit()).
- Calling native Java methods (so as not to corrupt memory).
- Executing for longer than 30 seconds.
- Accessing the SocialMiner file system.
- Executing certain SQL commands.

A script that violates these restrictions will upload but will have no impact on the social contacts in the campaign to which it is applied.

In addition to Script binding, you can create objects from these classes:

- java.lang.Object
- java.lang.Boolean

- java.lang.Integer
- java.lang.Float
- java.lang.Short
- java.lang.Long
- java.lang.Double
- java.util.Date
- java.util.List
- java.util.Map
- java.util.Set
- java.util.Collections
- java.lang.String
- java.lang.StringBuilder
- java.util.TreeSet
- java.util.Vector
- java.util.LinkedHashSet
- java.util.LinkedList
- java.util.Stack
- java.util.ArraySet
- java.util.Arrays
- java.util.HashMap
- java.util.SortedMap
- java.util.TreeMap
- java.util.LinkedHashMap
- org.apache.commons.lang.StringUtils
- org.apache.commons.lang.Validate
- groovyx.net.http.HttpResponseDecorator
- java.util.Random
- java.math.*

Sample script filters

You can create script filters to change or add to the content of social contacts and to call external web services. For example, you can create a script to translate text to another language, to analyze sentiment (opendover), or to recognize trends (Google Prediction).

As of release 8.5(5), SocialMiner script filters use the GroovyScriptEngine. SocialMiner runs scripts with Groovy 1.8.

Script filter for social contact modification

```
/*
* Example script that modifies a social contact
*
* This script will demonstrate the modification of a social contact.
* For the full list of script filter fields, see ->
http://cvp/display/ccpdev/Filter+Script+API
*/
//Set the author
socialContact.author = "John Doe"

//Set the title
socialContact.title = "New Title"

//Set the description
socialContact.description = "This is a socialContact"

//Set the categories. Takes a list of strings.
socialContact.categories = ["category_1", "category_2", "category_3"]

//Set the tags. Alternatively, you can use the Java syntax as well.
//NOTE: duplicate tags will be removed when the socialContact is saved.
//NOTE: setting tags like so will replace any existing tags.
// to append tags see below.
socialContact.tags = ["tag1", "tag1", "tag2"]

//Append new_tag to tags
def tags = socialContact.tags;
tags += "tag3"
socialContact.tags = tags;

//A shorter way
socialContact.tags.add("tag4")

//Alternatively,
socialContact.tags += "tag5"

//Or even
socialContact.tags += ["tag6", "tag7"]

//Log the author, title, description, categories, and tags using getter methods.
log "Author is " + socialContact.author //should be "John Doe"
log "Title is" + socialContact.title //should be "New Title"
log "Description is " + socialContact.description //should be "This is a socialContact"
log "Categories are" + socialContact.categories //should
be["category_1","category_2","category_3"]
log "Tags are" + socialContact.tags //should be [ "tag1", "tag1", "tag2", "tag3", "tag4",
"tag5",
"tag6", "tag7"]

//however, keep in mind that duplicates will be removed when saved

//Finally, return
```

Script filter for Klout score generation

This is an example of a script filter that generates a Klout score for social contacts that are gathered from Twitter Account feeds:

```
def KLOUT_KEY = <klout_key>;

restClient.setProxy("161.44.248.59", 80, null);
```

```
String user = socialContact.getAuthor();
int space = user.indexOf(' ');
if (space != -1)
user = user.substring(0, space);

log "user = " + user;

def resp = restClient.get( uri: "http://api.klout.com/1/klout.xml", query :
[key: KLOUT_KEY, users: user]);

if (!resp.isSuccess()) {
    log ("Request Failed");
}
else {
    log "Influence: " + resp.data.user.kscore;
    boolean influential = (Double.parseDouble(resp.data.user.kscore as String) > 20.0);
    log "User is influential" + influential;
    if (influential){
       def tags = socialContact.tags;
       tags += "influential" as String;
       log "New tags: " + tags;
       socialContact.tags = tags;
    }
    return influential?100:0;
}
```

CHAPTER 15

Filter results

The Filter Results API allows you to get the results of a specified filter for analysis of the text passed to it.

Filter results API commands

This section describes the supported command (GET) for the filter results API and the parameters for that command.

GET

Get results for the specified filter.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/filter/<Id variables>/results
HTTP method:	GET
URL parameters:	• **document**: the text that the configured filter analyzes. • **socialContact:** the refURL of the social contact that the configured filter analyzes. It is valid to provide either document or social contact, but invalid to provide both or neither.
Example XML response:	The filter results are returned as a single <FilterResult> element that contains five required child elements. **FilterResult**: the container for the result. • **document**: the text passed to the filter for analysis. • **logBuffer**: the aggregation of the social contact filter log output. • **refURL**: the URL of the filter results request. • **result**: the result of the filter analysis expressed as an integer from 1–100.

- **socialContact**: the refURL of the social contact that was passed to the filter for analysis.

```
<FilterResult>
  <refURL>
     http://[ServerIP]:[Port]/ccp-webapp/ccp/
     filter/[id]/results
  </refURL>
  <result>100</result>
  <document>
   The text that was passed to the filter for analysis.
  </document>
  <socialContact>
     The refURL of the social contact that was passed to the
     filter for analysis.
  </socialContact>
  <logBuffer>
   The social contact filter log output.
  </logBuffer>
</FilterResult>
```

CHAPTER 16

Notification Rule

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

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

This API is represented on the SocialMiner user interface in the Notifications panel.

There are four types of notification rules: email, IM, http and (connection to) CCE. The parameters to use when creating or updating a notification rule depend on the type. The following table lists the parameters that are applicable to each type.

Rule type	Required parameters	Optional parameters	Notes
email	name, campaignUrl, tags, type, targets	description, subject, body	
im	name, campaignUrl, tags, type, targets	description, body	
http	name, campaignUrl, tags, type, httpUrl	description, httpUsername, httpPassword	
cce	name, campaignUrl, tags, type, scriptSelector, mediaRoutingDomainId	description	The scriptSelector is the Dialed Number String or Script Selector from the CCE configuration. The mediaRoutingDomainId is the ID of the selected Media Routing Domain from the CCE configuration. Currently, the only valid value for mediaRoutingDomainId is "1" (for voice callback).

Note Fields not relevant to a given notification type will be ignored. For example, a body specified in a http notification rule will be ignored.

Note You must configure an Email (SMTP) Server before notification can be sent through email. You must configure an XMPP server before IM notifications can be sent. Connection to CCE Notifications send a request to CCE with media routing information.

Notification API commands

This section describes the supported commands for the Notification API and the parameters for those commands.

POST

Creates a notification rule.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/notificationrule
HTTP method:	POST

Example request payload (email):	`<NotificationRule>`
	` <name>test</name>`
	` <description>`
	`  this is the description`
	` </description>`
	` <campaignUrl>`
	`  http://[ServerIP]:[Port]/ccp-webapp/ccp/`
	`  campaign/MyTestCampaign`
	` </campaignUrl>`
	` <tags>`
	`  <tag>test</tag>`
	`  <tag>cisco</tag>`
	` </tags>`
	` <targets>`
	`  <target>test@cisco.com</target>`
	`  <target>cisco@cisco.com</target>`
	` </targets>`
	` <type>email</type>`
	` <subject>Notification Rules</subject>`
	` <body>Click on this link.</body>`
	`</NotificationRule>`

Example request	`<NotificationRule>`
	` <name>test</name>`

payload (http):	
	```
	 <description>
	  this is the description
	 </description>
	 <campaignUrl>
	   http://[ServerIP]:[Port]/ccp-webapp/ccp/
	   campaign/MyTestCampaign
	 </campaignUrl>
	 <tags>
	  <tag>test</tag>
	  <tag>cisco</tag>
	 </tags>
	 <type>http</type>
	 <httpUrl>
	   http://someserver/notification/handler
	 </httpUrl>
	 <httpUsername>username</httpUsername>
	 <httpPassword>password</httpPassword>
	 <sslVerifyCertificates
	   false
	 </sslVerifyCertificates>
	</NotificationRule>
	```

Example request payload (CCE):	
	```
	<NotificationRule>
	  <name>test</name>
	  <description>this is the description</description>
	  <campaignUrl>http://[ServerIP]:[Port]/ccp-webapp/ccp/campaign/MyTestCampaign

	  </campaignUrl>
	  <tags>
	     <tag>test</tag>
	     <tag>cisco</tag>
	  </tags>
	  <type>cce</type>
	  <scriptSelector>allSalesAndService</scriptSelector>
	  <mediaRoutingDomains>
	     <mediaRoutingDomain>
	        <mediaRoutingDomainId>1</mediaRoutingDomainId>
	     </mediaRoutingDomain>
	  </mediaRoutingDomains>
	</NotificationRule>
	```

Parameters:	See Notification API parameters.
HTTP response headers:	A *201 Created* http header is returned on success, along with the REST URL to the new notification rule.

PUT

Updates an existing notification rule.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/notificationrule/<Id variables>
HTTP method:	PUT
Parameters:	See Notification API parameters. All parameters are optional for the Notification API update operation.

Example XML response:	`<NotificationRule>`
	` <body>New Contact:</body>`
	` <campaignUrl>`
	`  http://[ServerIP]:[Port]/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>`

HTTP response headers:	A *200 OK* http header is returned on success.

DELETE

Deletes a notification rule.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/notificationrule/<Id variables>
HTTP method:	DELETE
HTTP response headers:	A *200 OK* http header is returned on success.

GET (list)

Lists all notification rules.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/notificationrule
HTTP method:	GET

Example XML response:	
	```
	<NotificationRules>
	 <NotificationRule>
	  <body>New Contact:</body>
	  <campaignUrl>
	   http://[ServerIP]:[Port]/ccpwebapp/ccp/campaign/
	   Pushed_Contacts
	  </campaignUrl>
	   <changeStamp>0</changeStamp>
	   <name>Push</name>
	   <refURL>
	    http://[ServerIP]:[Port]/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.

GET

Retrieves a specific notification rule.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/notificationrule/<Id variables>
HTTP method:	GET

Example XML response:	
	```
	<NotificationRule>
	 <body>New Contact:</body>
	 <campaignUrl>http://[ServerIP]:[Port]/
	  ccpwebapp/ccp/campaign/Pushed_Contacts</campaignUrl>
	 <changeStamp>0</changeStamp>
	 <name>Push</name>
	 <refURL>http://[ServerIP]:[Port]/
	  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.

Notification API Parameters

The parameters to use when creating or updating a notification rule depend on the notification type. The following table lists the parameters that are applicable to each type.

Parameters are optional unless otherwise noted.

Parameter	Description	Notes
changeStamp	The change stamp of the notification record.	Integer. Defaults to 0. changeStamp is returned in GET. changeStamp, on page 3 is required for PUT (Update API).
name	The name of the notification rule.	Required for POST.
campaignUrl	The URL of the campaign.	String. Required for POST.
description	Description.	String. Follows SocialMiner standard description naming conventions.
tags/tag	Tag or list of tags from which the rule is activated and a notification is sent.	String. Required for POST. Maximum of 5.
type	The type of notification to send out for this rule.	Valid values are: • *email* (Notification is sent over email.) • *IM* (Notification is sent over IM.) • *http* (Notification is used for chat and is reserved for other developer applications that deliver social contact data to an external application.) • *(connection to) CCE* (Notification sends a request to CCE with media routing information.) Required for POST.

Parameter	Description	Notes
targets/target	One or more targets to which the notification rule is sent.	String. Required for POST. Maximum of 10.
httpUrl	The URL of a REST API.	Required for POST. SocialMiner will post details of a social contact to this URL.
subject	The subject of a notification rule message.	String. Maximum of 255 characters.
body	The body of the notification rule message. The link to the social contact is automatically inserted after the body. Link URL will appear beneath the body.	String. Maximum of 2048 characters. The body may contain reserved keywords in a special syntax. These Notification keywords in Email and IM will be replaced with values from the social contact. The reserved word syntax takes the form ${KEYWORD}.
httpUsername	Username	Required if authentication is necessary to use the REST API specified by **httpUrl**.
httpPassword	Password	Required if authentication is necessary to use the REST API specified by **httpUrl**.
sslVerifyCertificates	Defines if SSL certificate verification will be enabled or disabled for connections made for notification.	Boolean. Default is True. HTTP only.
scriptSelector	The Dialed Number String/Script Selector from the CCE configuration.	
mediaRoutingDomains	A list of media routing domains, each of which contains the media routing domain Id of the media routing domain from the CCE Configuration.	Integer, not null. At present, only one Media Routing Domain is supported.

Notification keywords in Email and IM

The body of an email or IM may contain reserved words in a special syntax of the form ${KEYWORD}. These key words will be replaced with values from the social contact. Keywords are listed here in uppercase, but they are case-insensitive.

The currently defined keywords are:

- SC_AUTHOR—this keyword is replaced with the social contact author.
- SC_CREATED_DATE—this keyword is replaced with the social contact created date.
- SC_DESCRIPTION—this keyword is replaced with the social contact description.
- SC_PUBLISHED_DATE—this keyword is replaced with the social contact published date.
- SC_SCREEN_URL—this keyword is replaced with the URL (a live link).
- SC_SOURCE_TYPE—this keyword is replaced with the social contact source type, for example: RSS, Twitter stream, Twitter account, or Facebook.
- SC_TAGS—this keyword is replaced with the social contact tags.
- SC_TITLE—this keyword is replaced with the social contact title.
- SC_TAGS—this keyword is replaced with the social contact tags.
- SC_EXTENSION_FIELD.<fieldname>—An extension field is additional data for the social contact. You can add a maximum of 100 extension fields, up to one megabyte of information. Like the keyword, the fieldname extension must also be enter in all upper case letters.
- SC_EXTENSION_FIELDS—Returns all extension fields for the social contact. They appear in the body in alphabetical order by name in the format *Name: Value*. If no value was defined, you see *Name: (—)*. If there are no extension fields, the variable is removed from the message body.

For example, if the Body is set to:

- The author of this Social Contact is: **${SC_AUTHOR}**.
- The message contents are as follows: **${SC_DESCRIPTION}**.

Then the notification message body would contain:

```
The author of this Social Contact is: someAuthorNameIfItWasProvided.
The message contents are as follows: theContentsOfTheSocialContactMessage.
```

If the body is set empty (null or blank), then the notification message will contain the SC_SCREEN_URL value by default.

Email messages are sent in HTML format and the body contents in the notification rule may contain user entered HTML markup.

IM messages are in text format, not HTML.

HTTP notifications

HTTP notifications will post a message with the following body to the URL specified in the notification rule httpUrl parameter.

HTTP method:	POST
Fields:	• **author**: author of the social contact. • **description**: body of the social contact. • **id**: datastore id of the social contact. • **link**: unique id of the original social contact (RSS ID or Twitter Id or Facebook Id, etc). • **notificationTag**: the tag that fired off the notification rule. • **publishedDate**: the publish date of the document. • **refURL**: the REST id of the social contact. Applications can do a GET on this URL to get social contact detail. • **screenPopUrl**: URL used to access the social contact in the SocialMiner's UI. • **status**: string (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. ◦ queued: The callback request was successfully submitted to the contact center for routing. • **statusTimestamp**: timestamp of the last status update. • **statusUserId**: id of the agent who updated the status (initially this is empty). • **tags/tag**: one or more tags associated with the social contact—normally this is optional but for CCX integration this should contain the routing tag. • **extensionFields/extensionField**: A collection of custom name and value pairs. The person submitting the social contact may specify up to 100 pairs and the entire collection can contain up to one megabyte of information. • **title**: title of the social contact. • **replyTemplateRefURL:** link to the reply template configuration. Present if reply template is configured for the contact.

	• **replyTemplateURL:** link to the reply template. Present if reply template is configured for the contact. • **replyType:** expected reply format (currently web, chat, or email). The URI below will be in a format compatible with the reply type. • **URI:** for chat room.

Example http motification message:

```
<SocialContact>
<author>David Dahlquist</author>
<description>New app and video sharing service, Thwapr,
  helps overcome the iPhone’s video sharing
  limitations letting you easily capture and share
  videos and photos with many types of mobile
  devices.</description>
 <id>073D0E871000012E0000ED8B0A568DDF</id>
 <link>
  http://rss.macworld.com/click.phdo?
  i=b942ba3fe59d99b27b08996ad8a9f06a
 </link>
 <notificationTag>ccx:sales</notificationTag>
 <publishedDate>1297201140000</publishedDate>
 <refURL>
  https://[ServerIP]:[Port]/ccp-webapp/
  ccp/socialcontact/
  073D0E871000012E0000ED8B0A568DDF
 </refURL>
 <screenPopUrl>
 http://[ServerIP]:[Port]/results.jsp?
 scID=461E5C541000012F000027650A568DF5&amp;
 campaignID=httpNotificationCampaign-01465-
 0000000000057</screenPopUrl>
 <status>unread</status>
 <statusTimestamp>1302551491320</statusTimestamp>
 <statusUserId>admin</statusUserId>
 <tags>
  <tag>ccx:sales</tag>
  <tag>ccx:support</tag>
 </tags>
 <extensionFields>
  <extensionField>
   <name>accountNumber</name>
   <value>6722392</value>
 </extensionField>
 <extensionField>
  <name>remarks</name>
  <value>My CRS-3 is not cooling enough</value>
 </extensionField>
 </extensionFields>
 <title>Default Entry Title -01465-0000000000032</title>
</SocialContact>
```

HTTP response headers:	A *200 OK* http header is returned on success.

CHAPTER 17

Proxy

The proxy API allows you to update and read proxy server settings. The current system supports a single proxy. If the proxy is enabled, **all feeds** use the proxy.

This API is represented on the SocialMiner user interface in the System Administration panel.

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

Proxy API commands

This section describes the supported commands for the Proxy API and the parameters for those commands.

Related Topics

GET, on page 107
PUT, on page 108
Proxy API parameters, on page 108

GET

Retrieves the proxy configuration.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/proxy/default
HTTP method:	GET

Example XML response:	`<Proxy>`
	`  <host>[ServerIP]</host>`
	`  <port>[Port]</port>`
	`  <exclusions>`
	`    <exclusion>localhost</exclusion>`
	`    <exclusion>*.cisco.com</exclusion>`
	`  </exclusions>`
	`  <enabled>true</enabled>`
	`  <refURL>`
	`   http://[ServerIP]:[Port]/ccp-webapp/ccp/`
	`   proxy/default`
	`  </refURL>`
	`</Proxy>`

PUT

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

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/proxy/default
HTTP method:	PUT
Parameters:	See Proxy API parameters, on page 108.

Example XML payload:	`<Proxy>`
	`  <host>[ServerIP]</host>`
	`  <port>[port]</port>`
	`  <exclusions>`
	`    <exclusion>localhost</exclusion>`
	`    <exclusion>*.cisco.com</exclusion>`
	`    <exclusion>161.44.*</exclusion>`
	`    <exclusion>192.168.1.1</exclusion>`
	`  </exclusions>`
	`  <enabled>true</enabled>`
	`</Proxy>`

Proxy API parameters

Parameters are optional unless otherwise noted.

Parameter	Description	Notes
enabled	True or false. Defines whether or not proxy use is enabled.	Defaults to false.
host	The fully-qualified 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.

Parameter	Description	Notes
exclusions	A list of hostnames to exclude from being used by the proxy.	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. Wildcards can be used. Examples: • localhost • *.cisco.com • xxx.yy.*

CHAPTER 18

Public URL Prefix for Chat Invitation

This API specifies (sets and retrieves) a publicly accessed SocialMiner server URL as a pre-defined property named "publicPrefix", which by default is not set. We currently support only one public prefix configuration.

GET

Get the public prefix settings. If the property is not set, it will return no prefix.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/publicprefix/default
HTTP method:	GET

Example XML response:	`<Publicprefix>`
	`  <path>`
	`    http[s]://[public_server]:[serverport]`
	`    [/optional/path]`
	`  </path>`
	`  <refURL>`
	`    http://[ServerIP]:[Port]/ccp-webapp/ccp/`
	`    publicprefix/default`
	`  </refURL>`
	`</Publicprefix>`
	See also API conventions, on page 1.

PUT

Updates the proxy configuration with the public prefix.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/publicprefix/default
HTTP method:	PUT

Parameters:	**prefix**(optional)—specifies the public URL for the SocialMiner server and port. If no port number is provided, it defaults to 8000. If an empty prefix is provided (<Prefix><<prefix> </prefix></Prefix>), then the prefix will be set to nothing. The value for "public server" cannot be an IP address.

Example XML response:	`<Publicprefix>`
	`  <path>`
	`    http[s]://[public_server]:[serverport]`
	`    [/optional/path]</path>`
	`</Publicprefix>`
	See also API conventions, on page 1.

CHAPTER 19

Purge

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.

This API is represented on the SocialMiner user interface in the System Administration panel.

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

Purge API commands

This section describes the supported commands for the Purge API and the parameters for those commands.

Related Topics

GET (list), on page 113
PUT, on page 114
Purge API parameters, on page 114

GET (list)

Lists the current purge settings.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/purge
HTTP method:	GET
Parameters:	See Purge API parameters, on page 114.

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

PUT

Updates the purge settings.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/purge
HTTP method:	PUT
Parameters:	See Purge API parameters, on page 114.

Example XML request payload:	`<PurgeConfig>`
	` <dataStoreEmergencyPurgeDiskUsage>`
	`   80`
	` </dataStoreEmergencyPurgeDiskUsage>`
	` <dataStorePurgeAge>10</dataStorePurgeAge>`
	` <reportingPurgeAge>550</reportingPurgeAge>`
	` <reportingPurgeTime>01:00</reportingPurgeTime>`
	`</PurgeConfig>`

HTTP response headers:	A *200 OK* http header is returned on success.

Purge API parameters

All parameters are optional.

Parameter	Description	Notes
dataStoreEmergencyPurgeDiskUsage	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.
dataStorePurgeAge	The age of the social contacts that will be purged.	Must be a whole number greater than 0 (no decimal).

Parameter	Description	Notes
reportingPurgeAge	The time, in 24 hour format (HH:mm), when the purge is to start.	Valid values are 00:00 to 23:59.
reportingPurgeTime	The ages of reporting records that will be purged.	Valid values are 1 to 550.

CHAPTER 20

Push feed

A Push feed (feed type = 7) pushes a new social contact into SocialMiner.

There are two ways to push a social contact into the system:

- using the social contact create (POST) method.
- using a get method.

Both API calls send a message to the Feed subsystem with the new social contact information and the push feed id.

The “pushed” social contact is handled like all other social contacts.

Push feed API commands

This section describes the supported commands for the Push feed API and the parameters for those commands.

Related Topics

POST, on page 117
GET, on page 117

POST

The social contact create (POST) method (POST, on page 145) is the preferred method for creating a social contact. Use the payload shown in that API.

GET

The GET method for push feeds is a simplified alternative to the social contact create (POST) API. It requires no API authentication.

WARNING: the GET method is not the preferred API call for creating the social contact. This is because:

- GET does not offer as much information protection as the social contact create (POST) method.
- GET sends no payload body, but contains all parameters for the social contact in the URL. Both the network and the browser could truncate information if the URL is very long. (This would happen if many extensionFields were provided.)

An example of a push feed URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/pushfeed/<id>?title= Example&author=admin&description =some_example&tags= ccx:sales,ccx: engineering&extensionField_customerID =98765&extensionField_remarks=My_router_is_broken
HTTP method:	GET
Parameters:	• **Title—**the title of the social contact. • **Author—**the author of the social contact. • **Description—**the content of the social contact. • **Tags—**a comma separated list of tags for this social contact. • **ExtensionField_<name>—**a custom name and value pair for this social contact. Tags and the extension field names cannot contain commas or colons.
HTTP response headers:	A 200 response indicates success. See also HTTP responses.

CHAPTER 21

Reply template

The Reply template API allows you to add, edit, and delete the name and location of custom reply templates.

This API is represented on the SocialMiner user interface in the Templates panel.

Reply template API commands

This section describes the supported commands for the Reply template API and the parameters for those commands.

Related Topics

POST, on page 119
DELETE, on page 120
GET (list), on page 120
GET, on page 121
PUT, on page 121
Reply template API parameters, on page 122

POST

Creates a new template definition.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/campaign/template/reply
HTTP method:	POST
Parameters:	See Reply template API parameters, on page 122.

Example XML request payload:	`<Template>` `<name>My  Template</name>` `<templateURL>` `http://this.is.my.template.url/template.html` `</templateURL>` `</Template>`
Response:	Status: 201 Created

DELETE

Deletes a custom reply template definition.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/campaign/template/reply/<Id variables>
HTTP method:	DELETE
HTTP response headers:	Status 200: OK

GET (list)

Lists all custom reply templates stored on this system.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/campaign/template/reply/
HTTP method:	GET
Parameters:	See Reply template API parameters, on page 122.

Example response:	
	```
	<Templates>
	 <Template>
	  <changeStamp>0</changeStamp>
	  <name>Cisco Twitter</name>
	  <refURL>
	   http://[ServerIP]:[Port]/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://[ServerIP]:[Port]/ccp-webapp/ccp/
	   template/reply/100024
	  </refURL>
	  <systemDefined>false</systemDefined>
	  <templateURL>
	    http://this.is.my.template.url/template.html
	  </templateURL>
	 </Template>
	</Templates>
	```

GET

Gets the details for one reply template.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/campaign/template/ reply/<Id variables>
HTTP method:	GET

Example XML response:	
	```
	<Template>
	 <changeStamp>1</changeStamp>
	 <name>Remote Custom Template</name>
	 <refURL>
	  http://[ServerIP]:[Port]/
	  ccp-webapp/ccp/template/reply/105673
	 </refURL>
	 <systemDefined>false</systemDefined>
	 <templateURL>
	  http://[ServerIP]:[Port]/ccp-reply/
	  twitter-reply-gadget.jsp
	 </templateURL>
	</Template>
	```

Parameters:	See Reply template API parameters, on page 122.

PUT

Updates an existing reply template definition.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/campaign/template/

	reply/<Id variables>
HTTP method:	PUT
Parameters:	See Reply template API parameters, on page 122.

Example XML request payload:	`<Template>`
	` <changeStamp>1</changeStamp>`
	` <name>Remote Custom Template</name>`
	` <templateURL>`
	`  http://[ServerIP]:[Port]/ccp-reply/`
	`  twitter-reply-gadget.jsp`
	` </templateURL>`
	`</Template>`

Reply template API parameters

Parameters are optional unless otherwise noted.

Parameter	Description	Notes
changeStamp	The change stamp of the reply template record.	Integer. Defaults to 0. changeStamp is required for PUT (update API). The changeStamp increments by 1 if the update is successful.
name	The template name.	String. Required for POST.
refURL	The reference to the reply template.	Boolean.
systemDefined	True if the template was pre-installed on SocialMiner.	systemDefined templates cannot be deleted.
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://docs.opensocial.org/display/OS/Home.	String. Required for POST.

CHAPTER 22

Reporting server

The reporting server API returns the reporting Server database connection information.

This API is represented on the SocialMiner user interface in the System Administration panel.

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

Reporting server API commands

This section describes the supported command (GET) for the reporting server API and the parameters for that command.

GET

Gets the reporting server database connection information.

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

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/reportingserver
HTTP method:	get

Example XML response:	`<ReportingServer>`
	` <databaseType>Informix</databaseType>`
	` <reportingDatabase>[database name]</reportingDatabase>`
	` <reportingHost>[hostname]</reportingHost>`
	` <reportingHostIp>[ServerIP]</reportingHostIp>`
	` <reportingPort>[Port]</reportingPort>`
	` <reportingServer>[server name]</reportingServer>`
	`</ReportingServer>`

Parameters	See Reporting server API parameters, on page 124.

Reporting server API parameters

All parameters are optional.

Parameter	Description	Notes
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.	

CHAPTER 23

Reporting user

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 is not editable.

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

Reporting user API commands

This section describes the supported commands for the reporting user API and the parameters for those commands.

Related Topics

POST, on page 125
DELETE, on page 126
GET (list), on page 126
GET, on page 126
PUT, on page 127
Reporting user API parameters, on page 127

POST

Creates the reporting user.

Note The username of the reporting user must be *reportinguser* or an error is received.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/reportinguser/

HTTP method:	POST
Parameters:	See Reporting user API parameters, on page 127.

Example XML request payload:	`<ReportingUser>`
	` <username>reportinguser</username>`
	` <password>password</password>`
	`</ReportingUser>`

HTTP response headers:	A *201 Created* http header is returned on success.

DELETE

Deletes the reporting user.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/reportinguser/<Id variables>
HTTP method:	DELETE
HTTP response headers:	A *200 OK* http header is returned on success.

GET (list)

Lists all reporting users.

Note Currently only a single reporting user is supported.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/reportinguser
HTTP method:	GET
Parameters	See Reporting user API parameters, on page 127.

Example XML response:	`<ReportingUsers>`
	` <ReportingUser>`
	`  <refURL>http://[ServerIP]:[Port]/`
	`   ccp-webapp/ccp/reportinguser/100001</refURL>`
	`  <username>reportinguser</username>`
	` </ReportingUser>`
	`</ReportingUsers>`

GET

Gets a single reporting user from the database.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/reportinguser/<Id variables>

HTTP method:	GET

Example response:	`<ReportingUser>`
	` <refURL>http://[ServerIP]:[Port]/`
	`  ccp-webapp/ccp/reportinguser/100001</refURL>`
	` <username>reportinguser</username>`
	`</ReportingUser>`

PUT

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

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/reportinguser/<Id variables>
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>`

HTTP response headers:	A *200 OK* http header is returned on success.

Reporting user API parameters

Parameter	Description	Notes
username	Must be *reportinguser.*	String. Required for POST
password	The password for the reporting user.	String. Required for POST

CHAPTER 24

Serviceability

The serviceability API provides details on the SocialMiner version, service status, and various other information used primarily by SocialMiner developers for debugging issues.

Note Only the administrator can use this API.

This API is represented on the SocialMiner user interface in the System Administration panel.

Serviceability API commands

This section describes the supported commands for the serviceability API and the parameters for those commands.

Related Topics

GET, on page 129
GET (list), on page 130
Serviceability API parameters, on page 133

GET

Gets the value for a given serviceability attribute. Each piece of information is returned as an XML element named for each category of information.

Any combination of the categories listed below can be retrieved by providing any number of category parameters on the URL.

For example,

```
http://<ServerIP>:<Port>/ccp-webapp/ccp/serviceability?

category=diskUsage&category=serviceState
```

returns only the *diskUsage* and *serviceStates* categories.

Note With no categories specified, all categories except *systemInfo* are returned. This is because *systemInfo* is very expensive to run as it collects all JMX attributes from all four JVMs in the product (runtime server, cassandra, solr and Tomcat).

To retrieve all categories including systemInfo, use the special *all* category: *http://<ServerIP>:<Port>/ccp-webapp/ccp/serviceability?category=all*

.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/serviceability
HTTP method:	GET
Parameters:	See Serviceability API parameters, on page 133.

Example XML response:	Example response for http://<ServerIP>:<Port>/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>`

GET (list)

Lists all of the available serviceability attributes and their current values. See GET, on page 129 for details.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/serviceability/
HTTP method:	GET

Example XML response:	This example is an excerpt of the XML response. The full output consists of more than 3 MB of data.
	`<Serviceability>`
	` <diskUsage>`
	`  <activePartitionTotalBytes>`
	`   13463810048`
	`  </activePartitionTotalBytes>`
	`  <activePartitionUsableBytes>`
	`   2352652288`
	`  </activePartitionUsableBytes>`
	`  <commonDatastorePartitionTotalBytes>`
	`   46835601408`
	`  </commonDatastorePartitionTotalBytes>`
	`  <commonDatastorePartitionUsableBytes>`
	`   37941018624`
	`  </commonDatastorePartitionUsableBytes>`

```
  <inactivePartitionTotalBytes>
   13463781376
  </inactivePartitionTotalBytes>
  <inactivePartitionUsableBytes>
   2357100544
  </inactivePartitionUsableBytes>
 </diskUsage>
...
 <facebookAppInfo>
  <facebookAppId>
   facebookAppIdAsString
  </facebookAppId>
 </facebookAppInfo>
...
 <feedStatuses>
  <FeedStatus>
   <feedRefURL>
    http://[ServerIP]:[Port]/ccp-webapp/ccp/feed/100000
   </feedRefURL>
   <lastFetchCount>20</lastFetchCount>
   <statusDescription>
    NORMAL
   </statusDescription>
  </FeedStatus>
 </feedStatuses>
...
 <filterStatuses>
  <FilterStatus>
   <timeSinceLastExecution>
    125
   </timeSinceLastExecution>
   <name>TestFilter</name>
   <status>FILTER_EXECUTION_ERROR</name>
   <statusDescription>
    Cannot cast object 'null' with class 'null'
    to class 'int'.
    Try 'java.lang.Integer' instead.
   </statusDescription>
  </FilterStatus>
 </filterStatuses>
...
 <notifiers>
  <Notifier>
   <connectionStatus>
    STATE_IN_SERVICE
   </connectionStatus>
   <notificationsDropped>
    0
   </notificationsDropped>
   <notificationsFailed>
    0
   </notificationsFailed>
   <notificationsSent>
    10
   </notificationsSent>
   <outQueueDepth>2</outQueueDepth>
   <outQueueWait>1</outQueueWait>
   <type>im</type>
  </Notifier>
 </notifiers>
...
 <notificationRuleStatuses>
    <NotificationRuleStatus>
      <id>10001</id>
      <name>CCE</name>
      <type>Connection to CCE</type>
      <status>SUCCESS<status>
      <statusReason>
       NOTIFICATION_STATUS_NORMAL
      <statusReasonDescription>
       The notification status is normal.
      </statusReasonDescription>
      <success>10</success>
      <failure>5</failure>
      <total>15</total>
      <statusChangeTime>
       Sat Oct 30 18:38:41 EDT 2010
</statusChangeTime>
    </NotificationRuleStatus>
    <NotificationRuleStatus>
      <id>10002</id>
      <name>MyNotification</name>
      <type>http</type>
      <status>FAILURE<status>
      <statusReason>
       NOTIFICATION_STATUS_BAD_CONFIGURATION
      </statusReason>
      <statusReasonDescription>
       Failed to send notification due to
       Bad Configuration.
     </statusReasonDescription>
      <success>0</success>
      <failure>5</failure>
      <total>5</total>
      <statusChangeTime>
       Sat Oct 30 18:38:41 EDT 2010
      </statusChangeTime>
    </NotificationRuleStatus>
 </notificationRuleStatuses>
...
 <serviceStates>
  <datastoreServerState>
    SERVER_STATE_IN_SERVICE
  </datastoreServerState>
  <indexerServerState>
   SERVER_STATE_IN_SERVICE
  </indexerServerState>
  <runtimeServerState>
   SERVER_STATE_IN_SERVICE
  </runtimeServerState>
 </serviceStates>

 <systemPerformance>
  <socialContactsPerHour>
    10000
  </socialContactsPerHour>
 </systemPerformance>

 <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>
...
 <eventingInfo>
  <connectionStatus>
    CONNECTED
  </connectionStatus>
  <dsNfyMsgsRcvd>15</dsNfyMsgsRcvd>
  <outQueueDepth>0</outQueueDepth>
  <outQueueWait>0</outQueueWait>
```

```
   <xmppEventsDropped>0</xmppEventsDropped>
   <xmppEventsFailed>0</xmppEventsFailed>
   <xmppEventsSent>15</xmppEventsSent>
  </eventingInfo>
...
</Serviceability>
```

Serviceability API parameters

Parameters are optional unless otherwise noted.

Parameter	Description	Notes
apistats	Timing statistics on each API type exposed by the system.	All time units are in milliseconds. The interval for "lastInterval" stats is 30 seconds.
configuration	An XML representation of all configuration objects in the system.	This is the same XML that the API produces and consumes.
diskusage	Reports on the total size in bytes and remaining usable bytes for each partition on the disk.	In some small deployments, there will be no reportingDatabase partition because it is the same as the commonDatastore partition. In large deployments, the partitions reported for both total and usable size will be: • activePartition—main OS partition. Contains config database. Mounted on */*. • inactivePartition—inactive side used to upgrade. Will be activePartition after a switch. Mounted on */partB*. • commonDatastorePartition—social contact datastore. Both Cassandra and Solr data files. Mounted on */common*. • reportingDatabasePartition—contains the reporting database. Mounted on */spare*. On small deployments this is sometimes a symlink into /common. In those cases, this partition will be omitted from the results.

Parameter	Description	Notes
eventingInfo	Statistics about the Eventing Subsystem which publishes social contact state change events to XMPP clients using the embedded XMPP server.	• dsNfyMsgsRcvd: the number of social contact state change messages received from the data store. • xmppEventsSent: the number of xmpp events successfully sent by this notifier. • xmppEventFailed: the number of failed xmpp notification attempts. • xmppEventDropped: the number of xmpp events that were dropped because the output queue was full. • outQueueDepth: the number of items in the output queue. • outQueueWait: the average amount of time between when an xmpp event is queued and when it is sent. • connectionStatus: one of the following states: ◦ *CONNECTED*: successfully connected to the embedded XMPP 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.
facebookAppInfo	SocialMiner's Facebook Application ID.	

Parameter	Description	Notes
feedstatuses	Any number of FeedStatus elements containing the last known status of every feed in the system that is currently being fetched because at least one campaign has that feed configured in it. See feed statusDescription values, on page 142.	FeedStatus can include this information: • FETCH_SUCCESS_RATE: calculated based on the last 10 fetches: FetchSuccessRate = ((FetchSuccess / (FetchSuccess+FetchFailure))/100). Applies to these classes: ccp-feeds and ccp-webapp. • LAST_GOOD_FETCH_TIME: applies to classes ccp-feeds and ccp-webapp.
filterStatuses	FilterStatus elements containing the last known status of every filter in the system.	• name: filter name • status: NORMAL or FILTER_EXECUTION_ERROR (for script filter). As the runtime does not track whether a filter is in a campaign, that information is not available in filter status. • status description: a detailed error message in the case of a script filter execution error. Otherwise, normal. • timeSinceLastExecution: the time in seconds since the last execution of the script filter. For all other filter types and for a script filter that has never been executed since runtime started, returns -1.
hardware	The status of the hardware SMALL, LARGE, UNSUPPORTED.	diskOneSize: size of disk 1. diskTwoSize: size of disk 2. diskOneStar: start byte of disk 1. diskTwoStart: start byte of disk 2. memory: total physical memory. numberOfCpus: total number of cpus(cores). type: SMALL, LARGE, or UNSUPPORTED. Is determined by looking at the values listed above and comparing them to known ovf values.

Parameter	Description	Notes
jvmStats	Statistics about each running Java Virtual Machine (JVM) on the server. These are the same as the serviceStates category.	The stats for each JVM are: • connectionName: the name and JMX URL (only accessible from the local machine) of the JVM. • cpuPercentageSamples: samples of the percentage of CPU used by this JVM. Each sample is a 30 second interval. Five days of samples are kept. • heapSamples: samples of the total heap usage in MB. Each sample is a 2 minute interval. One week of samples are kept. • heapSlope: the slope of the graph of the heapSamples. After a long period of uptime this should be very close to 0 (flat slope) indicating no heap memory leakage over time. Over short periods the slope is not useful because of the volatility of the heap usage. • uptime: elapsed time since this JVM started in milliseconds. • uptimeString: a human readable version of uptime.

Parameter	Description	Notes
notifiers	The status of the notification services for this server.	**For "email", "im" and "http" notification types, associated data are** • type: the type of notification sent by this notifier. • notificationsSent: the number of notifications 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.

Parameter	Description	Notes
notifiers		**For "connection to cce" notification type, associated data are** • type: connection to cce. • notificationsSent: the number of tasks successfully allocated to an agent by cce. • notificationsFailed: the number of tasks failed to find an agent. • notificationsDropped: the number of tasks that were dropped because there was no connection to cce. • outQueueDepth: the number of tasks currently being processed by cce. • outQueueWait: 0.
		• **ConnectionStatus**: Can be one of ◦ *NOT_ESTABLISHED*: CCE configuration for multichannel routing is disabled. ◦ *LISTENING*: listening for incoming connection from CCE media routing PG. ◦ *ESTABLISHING*: establishing connection from CCE media routing PG. ◦ *ESTABLISHED_ ROUTING_ ENABLED*: established connection with CCE media routing PG.

Parameter	Description	Notes
notificationRule Statuses	Status and statistics for each notification rule.	• id: the id of the notification rule. • name: the name of the notification rule. • type: the type of notification rule. • status: an explanation of the status for the most recent notification sent. ◦ success: the number of notifications that were successfully sent. ◦ failure: the number of notifications that failed to be sent. ◦ total: The total number of notifications processed.

Parameter	Description	Notes
		• statusReason: Reason code for the current status. ◦ **NOTIFICATION_ STATUS_ UNEXPECTED_ ERROR** ◦ **NOTIFICATION_ STATUS_ NORMAL** ◦ **NOTIFICATION_ STATUS_BAD_ CONFIGURATION** ◦ **NOTIFICATION_ STATUS_ CONNECTION_ PROBLEM** ◦ **NOTIFICATION_ STATUS_ AUTHENTICATION_ FAILED** ◦ **NOTIFICATION_ STATUS_CCE_ ROUTING_PROBLEM** ◦ **NOTIFICATION_ STATUS_ NO_CONNECTION_ TO_CCE** ◦ **NOTIFICATION_ STATUS_ CCE_MESSAGE_ QUEUE_FULL** ◦ **NOTIFICATION_ STATUS_ RATE_LIMITED**

Parameter	Description	Notes
		• statusReasonDescription: an explanation for the current statusReason code. • statusChangeTime: time when the status was last changed.
serviceStates	Get the state of services.	Values for serverState can be: • SERVER_STATE_API_INIT: the Serviceability API is initializing. This is the 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.
sessionInfo	A list of current sessions (ip address:username combination.	This is used to indicate active sessions and calculate approximate number of logged in users.
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.
systemConditions	Get a list of SystemCondition elements describing persistent states reported by the services.	

Parameter	Description	Notes
systemPerformance	**socialContactsPerHour:** This value will be displayed in the Admin gadget along with an icon to show the user if they are in the safe (< 8,000), warning (between 8,000 and 10,000), or critical range (> 10,000).	The socialContactsPerHour comes from the ReportingIntervalStatsMbean. It retrieves the number of social contacts every 15 minutes and then to get the social contacts per hour it averages the last 4 intervals.
version	Application version information.	Shows the currently installed version of SocialMiner and the prior version of SocialMiner.

feed statusDescription values

Values for a feed's statusDescription can be:

- **AUTHENTICATION_FAILED**: an authenticated feed failed because of incorrect credentials.
- **DATASTORE_ERROR**: error attempting to write the social contacts to the datastore.
- **FACEBOOK_AUTHORIZATION_FAILED**: Authorization was denied when SocialMiner attempted to pull contacts from Facebook.
- **FACEBOOK_PARSE_ERROR**: The contacts that the feed pulled from Facebook could not be parsed. This may be due to an unknown change in the Facebook API.
- **NETWORK_ERROR**: unhandled network error.
- **NETWORK_NOT_REACHABLE**: could not connect to the remote server.
- **NETWORK_TIMEOUT**: the remote server was reachable but did not respond to the request in a timely manner.
- **NORMAL**: the feed is operating normally.
- **RATE_LIMIT_REACHED**: A Twitter account, search, or stream feed has exceeded the number of requests that Twitter allows it to make over a given period of time. The status indicates when Twitter will allow SocialMiner to begin pulling contacts again.
- **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.
- **TWITTER_STREAM_INTERRUPTED**: the thread that runs to the Twitter stream client was interrupted.
- **TWITTER_STREAM_CONNECTED**: the Twitter stream feed has connected to Twitter, but has not yet received any contacts.
- **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.
- **UNKNOWN_ERROR**: an error occurred that does not have a specific exception.
- **UNSUPPORTED_FEED_CONTENT**: the content retrieved from the feed is not in a format that SocialMiner supports.

CHAPTER 25

Social contact

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

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

Social Contact API commands

This section describes the supported commands for the Social Contact API and the parameters for those commands.

POST

Creates a social contact for a Push feed.

Before you do this, you must:

- Create a type 7 push feed and POST it.
- Confirm that the post returned a 201 (success) response code, then look in the location field of the http response for the reference URL (refURL) of the feed just created.
- Add the chat feed to a campaign. You can create a new campaign and then use the PUT API to add the feed to it or to any existing campaign.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/socialcontact/
HTTP method:	POST
Parameters:	See Social contact API parameters, on page 153.

Example XML request	
	```
	<SocialContact>
	 <feedRefURL>http://[ServerIP]:[Port]/
	  ccp-webapp/ccp/feed/(id)</feedRefURL>
	 <title>social contact title</title>
	 <publishedDate>social contact date of publish</publishedDate>
	 <author>Customer_Name</author>
	 <isInvited>true</isInvited>
	 <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>
	 <extensionFields>
	  <extensionField>
	   <name>accountNumber</name>
	   <value>6722392</value>
	  </extensionField>
	  <extensionField>
	   <name>remarks</name>
	   <value>My CRS-3 is not cooling enough</value>
	  </extensionField>
	 </extensionFields>
	 <isSoftLocked>false</isSoftLocked>
	</SocialContact>
	```
	 Note During creation of a social contact, the following fields are restricted (they can only be set by the system):
	• inviteStatus: is the status of chat invitations sent (if any) from this social contact. The default is NONE.
	• shortURLIds: is a list of short URL Ids which were generated for this social contact. This list is updated when a short URL is created for a social contact.
	
	You can set the QUEUED status from the API. However, currently it is only set internally by SocialMiner. SocialMiner sets the social contact status to QUEUED when a social contact is to be or has been routed to some external entity.

HTTP response:	A *201 Created* http header is returned on success along with a URL to the newly created social contact.

GET

Gets status and detail for the specified social contact.

The <objectId> attribute required for this command is found in campaign results, in the feed/entry/link rel="socialcontact" element. For example: <link rel="socialcontact" href="http://<ServerIP>:<Port>/ccp-webapp/ccp/socialcontact/22E00F5310000129460A1EB40A568DDE" />

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/socialcontact/<Id variables>
HTTP method:	GET

Example responses:

Results are returned as XML.

```
<SocialContact>
 <refURL>
  http://[ServerIP]:[Port]/ccp-webapp/
  ccp/socialcontact/
  F9B43F9B100001282E3C18380A568DDD
 </refURL>
<statusTimestamp>1276008213</statusTimestamp>
<status>Reserved</status>
<statusUserId>admin</statusUserId>
 <tags>
  <tag>tag1</tag>
  <tag>tag2</tag>
 </tags>
<extensionFields>
 <extensionField>
  <name>accountNumber</name>
  <value>13131313</value>
 </extensionField>
</extensionFields>
<sourceType>chat</sourceType>
<isInvited>false</isInvited>
<inviteStatus>sent</inviteStatus>
<shortUrlIds>
   <shortUrlId>
     6DCFBB7A10000139000076C00A568DDB
   </shortUrlId>
</shortUrlIds>
<transcriptRefURL>
  http://[ServerIP]:[Port]/ccpwebapp/ccp/socialcontact/
  F9B43F9B100001282E3C18380A568DDD/transcript
</transcriptRefURL>
<statusUserId>myname</statusUserId>
<replyTemplateRefURL>
  http://[ServerIP]:[Port]/ccp-webapp/ccp/
  template/reply/105678
</replyTemplateRefURL>
<replyTemplateURL>
  http://[ServerIP]:[Port]/gadgets/files/ccp/templates/
  reply/cisco_twitter.jsp
</replyTemplateURL>
<id>F9B43F9B100001282E3C18380A568DDD</id>
<isSoftLocked>false</isSoftLocked>
</SocialContact

<SocialContact>
 <author>AlmaDao1@twitter.com</author>
 <createdDate>1317829257152</createdDate>
 <description>
  Top 50 Companies With Best CSR Announced October 5 on
  &lt;em&gt;Boston&lt;/em&gt; College Webinar | 3BL Media: &lt;a
  href=&quot;http://t.co/VXbzViaj&quot;&gt;
  http://t.co/VXbzViaj&lt;/a&gt;
  via @&lt;a class=&quot; &quot;
  href=&quot;http://twitter.com/AddThis&quot;
  &gt;AddThis&lt;/a&gt;
 </description>
 <extensionFields/>
 <id>D4BEA7BC100001320003D80A0A568DF8</id>
 <link>
  http://twitter.com/AlmaDao1/statuses/121610791286878209
 </link>
 <publishedDate>1317829246000</publishedDate>
 <refURL>
  https://[ServerIP]:[Port]/ccpwebapp/ccp/socialcontact/
  D4BEA7BC100001320003D80A0A568DF8
 </refURL>
 <replyTemplateRefURL>
  https://[ServerIP]:[Port]/ccpwebapp/ccp/template/reply/
  103184
 </replyTemplateRefURL>
 <replyTemplateURL>http://test.com</replyTemplateURL>
 <replyToId></replyToId>
 <status>unread</status>
 <statusTimestamp>1317829257148</statusTimestamp>
 <statusUserId></statusUserId>
 <tags/>
 <title>
   Top 50 Companies With Best CSR
   Announced October 5 on Boston
   College Webinar | 3BL Media:
   http://t.co/VXbzViaj via @AddThis
 </title>
</SocialContact>
```

Note 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 cannot be changed by the social contact API.

You can set the QUEUED status from the API. However, currently it is only set internally by SocialMiner. SocialMiner sets the social contact status to QUEUED when a social contact is to be or has been routed to some external entity.

PUT

Updates the status or tags of an existing social contact.

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

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/socialcontact/<Id variables>
HTTP method:	PUT
Parameters:	See Social contact API parameters, on page 153.

Example XML request payload:	`<SocialContact>`
	` <statusTimestamp>1276008213</statusTimestamp>`
	` <status>Reserved</status>`
	` <statusUserId>admin</statusUserId>`
	` <tags>`
	`  <tag>cool</tag>`
	`  <tag>fresh</tag>`
	` </tags>`
	`  <replyTemplateRefURL>http://[ServerIP]:[Port]/ccp-webapp/ccp/`
	`  template/reply/105678</replyTemplateRefURL>`
	` <replyTemplateURL>http://[ServerIP]:[Port]/gadgets/files/ccp/`
	`  templates/reply/cisco_twitter.jsp</replyTemplateURL>`
	` <isSoftLocked>false</isSoftLocked>`
	`</SocialContact>`
	
	Note 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. The fields draftResponse and draftAction are saved only with a social contact update that transitions to state **draft**. If the social contact transitions to the handled state, then the draftResponse and draftAction are cleared.
	Updates to following fields are restricted (updates can only be made by the system):
	• isInvited: this field is set to true if the chat contact was created as a result of a chat invitation. This field is set by the system when the chat contact is created. It is set to false by default for all other social contacts.
	• shortURLIds: is a list of short URL Ids which were generated for this social contact. This list is updated when a short URL is created for a social contact.
	
	You can set the QUEUED status from the API. However, currently it is only set internally by SocialMiner. SocialMiner sets the social contact status to QUEUED when a social contact is to be or has been routed to some external entity.

Example XML response:	`<SocialContact>`
	` <refURL>http://[ServerIP]:[Port]/ccp-webapp/ccp/socialcontact/`
	`  22E00F5310000129460A1EB40A568DDE</refURL>`
	` <status>reserved</status>`
	` <statusTimestamp>1276190792688</statusTimestamp>`
	` <tags>`
	`  <tag>cool</tag>`
	`  <tag>fresh</tag>`
	` </tags>`
	` <statusUserId>admin</statusUserId>`
	`</SocialContact>`

GET (chat transcript)

Retrieves the transcript for a chat contact (is only valid for chat feeds).

URL:	http://<ServerIP>:<Port>/<webapp>/ccp/socialcontact/<Id variables>/transcript
HTTP method:	GET
Parameters:	See Social contact API parameters, on page 153.

Example XML response :	`<ChatTranscript>`
	`<endDate>1324587047731</endDate>`
	`<participants>`
	` <participant>Test_Author</participant>`
	` <participant>admin_ChatTest-07034-0000000000010</participant>`
	`</participants>`
	`<startDate>1324587046492</startDate>`
	`<transcript>`
	`<chat>`
	` <time>Thu Dec 22 15:50:47 EST 2011</time>`
	` <name>Test_Author</name>`
	` <msg>Hello world!!!!</msg>`
	`</chat>`
	
	`<chat>`
	` <time>Thu Dec 22 15:50:47 EST 2011</time>`
	` <name>admin_ChatTest-07034-0000000000010</name>`
	` <msg>Hello world!!!!</msg>`
	`</chat>`
	`</transcript>`
	`</ChatTranscript>`

GET (search)

Use GET (search) to search for social contacts. GET (search) is based on a Solr search. Wildcard-based searches using "?" for a single-character and "*" for multiple-characters are supported for the fields specified.

The following search fields are currently supported:

- *sc.author* is the author of the contact. In chat, this field contains the customer nickname. Wild card searches are supported for "?" and "*".
- *sc.title* is the title of the contact.
- *sc.tags* are the attributes that the contact is tagged with. Wild card searches are supported for "?" and "*".
- *sc.description* is the description of the contact contents.
- *chat.participantNickNames* are the nicknames of the parties in a chat room. Wild card searches are supported for "?" and "*".
- *sc.publishedDate* is the date the contact occurred. Searches can be narrowed down to a specific interval.
- *sc.socialContactStatus* is the SocialMiner status of the contact (for example, open or handled).
- *chat.transcript* is the content of a chat transcript.

The pairs of field names and values in a search query use the following syntax: *Solr_fileld_name:value*.

The search name and value pairs can be joined in logical expressions by AND or OR. The search terms should be encoded if they contain Solr special characters. (For more details, see http://wiki.apache.org/solr/SolrQuerySyntax.)

Note The following limitations apply to the search function:

- Field-based searches search only the specified field for the given term or terms (multiple terms are enclosed in double quotes).

  For example, to search for contacts authored by John Smith, you can search for sc.author:John*, sc.author:*Smith, or sc.author:"John Smith".

- For default searches of the following fields, just type the words in the search field to find the terms:

  sc.title
  sc.description
  chat.transcript

  For example, if there is a description "These are important chats from a special customer"; you can just type 'important chats' into the search box (no quotes required).

- You can do wildcard-based searches of these fields.

  sc.author
  sc.tags
  chat.agentName
  chat.agentNickname

  For example, to find social contacts with authors chatAuthorA, chatAuthorB, or chatAuthorC, you can search for chatAuthor*.

- There is a 32-character limit to the word length (it can only search for words up to 32-characters long).

URL:	http://ServerIP:8080/ccp-webapp/ccp/search/contacts
HTTP method:	GET
Output format:	ATOM
Parameters:	• *q* (required). Query parameter. • *count* (optional.) Defines how many results to return (default = 50, max = 200). • *startIndex* (optional). Identifies the index of the first search result. Because we are using Solr for querying, the first index should be 0 (default = 0).

Example query request:	`http://<ServerIP>:<Port>/ccp-webapp/ccp/search/contacts?q=sc.sourceType:chat%20AND%20sc.author:*%20AND%20chat.participant NickNames:*%20AND%20sc. publishedDate:[2012-01-05T23:59:59.999Z%20TO%202012-01-06T23:59:59.999Z]` As illustrated in the example above, the search request must be URL encoded in order to work (for example, spaces in the text are represented by "%20").
Example XML response:	see below

```
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:dc=
   "http://purl.org/dc/elements/1.1/"
   xmlns:ccp="http://www.cisco.com/ccbu/ccp/xml/socialcontact/1.0/"
   xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/">

  <title>Results of Search</title>
  <link rel="self" href="http://[ServerIP]:[Port]/ccp-webapp/ccp/
     search?q=sc.sourceType:chat AND
     sc.author:author*&count=50&startIndex=0"/>
  <subtitle>This feed has been created by Cisco SocialMiner</subtitle>
  <id>http://[ServerIP]:[Port]/ccp-webapp/ccp/search</id>
  <updated>2012-01-11T03:12:03Z</updated>
  <dc:date>2012-01-11T03:12:03Z</dc:date>
  <opensearch:itemsPerPage>3</opensearch:itemsPerPage>
  <opensearch:totalResults>3</opensearch:totalResults>
  <opensearch:startIndex>1</opensearch:startIndex>
  <opensearch:Query role="request" searchTerms="sc.sourceType:chat AND
   sc.author:author*"/>
  <opensearch:link rel="search" type="application/opensearchdescription+xml"

   href="http://www.cisco.com/opensearch-description.xml"/>

 <entry>
  <title>search&%Chat</title>
  <link rel="alternate" href="https://10.86.141.245/ccpwebapp/
    ccp/socialcontact/B5293B5310000134000039590A568DF5"/>
  <link rel="socialcontact" href="https://10.86.141.245/ccpwebapp/
    ccp/socialcontact/B5293B5310000134000039590A568DF5"/>
  <author>
    <name>author</name>
  </author>
  <id> https://10.86.141.245/ccp-webapp/ccp/socialcontact/
      B5293B5310000134000039590A568DF5</id>
  <updated>2012-01-06T22:35:05Z</updated>
  <published>2012-01-06T22:35:05Z</published>
  <content type="application/xml">

  <ChatTranscript xmlns="">
    <endDate>1325889843282</endDate>
    <participants>
      <participant>author@user</participant>
      <participant>admin</participant>
    </participants>
    <startDate>1325889305403</startDate>
    <transcript>
    <chat>
      <time>1325889340093</time>
      <name>admin</name>
      <msg>Hi, are you there?</msg>
    </chat>

    <chat>
      <time>1325889364626</time>
      <name>admin</name>
      <msg>Yes, I am here.</msg>
    </chat>
     ...
    </transcript>
  </ChatTranscript>
  </content>
  <dc:creator>author</dc:creator>
  <dc:date>2012-01-06T22:16:34Z</dc:date>
  <ccp:scstatustimestamp>1325888701448</ccp:scstatustimestamp>
  <ccp:scstatus>handled</ccp:scstatus>
  <ccp:scstatususerid>admin</ccp:scstatususerid>
  <ccp:sctags>
  <ccp:sctag>gg</ccp:sctag>
  </ccp:sctags>
 </entry>

 <entry>
  <title>SearchChat</title>
  <link rel="alternate" href="https://10.86.141.245/ccpwebapp/
     ccp/socialcontact/B4F15D6D1000013400002AEE0A568DF5"/>
  <link rel="socialcontact" href="https://10.86.141.245/ccpwebapp/
     ccp/socialcontact/B4F15D6D1000013400002AEE0A568DF5"/>
  <author>
    <name>author</name>
  </author>
  <id> https://10.86.141.245/ccp-webapp/ccp/socialcontact/
       B4F15D6D1000013400002AEE0A568DF5</id>
  <updated>2012-01-06T21:34:04Z</updated>
  <published>2012-01-06T21:34:04Z</published>
  <dc:creator>author</dc:creator>
  <dc:date>2012-01-06T21:34:04Z</dc:date>
  <ccp:scstatustimestamp>1325888037427</ccp:scstatustimestamp>
  <ccp:scstatus>handled</ccp:scstatus>
  <ccp:scstatususerid>admin</ccp:scstatususerid>
  <ccp:sctags>
  <ccp:sctag>gg</ccp:sctag>
  </ccp:sctags>
 </entry>
</feed>
```

Social contact API parameters

Parameters are optional unless otherwise noted.

Parameter	Description	Notes
author	The social contact author name.	String. Required for POST.
description	The body of the social contact.	String.
extensionField	A wrapper tag for a custom name and value pair.	
extensionFields	A collection of custom name and value pairs.	The person submitting the social contact may specify up to 100 pairs, and the entire collection can contain up to one megabyte of information. To update user data, provide a new value for the existing extension field name. If you include an extension field element with the name but no value, the corresponding name and value pair will be deleted during the update.

Parameter	Description	Notes
feedRefURL	The feed refURL that the social contact is associated with.	String. Required for POST.
isSoftLocked	Indicates that the social contact should not be modified, but is not enforced in the API.	Boolean. The SocialMiner UI will not permit any modifications via the UI to a social contact when isSoftLocked is set to true. This is normally used by SocialMiner when a social contact is to be or has been queued to an outside entity.
isInvited	Indicates if the social contact was created as a result of a chat invitation.	Boolean. Default is false. If the Social Contact was created as a result of a chat invitation, then isInvited must be set to true.
inviteStatus	The status of chat invitations sent (if any) from this social contact.	String, case-insensitive. Default is NONE. Valid values are: • NONE (no chat invitations were sent from this social contact) • SENT (chat invitations were sent from this social contact) • EXPIRED (chat invitations were sent but were not accepted by the customer)
publishedDate	The social contact published date.	String. Leave blank to use the current timestamp or provide a valid Unix timestamp.
refURL	A copy of the URL requested.	
replyTemplateRefURL	The reference URL of the reply template. This can be used to retrieve further template details.	Returned by update if the social contact is associated with a feed that has been configured to use a reply template.

Parameter	Description	Notes
replyTemplateURL	The URL of the reply template.	Returned by update if the social contact is associated with a feed that has been configured to use a reply template.
shortUrlId/shortUrlIds	Is a list of short URL Ids which were generated for this social contact.	String. A short URL is generated by the system when a SocialMiner user sends a chat invitation to a customer. The social contact from which the invitation was sent maintains a short URL for each invitation.
sourceType	Is the type of feed this social contact came from: rss, facebook, twitter_stream, twitter_account, ,twitter_search, callback, chat, or push.	String. This is set by SocialMiner and cannot be set through the create or update APIs.
status	One of: • **unread—**The default state of a new contact. • **reserved—**Reserved to be handled. • **handled—**This contact has been handled and no further action is required. • **discarded—**This contact does not require a response and is filed in the recycle bin. • **queued—**The contact is in the process of or has been routed to some external entity. • **draft—**A draft response to the contact has been created and saved, but not sent.	String (case-sensitive). If the submitted and the current status of the social contact are not equal, the submitted status becomes the effective status of the social contact. The QUEUED status is settable from the API. However, currently, it is only set internally by SocialMiner. SocialMiner sets the social contact status to QUEUED when a social contact is to be or has been routed to some external entity.
statusReason	The reason why the contact is in the current state.	
statusUserId	The user modifying the status to a state other than UNREAD.	The value changes to the user who is currently authenticated against the API.

Parameter	Description	Notes
statusTimestamp	The time stamp of the last state change of the social contact.	Long integer. Required for PUT. **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.
tag/tags	One or more tags to associate with this social contact.	The tags can be new or existing tags. If you include the *tags* element, but do not include any *tag* elements, then tags are deleted during an update.
title	The title of the social contact.	String. Required for POST.
transcriptRefURL	Is a URL to get the chat transcript.	String. Only applies to contacts where sourceType is chat.

CHAPTER 26

Tag

SocialMiner supports the labeling of contacts with tags. Tags can be added, edited, and removed to or from a social contact using the social contact API (Social contact, on page 145).

Tag API command

This section describes the supported command (GET) for the tag API and the parameters for that command.

GET (list)

List all configured tags that exist.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/tag
HTTP method:	GET

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

CHAPTER 27

Twitter reply

The Twitter reply API allows you to respond to tweets or Twitter direct messages. You must configure a Twitter account, stream, or search feed before you can use this API.

Note The Shindig OpenSocial container in which SocialMiner runs requires that REST requests complete within five seconds. Communication with 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.

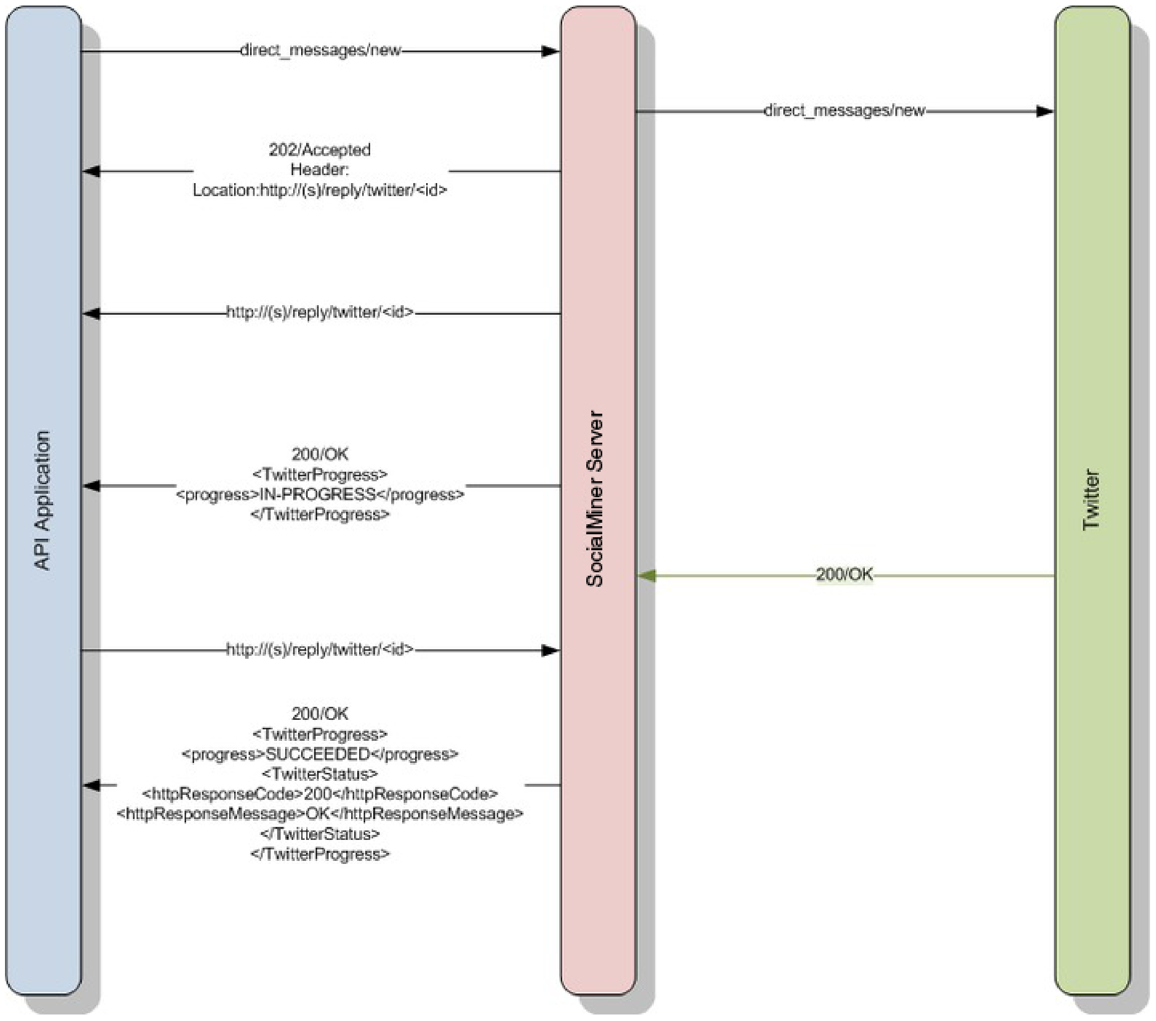
The SocialMiner Troubleshooting Tips has a list of common Twitter errors and their causes. Refer to http://dev.twitter.com/pages/responses_errors for additional information about Twitter errors.

See also Authorize against Twitter feeds, on page 73.

Twitter follow API flow

This diagram illustrates the flow of API calls and expected poll responses for Twitter follow.

Figure 3: Twitter Follow API Flow diagram


API Application
SocialMiner Server
Twitter
direct_messages/new
direct_messages/new
202/Accepted
Header:
Location:http://(s)/reply/twitter/<id>
http://(s)/reply/twitter/<id>
200/OK
<TwitterProgress>
<progress>IN-PROGRESS</progress>
</TwitterProgress>
200/OK
http://(s)/reply/twitter/<id>
200/OK
<TwitterProgress>
<progress>SUCCEEDED</progress>
<TwitterStatus>
<httpResponseCode>200</httpResponseCode>
<httpResponseMessage>OK</httpResponseMessage>
</TwitterStatus>
</TwitterProgress>

SocialMiner will wait for a response from Twitter for 30 seconds. If SocialMiner does not receive a response within 30 seconds, SocialMiner will fail the request and return an http response code of 408.

Twitter reply API commands

This section describes the supported commands for the Twitter reply API and the parameters for those commands.

Related Topics

GET, on page 161
GET (user), on page 162
GET (friendships/exists), on page 163
POST (create status - tweet), on page 163
POST (create direct message), on page 164
POST (create retweet), on page 165
POST (create follow), on page 166
POST (create unfollow), on page 166

GET

Gets the status of a Twitter Reply API call.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/reply/twitter/<id> For more information about the elements in the URL, see API conventions, on page 1. In this instance, <id> represents the ProgressID being requested.
HTTP method:	GET

Example response:	If the fields in the initial request are valid, the Location field in the response contains the URL for the social contact associated with the post being queried.
	If the operation succeeds, the response returns the following XML.
	`<TwitterProgress>`
	` <progress>SUCCEEDED|FAILED|IN-PROGRESS</progress>`
	` <TwitterStatus></TwitterStatus>`
	` <httpResponseCode>responseCode</httpResponseCode>`
	` <httpResponseMessage>responseMessage</httpResponseMessage>`
	`</TwitterProgress>`
	If the operation fails, the httpResponseCode and httpResponseMessage fields contain the code and message returned by Twitter. The apiErrors field can contain additional detailed information about the error.

Response payload:	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**: 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. • **ApiErrors**: a list of errors describing the failure. • **ApiError**: individual error details. • **ErrorMessage**: the error details returned by Twitter. • **ErrorType**: SocialMiner's translation of the error returned by Twitter. SocialMiner uses this field for internationalization and localization purposes.

GET (user)

Retrieves the profile information of a given Twitter user. It works similarly to the way GET does.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/reply/twitter/users/show
HTTP method:	GET
Parameters:	• **screenName**: the user name of the Twitter account. • **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 users/show call is made to Twitter using the account user's oAuth credentials.
Example http request:	`http://<ServerIP>:<Port>/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 for the social contact associated with the post being queried.
	If the operation succeeds, the polling URL
	(`http://<ServerIP>:<Port>/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>`
	If the operation fails, the httpResponseCode and httpResponseMessage fields will contain the code and message returned by Twitter.

GET (friendships/exists)

Determines if user_a is following user_b.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/reply/twitter/friendships/exists
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 to 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 users/show call is made to Twitter using the account user's oAuth credentials.
Example http request:	`http://<ServerIP>:<Port>/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 for the social contact associated with the post being queried.
	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>`
	` <TwitterStatus>`
	`  <friends>true|false</friends>`
	` </TwitterStatus>`
	` <httpResponseCode>responseCode</httpResponseCode>`
	` <httpResponseMessage>responseMessage</httpResponseMessage>`
	`</TwitterProgress>`
	If the operation fails, the httpResponseCode and httpResponseMessage fields will contain the code and message returned by Twitter.

POST (create status - tweet)

Sends a Twitter status message (tweet) from a configured twitter account.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/reply/twitter/statuses/update
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 (which is limited to 140 characters). • **inReplyToStatusId**: optional. If this tweet is in reply to another tweet, use this field to specify the ID of the original tweet.

Example XML request payload:	`<Status>`
	` <username>twitterScreenName</username>`
	` <message>Tweet text</message>`
	`</Status>`

Response:	If the fields in the initial request are valid, then the response header's location field contains the URL for the social contact associated with the post being queried.
	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 contain the code and message returned by Twitter.

POST (create direct message)

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

URL:	http://<ServerIP>:<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 (which is 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>`

Response:	If the fields in the initial request are valid, then the response header's location field contains the URL for the social contact associated with the post being queried.
	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 contain the code and message returned by Twitter.

POST (create retweet)

Retweets a Twitter social contact from a configured Twitter account.

URL:	http://<ServerIP>:<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>`

Response:	If the fields in the initial request are valid, then the response header's location field contains the URL for the social contact associated with the post being queried.
	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 contain the code and message returned by Twitter.

POST (create follow)

Follows a user from a configured Twitter account feed.

URL:	http://<ServerIP>:<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 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 for the social contact associated with the post being queried. 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 contain the code and message returned by Twitter.

POST (create unfollow)

Stops following a user from a configured Twitter account feed.

URL:	http://<ServerIP>:<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 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 for the social contact associated with the post being queried.
	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 contain the code and message returned by Twitter.

CHAPTER 28

URL shortener

The URL shortener API provides a shortened version of a longer URL to the public. The shortened URL has an expiration time and may only be used once. Shortened URLs may not be modified after they are created.

There are two supported types for ShortURL: generic and chat_invite. The default type is generic when a short URL is created with no <type> field explicitly provided in the XML body.

Note All time stamps should be expressed as milliseconds since January 1,1970 in UTC/GMT.

See also Public URL Prefix for Chat Invitation for information on building the full URL.

URL shortener API commands

This section describes the supported commands for the URL shortener API and the parameters for those commands.

POST

Creates a shortened URL.

Based on the type of shortURL created, there are different required fields: generic shortURLs require only url; chat_invite shortURLs require url, campaignRefURL and scRefURL.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/shorturl/
HTTP method:	POST
Parameters:	**Type**: optional. The shortURL type. Valid values are generic (the default) and chat_invite. **URL**: required. The URL being shortened (required).

	campaignRefURL: required for chat_invite type. The refURL of the campaign for which this short URL is being created. **scRefURL**: required for chat_invite type. The refURL of the inviting social contact. **active**: optional. For the chat_invite type, active means that the shortURL was successfully sent with the reply as a chat invitation. Valid values are false (the default) or true. **expireDate**: optional. The expiration date of the shortened URL. Defaults to 24 hours from creation. Expiration may be no more than 30 days in the future.
Example XML request payload:	`<ShortURL>` `<type>chat_invite</type>` `<url>http://theurl.com?param1=val1</url>` `<campaignRefURL>http://[ServerIP]:[Port]/ccp-webapp/ccp/campaign/[public ID]</campaignRefURL>` `<scRefURL>http://[ServerIP]:[Port]/ccp-webapp/ccp/socialcontact/[SC ID]</scRefURL>` `<expireDate>[timestamp]</expireDate>` `</ShortURL>` Or, for a generic shortURL: `<ShortURL>` `<url>http://theurl.com?param1=val1</url>` `</ShortURL>` **Note** When using a full url with multiple parameters, special characters must be properly escaped. This means, for example, that characters such as "&" should appear inside the xml body as "&amp;".
HTTP response headers:	If successful, the location field in the http response header will have a URL to the newly created short URL. A GET of the newly created short URL will provide the shortened URL. See API conventions for error information.

GET

Get a shortened URL.

URL:	http:// <ServerIP>:<Port> /ccp-webapp/ccp/shorturl/<id>
HTTP method:	GET

Example XML response payload:	`<ShortURL>`
	`  <type>chat_invite</type>`
	`  <active>false</active>`
	`  <url>http://theurl.com?param1=val1</url>`
	`  <creator>[userID]</creator>`
	`  <campaignRefURL>http://[ServerIP]:[Port]/ccp-webapp/ccp/campaign/`
	`   [public ID]</campaignRefURL>`
	`  <scRefURL>http://[ServerIP]:[Port]/ccp/socialcontact/`
	`   [SC ID]</scRefURL>`
	`  <createdDate>[timestamp]</createdDate>`
	`  <expireDate>[timestamp]</expireDate>`
	`  <usedDate>[timestamp]</usedDate>`
	`  <shortURL>/ccp/s/[id]</shortURL>`
	`  <refURL>http://[ServerIP]:[Port]/ccp-webapp/ccp/shorturl/[id]</refURL>`
	`</ShortURL>`
	
	The "refURL" is a copy of the URL requested.

Elements:	**type**: the shortURL type. **URL**: the URL being shortened. **creator**: user that created the short URL. **active**: for the chat_invite type, active means that the shortURL was successfully sent with the reply as a chat invitation. **campaignRefURL**: the refURL of the campaign for which this short URL was created. **scRefURL**: the refURL of the inviting social contact. **createdDate**: timestamp on which short URL was created. **expireDate**: the expiration date of the shortened URL. **usedDate**: the timestamp on which this URL was used. Empty if not used yet. **shortURL**: the shortened URL absolute path on the SocialMiner server. **refURL**: the URL of this short URL object.
HTTP response headers:	See HTTP responses.

GET (list)

List all shortened URLs.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/shorturl/
HTTP method:	GET

Example XML request payload:	`<shortURLs>`
	
	`  <ShortURL>`
	`    <type>generic</type>`
	`    <active>false</active>`
	`    <url>http://theurl.com?param1=val1</url>`
	`    <creator>[userID]</creator>`
	`    <createdDate>[timestamp]</createdDate>`
	`    <expireDate>[timestamp]</expireDate>`
	`    <usedDate>[timestamp]</usedDate>`
	`    <shortURL>/ccp-webapp/ccp/s/[id]</shortURL>`
	`    <refURL>http://[ServerIP]:[Port]`
	`     /ccp-webapp/ccp/shorturl/[id]</refURL>`
	`  </ShortURL>`
	
	`  <ShortURL>`
	`    <type>chat_invite</type>`
	`    <active>true</active>`
	`    <url>http://cisco.com/index.htm</url>`
	`    <creator>[userID]</creator>`
	`    <campaignRefURL>http://[ServerIP]:[Port]/ccp-webapp/ccp/campaign/`
	`     [public ID]</campaignRefURL>`
	`    <scRefURL>http://[ServerIP]:[Port]/ccp-webapp/ccp/socialcontact/`
	`     [SC ID]</scRefURL>`
	`    <expireDate>[timestamp]</expireDate>`
	`    <usedDate>[timestamp]</usedDate>`
	`    <shortURL>/ccp-webapp/ccp/s/[id]</shortURL>`
	`    <refURL>http://[ServerIP]:[Port]`
	`     /ccp-webapp/ccp/shorturl/[id]</refURL>`
	`  </ShortURL>`
	
	`</shortURLs>`

POST (update)

Update a shortened URL.

Only two fields can be updated on an already created shortened URL : active and usedDate.

Any attempt to update other fields will result in an error.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/shorturl/<id>
HTTP method:	PUT

Example XML request payload:	`<ShortURL>`
	`  <active>true</active>`
	`  <usedDate>[timestamp]</usedDate>`
	`</ShortURL>`

HTTP response headers:	See HTTP responses.

DELETE

Delete a shortened URL.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/shorturl/<id>

HTTP method:	DELETE
HTTP response headers:	See API conventions for error information.

CHAPTER 29

XMPP

The XMPP API allows an administrator to retrieve the existing XMPP server configuration and to update it if necessary. An XMPP server connection is required to send Instant Messaging (IM) notifications.

This API is represented on the SocialMiner user interface in the System Administration panel. Only one XMPP configuration of both Server and User is allowed at this time.

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

XMPP API commands

This section describes the supported commands for the XMPP API and the parameters for those commands.

Related Topics

GET, on page 175
PUT, on page 176
XMPP API parameters, on page 176

GET

Get the XMPP configuration.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/xmpp/default
HTTP method:	GET

Example XML output:	`<Xmpp>`
	`  <xmppService>xmpp.cisco.com</xmppService>`
	`  <xmppHost>[ServerIP]</xmppHost>`
	`  <xmppPort>[Port]</xmppPort>`
	`  <xmppServiceLookup>true</xmppServiceLookup>`
	`  <xmppEnabled>true</xmppEnabled>`
	`  <xmppServiceUsername>gding</xmppServiceUsername>`
	`  <xmppServicePassword>...</xmppServicePassword>`
	`  <refURL>`
	`    http://[ServerIP]:[Port]/ccp-webapp/ccp/`
	`    xmpp/default`
	`  </refURL>`
	`</Xmpp>`

Parameters:	See XMPP API parameters, on page 176.

PUT

Updates the XMPP configuration.

URL:	http://<ServerIP>:<Port>/ccp-webapp/ccp/xmpp/default
HTTP method:	PUT
Parameters:	See XMPP API parameters, on page 176.

XMPP API parameters

All parameters are optional.

Parameter	Description	Notes
xmppEnabled	Identifies whether this XMPP configuration is enabled or disabled.	Boolean. Default is false.
xmppHost	The IP address or hostname of the XMPP server.	String.
xmppPort	The XMPP port number. The default port is 5222.	Integer. Not used if *xmppServiceLookup* is set to *true*.
xmppService	The xmpp service lookup name.	String.

Parameter	Description	Notes
xmppServiceLookup	If this flag is true, the xmppService field will be used to perform a DNS Service lookup for the XMPP Service; otherwise the xmppHost and xmppPort will be used to directly connect to the XMPP server.	Boolean. Default is true.
xmppServiceUserName	The username used to log into the XMPP server.	String.
xmppServicePassword	The password used to log into the XMPP server.	String.

APPENDIX A

Reporting database connection

You connect to the reporting database using Java database connectivity (JDBC). The reporting database runs on Informix.

Reporting database SQL connection

Connection to the SocialMiner Informix reporting database is made using the following format:

```
jdbc:informix-sqli://<hostname>:<port>/

<databaseName>:INFORMIXSERVER=<informixserver>;
```

Where:

- 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. Also, any dashes ("-") in the hostname are replace by underscores ("_").

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

```
jdbc:informix-sqli://my-server.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 panel.

Reporting database schema

The reporting database schema 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 1: mmca_report_campaign

Field name:	Description:	Data type:	Constraints:
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 panel.	nvarchar	Not Null.
lastupdated	Last time this row was updated.	datetime	Not Null.
active	Indicates if the campaign exists in the campaign database.	int; 1 = exists, 0 = deleted	Not Null.

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

Table 2: mmca_campaign_activity

Field name:	Description:	Data type:	Constraints:
recordid	Auto-incrementing ID	serial(8)	Primary key, not Null.
interval	Reporting interval to which this record applies (currently only a 15 minute interval is supported).	datetime	Not Null.

Field name:	Description:	Data type:	Constraints:
campaignid	ID of the campaign from the mmca_reportcampaign table.	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

Field name:	Description:	Data type:	Constraints:
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
chatinvitationssent	The number of chat invitations sent from this campaign during the interval (whether they end up being handled in this campaign or not).	int	Not Null
chatinvitationshandled	The number of chat invitations handled within this campaign during the interval (whether sent from this campaign or not).	int	Not Null
chatinvitationsexpired	The number of chat invitations sent from this campaign that expired during the interval (the customer didn't click the chat invitation link before the invitation timed out).	int	Not Null

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

Table 3: mmca_agent_campaign_activity

Field name:	Description:	Data type:	Constraints:
recordid	Auto-incrementing ID	serial(8)	Primary key, not Null.
interval	Reporting interval to which this record applies (currently only a 15 minute interval is supported).	datetime	Not Null.

Field name:	Description:	Data type:	Constraints:
campaignid	ID of the campaign from the mmca_reportcampaign table.	int(8)	Not Null.
userid	String representing the login name of the user 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.

Field name:	Description:	Data type:	Constraints:
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.
chatinvitationssent	The number of chat invitations sent by the user from this campaign during the interval (whether they end up being handled in this campaign or not).	int	Not Null.
chatinvitationshandled	The number of chat invitations handled by the user within this campaign during the interval (whether sent from this campaign or not).	int	Not Null.
chatinvitationsexpired	The number of chat invitations sent by the user from this campaign that expired during the interval (the customer didn't click the chat invitation link before the invitation timed out).	int	Not Null.

APPENDIX B

SocialMiner server configuration

Security configuration options

SocialMiner may be deployed where some users access the server through a firewall or proxy, and others do not. For the customer chat interface, it is possible to prevent the SocialMiner server from being abused or limiting access for those outside the firewall to specific server functionality by deploying it behind a firewall or proxy server. Within one deployment, a reverse proxy or a firewall may be used — but not both.

Port-forward firewall configuration

When placed behind a port-forwarding firewall, the SocialMiner server is only reachable (for some users) by going through a specific port on a specific machine. All traffic on that port is forwarded to SocialMiner, and there is no alteration of the user's request as it traverses the firewall.

Any port may be chosen through which to forward traffic. Typically this would be port 80 or 443 (for http and https respectively), but there are no restrictions. For a port intended to forward non-SSL (http) traffic, the destination should be port 80 on the SocialMiner server. SSL (https) traffic should be forwarded to port 443.

There is no additional configuration required on the SocialMiner server.

Reverse proxy

A reverse proxy is used to forward specific requests to SocialMiner. During proxying, request headers are altered so that the proxied server has enough original request information to correctly create the served content (for example, so that links reference the proxy host and not the SocialMiner server). http or https may be used at the proxy server and requests may be forward to SocialMiner using either http or https.

The customer chat interface and URL redirect interfaces are supported for reverse proxying.

SocialMiner recognizes the following reverse proxy headers:

Header	Required?	Comments
X-Forwarded-Host	Y	Includes the proxy host name as visible to the user. May also include a port in the form <server name>:<port>
X-Forwarded-Proto	N	If present, determines the protocol of generated links. Defaults to http unless the proxy port is determined to be 443, in which case it will be https. Overrides Front-End-https and X-Forwarded-https values when present.
X-Forwarded-Port	N	If present, is returned by subsequent calls to Request.getServerPort(). If this header is present and a port is provided in X-Forwarded-Host, this value is overridden by the X-Forwarded-Host value.
Front-End-https	N	If present and value is "on", returned links will use https. It overrides X-Forwarded-https when present.
X-Forwarded-https	N	If present and value is "on", returned links will use https.

Note For Apache users, by default Apache will not indicate when SSL was used to reach the proxy server. In order for SocialMiner links to be correctly formatted when SSL is being used between the user browser and the Apache reverse proxy, you must add a request header to proxied requests to tell SocialMiner to use https. You can do this by adding the following to your server configuration:

```
RequestHeader set X-Forwarded-Proto "https"
```

APPENDIX C

XMPP BOSH eventing

SocialMiner sends asynchronous state change and tag update events to an XMPP client using the XMPP Publish-Subscribe protocol (XEP-0060).

Authentication: Only SocialMiner authorized users are allowed to connect to the embedded XMPP server.

Ports: Connect on these ports for eventing:

- Port 7071 for unsecure XMPP BOSH connections
- Port 7443 for secure XMPP BOSH connections

Publish and subscribe

SocialMiner's event mechanism uses XMPP extensions for event subscription and publication. Details can be found here: http://xmpp.org/extensions/xep-0060.html.

When a tag is created or modified or when a contact state changes, the information is published using XMPP. The campaign results panel subscribes to the specific XMPP topic for the selected campaign, receives the change events, and updates the user interface appropriately.

Nodes

In XMPP, publishers publish events to a node. Subscribers subscribe to nodes in order to receive events related to the node. Nodes are string-carried in the XML used to publish and subscribe. These strings are also carried in the notifications sent to subscribers.

Creating a campaign creates a node to allow subscribers to subscribe to events related to the campaign results. This node has the form *ccp.campaign.updates.<campaignpublicId>*, where *campaignpublicId* is the publicId field returned by a campaign's GET request.

When a campaign is deleted, the corresponding node is also deleted.

SocialMiner also creates a global node "ccp.contacts.chat" to publish event related to chat contacts.

Events

Global chat contact events

Field	Description
id	The unique ID of the contact.
author	The author of the contact's status (from the author field of the contact).
title	The title of the contact's status (from the title field of the contact).
status	The status of the contact.
statusUserId	The user who most recently changed the contact status.
statusReason	The reason the contact is in the current state.
statusTimeStamp	The time at which the contact's status was changed.
publishDate	The date when SocialMiner received the contact.
tags	The list of tags associated with the contact.
refURL	The REST reference URL of the contact.
chatIsInvited	A boolean value to indicate whether this chat was initiated using a chat invitation or not.

When an application subscribes to *ccp.contacts.chat node,* it receives events when a contact is updated in one of the following ways:

- The contact's tags are modified. (This happens any time the contact's update REST API includes the tags field.)
- The contact's status is modified.

The actual payload of the XML event is as follows:

```
<SocialContact xmlns="http://jabber.org/protocol/pubsub">
 <author>author1</author>
 <title>title1</title>
 <id>DA476CF81000012F000002FB0A568DF5</id>
 <publishedDate>1305037194000</publishedDate>
 <refURL>http://[ServerIP]:[Port]/ccp-webapp/ccp/socialcontact/
    DA476CF81000012F000002FB0A568DF5
 </refURL>
 <status>reserved</status>
 <statusTimestamp>1305037210727</statusTimestamp>
 <statusUserId>admin</statusUserId>
 <chatIsInvited>false</chatIsInvited>
 <tags>
  <tag>tag1</tag>
  <tag>tag2</tag>
 </tags>
</SocialContact>
```

Campaign contact events

Each campaign results event contains the following attributes of a social contact:

Field	Description
id	The unique ID of the social contact.
status	The status of the contact.
statusUserId	The user who most recently changed the contact status.
statusTimeStamp	The time at which the social contact's status was changed.
publishDate	The date when SocialMiner received the contact.
tags	The list of tags associated with the contact.
refURL	The REST reference URL of the social contact.
campaignpublicId	The ID of the campaign to which this contact belongs.

When an application subscribes to *campaign.updates*, it receives events when a social contact associated with that campaign changes in one of the following ways:

- The social contact's tags are modified. (This happens any time the social contact's update REST API includes the tags field.)
- The social contact's status is modified.

The actual payload of the XML event is as follows:

```
<SocialContact xmlns="http://jabber.org/protocol/pubsub">
 <campaignpublicId>
   EventingCampaign-07192-0000000000012
 </campaignpublicId>
 <id>DA476CF81000012F000002FB0A568DF5</id>
 <publishedDate>1305037194000</publishedDate>
 <refURL>http://[ServerIP]:[Port]/ccp-webapp/ccp/socialcontact/
   DA476CF81000012F000002FB0A568DF5
 </refURL>
 <status>reserved</status>
 <statusTimestamp>1305037210727</statusTimestamp>
 <statusUserId>admin</statusUserId>
 <tags>
  <tag>tag1</tag>
  <tag>tag2</tag>
 </tags>
</SocialContact>
```

APPENDIX D

Custom reply templates

This information is intended for experienced web developers wishing to create a custom reply template. Developers should already be familiar with HTML and Javascript (including AJAX).

Custom reply templates give developers a way to provide extended functionality to users. Templates can interact with content on a variety of servers and can deliver rich user experiences; even embedding entire sub-applications within the SocialMiner interface.

SocialMiner provides a Javascript API to interact with the template container to access SocialMiner REST APIs, display messages, and close the template when finished. This section describes the primary files and functions used when developing custom reply templates. It also covers how to migrate templates written against pre-9.0.1 SocialMiner releases.

Javascript concepts

Gadgets API

Custom reply templates are instances of OpenSocial gadgets (see http://opensocial.org). Gadgets are web pages that adhere to a certain format and that have access to a gadgets Javascript API. SocialMiner gadgets (including custom reply gadgets) are hosted within an Apache Shindig 2.0-based container. For further reading on gadget development, see the Shindig website or the Google gadgets API reference site.

Gadgets exist in the browser within an IFrame. Therefore, gadgets may not directly call anything outside of their frame but instead must communicate using the postMessage Javascript function. The SocialMiner gadget container waits for certain messages on this mechanism before displaying user messages or closing reply templates.

In addition to IFrame limitations, gadgets are subject to standard same-domain request policies, meaning that AJAX POST, PUT, and DELETE requests may only be made to the host that originally served the gadget web page (in our case, the SocialMiner server). In practice, this means that all API requests must use the makeRequest function provided by the gadgets API to make gadget container-proxied requests for REST service URLs.

Javascript files

All of the objects and functions are found in ccp-base.js. This file is available from a SocialMiner server at http://<server name or IP>/templates/reply/js/ccp-base.js. Note that anything not documented in the API reference may be modified in the future.

Getting started

A basic example custom reply gadget is provided on the SocialMiner server at http://<server name or IP>/templates/reply/custom_reply_sample.jsp. Developers may download and experiment with this example to understand basic template structure and use of the CcpSession object.

SocialMiner Javascript objects

Documentation of SocialMiner Javascript APIs can be found on the Cisco Developer Network under Tools and Samples.

CcpSession

The CcpSession object facilitates interaction with the reply template gadget container. The first thing a reply template gadget does is to create this object using the gadget page URL.

APIMessage

Reply template gadgets interact with any web service that the server communicates with by using the gadgets.io.makeRequest function. The APIMessage objects provide a convenient way of calling APIs that invoke long-running operations (operations that don't immediately return a 200 on success, but instead return 201 or 202). APIMessages poll an operation until it is complete and only then call any provided callback.

Template migration

Earlier versions of SocialMiner organized reply template API calls differently than they are organized in more current releases. In the 9.0(1) release, these calls are collected into consistent Javascript namespaces and some objects are renamed. If you have already created a template that works with an earlier version of SocialMiner, you will need to make changes for it to work with 9.0(1) and subsequent releases.

Namespaces

All functions, objects, and constants are organized into namespaces. Namespaces are similar to packages in Java and are used to prevent object name collisions in the global/window namespace. Namespaces in SocialMiner start with 'ccp' and are further subdivided by object, function type, or purpose.

Namespace objects are bound to the window object in Javascript and must always be referenced using their full dot-separated name.

Changed object and function names

A number of name changes were made to objects and functions. References should be updated to the new names indicated in the table below.

Old name	New name
CcpSession <object>	ccp.base.CcpSession

Old name	New name
TwitterMessage <object>	ccp.base.APIMessage
log <function>	ccp.log
isEmptyString <function>	ccp.utils.isEmptyString
getRequestParameterValue <function>	ccp.utils.getRequestParameterValue
getElementValue <function>	ccp.utils.getElementValue
getErrorStringForApiError <function>	ccp.utils.getErrorStringForApiError
getDetailedErrorFromXml <function>	ccp.utils.getDetailedErrorFromXml
getErrorDetails <function>	ccp.utils.getErrorDetails

Removed Javascript files

The files listed in the table below were removed and their functionality consolidated into other files as indicated.

File	Migration action
template-utils.js	Similarly-named functions in CcpSession replace functions in this file.
dialog-manager.js	Replaced with jQuery UI Dialog.
social-contact.js	The SocialContact object is now in ccp-base.js and is named ccp.base.SocialContact.
jquery.ui.*	jQuery UI files are now combined into jquery-ui.custom.min.js.