MICRO FOCUS

TeamWorks 18.1

Using the Real-time WebSocket APIs

April 2018

Legal Notice

Contents

About This Guide

Audience

This guide is intended for GroupWise TeamWorks developers.

Feedback

We want to hear your comments and suggestions about this manual and the other documentation included with this product. Please use the **comment on this topic** link at the bottom of each page of the online documentation.

Documentation Updates

For the most recent version of the this guide, visit the TeamWorks API Documentation on (http://wwwtest.provo.novell.com/documentation/teamworks-18).

1 Introduction

The TeamWorks Real-time API is a WebSocket-based API that allows clients to receive information about updates in the system as they occur. It also has limited RPC capabilites.

2 Using the Real-time API

2.1 Establishing a connection

The WebSocket URL for the Real-time API is:

`wss://server_IP_or_DNS:8443/ssf/websocket/default`

In order to establish a WebSocket connection, the WebSocket protocol specifies that the client is to connect to the server and send an HTTP Upgrade request. The TeamWorks Real-time API requires that this HTTP upgrade request include Basic Authentication information in order to authenticate the user.

Most WebSocket client libraries hide the details of this Upgrade request, only require the WebSocket URL, and let you supply custom headers. For example, the following code establishes a WebSocket connection using the Python websocket-client library (https://pypi.python.org/pypi/websocket-client):

```
import base64
import websocket
auth = 'testuser:testpasswd'
headers = ['Authorization: Basic %s' % base64.b64encode(auth)]
ws = websocket.WebSocketApp(
  'wss://amethyst.provo.novell.com:8443/ssf/websocket/default',
  header = headers)
```

Once the connection is successfully established and authenticated, the client will receive a `hello` event message from the server.

2.1.1 Formatting Preferences

Each WebSocket connection has formatting preferences that control how certain messages that the server sends are formatted. The client can control these preferences with the `set_format_preferences` RPC method.

Currently, the only supported formatting preference is mention_format, which can be either *readable* (default) or *raw*. This preference controls how references to users (mentions) in topics and comments are formatted. Raw mentions are in the format `@[{object-type}:{object-id}:{display-text}]`, while readable mentions are `@{display-text}`.

Raw Example

```
{
    “@type”: “topic”,
    "id": "123",
    “body”: “Hi, @[user:97:John Doe]”
}
```

For more information about RPC requests, see “RPC” on page 15.

Readable Example

```
{
    “@type”: “topic”,
    "id": "123",
    “body”: “Hi, @John Doe”
}
```

2.2 Real-time event streams

Event streams are a continuous feed of updates (events) related to a particular entity in the system. By subscribing to streams, real-time API clients will receive these events as they occur.

An event message contains information about the type of event, the stream it belongs to and the affected entity. For example:

```
{
   "message_type": "event",
   "event_type": "topic_created",
   "event_stream": {
      "type": "room",
      "id": "254"
   },
   "entity": {
      // ... Topic object ...
   }
}
```

For more information about the entity objects (Room, Topic, Comment, etc.), see the Data Types section of the REST API documentation.

2.2.1 Session stream

The session stream is a source for events related to the client’s WebSocket session. The client is automatically subscribed to the session stream.

Session events

Event Type	Entity Object Type	Description
`hello`	None	The client session has been successfully established

Event Type	Entity Object Type	Description
`stream_unsubscribed`	`EventStream`	The client session has been unsubscribed from the stream. Sessions are unsubscribed from streams in the following situations: ◆ The client sends an `unsubscribe_from_stream` RPC request. ◆ The room is deleted. ◆ The user loses access to the room because the room was changed to a private room. ◆ The user loses access to the room because the user was removed as a member of the private room. ◆ User access to the real-time API has been disabled for system maintenance.

2.2.2 User stream

The user stream is a source for events related to a particular user. When a client connects to the real-time API, it is automatically subscribed to the authenticated user's stream.

User events

Event Type	Entity Object Type	Description
`favorite_room_added`	`Room`	A room had been added to the user's favorite rooms list. Rooms are added in each of the following situations: ◆ The user marks the room as a favorite. ◆ The user creates a room. ◆ The user is added as a member of a room.

Event Type	Entity Object Type	Description
`favorite_room_removed`	`RoomReference`	A room has been removed from the user's favorite rooms list. Rooms are removed in the following situations: ◆ The user unfavorites a room. ◆ A favorite room is deleted. ◆ The user is removed as a member of a private room. ◆ A favorite room is no longer visible to the user because it was changed to a private room.
`notification`	`Notification`	The user has received a notification.
`notifications_refresh_required`	`None`	One or more of the user's notifications have been removed because the room, topic, or comment they reference has been deleted. The client should refresh any data it has dealing with the user's notifications.
`room_visited`	`UserRoomDetails`	The user has visited the room, and the user's last visit date has been updated for the room. The last visit date determines which topics and comments in the room are considered to be new.

2.2.3 Room stream

A room stream is a source for events related to a particular room. The client is not automatically subscribed to any room streams. In a typical TeamWorks client application, the client will subscribe to a room stream when the user enters that room in the UI. This allows the client to update the UI immediately when changes occur in the room, for example, when new topics or comments are added. When the user leaves the room and goes to another place in the client application, the client will unsubscribe from the room stream because it no longer needs to know about updates in that room in real-time.

Subscribing to a room stream

To subscribe to a room stream, the client sends a subscribe_to_stream RPC request message via the WebSocket:

```
{
   "message_type": "rpc_request",
   "method": "subscribe_to_stream",
   "request_id": "12345abcde",
   "entity": {
      "@type": "stream",
      "type": "room",
      "id": "236"
   }
}
```

For more information about RPC requests, see “RPC” on page 15.

Room events

Event Type	Entity Object Type	Description
`comment_created`	`Comment`	Someone has commented on a topic in the room.
`comment_deleted`	`CommentReference`	A comment in the room has been deleted.
`comment_updated`	`Comment`	A comment in the room has been updated.
`member_added`	`RoomMembership`	A user has been added as a member of the room.
`member_removed`	`RoomMembership`	A user has been removed as a member of the room.
`room_deleted`	`RoomReference`	The room has been deleted.
`room_updated`	`Room`	The room has been updated.
`topic_created`	`Topic`	A new topic has been created in the room.
`topic_deleted`	`TopicReference`	A topic has been deleted.
`topic_updated`	`Topic`	A topic has been updated.

2.2.4 Room badge stream

A room badge stream is a source for "summary" events related to a particular room. It provides a limited set of events that allow clients to track when messages are added to a room.

Like room streams, the client is not automatically subscribed to any room badge streams. In a typical TeamWorks client application, the client will subscribe to each badge stream in the user's favorite rooms list. This way, the client can present the favorite rooms list with information indicating whether or not the room has new content since the last time the user visited the room (room badging).

Subscribing to a room badge stream

To subscribe to a room stream, the client sends a "subscribe_to_stream" RPC request message via the WebSocket:

```
{
   "message_type": "rpc_request",
   "method": "subscribe_to_stream",
   "request_id": "12345abcde",
   "entity": {
      "@type": "stream",
      "type": "room_badge",
      "id": "236"
   }
}
```

For more information about RPC requests, seeSection 2.3, “RPC,” on page 15.

Room events

Event Type	Entity Object Type	Description
`member_added`	`RoomMembership`	A user has been added as a member of the room.
`member_removed`	`RoomMembership`	A user has been removed as a member of the room.
`message_posted`	`PostedMessage`	A new message (topic or comment) has been posted to the room.
`room_deleted`	`RoomReference`	The room has been deleted.
`room_updated`	`Room`	The room has been updated.

2.2.5 System stream

The system stream is a source for events related to the TeamWorks system. The client is automatically subscribed to the system stream.

System events

Event Type	Entity Object Type	Description
`api_available`	`None`	Access to the Real-time API has been reenabled. The client can resubscribe to room streams and receive real-time events again.
`api_unavailable`	`None`	Access to the Real-time API has been disabled due to system maintenance. When access is disabled, the websocket session is unsubscribed from all room and room badge streams and all RPC requests will fail with an `API_UNAVAILABLE` error. Once access is reenabled, the server will send an `api-available` event.

2.3 RPC

The TeamWorks Real-time API supports RPC requests. Some of these RPC methods deal with managing the WebSocket session and others make changes in the TeamWorks system.

The RPC methods that make changes to the TeamWorks system are also available in the REST API. For example, both APIs provide the ability to comment on a topic. Because each REST request has overhead when establishing a connection and authenticating the user, the Real-time API is more efficient. However, the Real-time API does not duplicate all functionality available through REST. It only supports actions where the resulting efficiency and immediacy significantly improve the user experience.

Making an RPC request involves sending an RPC request message to the server. The server will process the request asynchronously and send an RPC response message sometime later. If a client sends multiple requests, the server makes no guarantee about the order of the response messages. The client must keep track of pending requests in order to match response messages with the original request message.

To help the client do this, the request and response messages include a `request_id` field. The server includes in the response whatever ID the client provided in the request message. As a best practice, the client should generate a unique ID for each request. These IDs need only be unique for the client session. Something as simple as a number sequence, i.e. `1, 2, 3`, etc., is sufficient.

NOTE: If the request message is not well-formed JSON, or if it specifies an invalid value for an enumerated type (such as EventStream type), the server will be unable to parse the message. When this occurs, the server is not able to read the `request_id` or `method` fields and therefore does not include those fields in the response message.

2.3.1 Request Messages

```
{
   "message_type": "rpc_request",
   "method": "create_topic",
   "request_id": "12345abcde",
   "entity": {
      "@type": "topic",
      "room":{
         "id": "456"
      },
      "body": "Hi, everyone, here’s a new topic to discuss."
   }
}
```

2.3.2 Response Messages

```
{
   "message_type": "rpc_request",
   "method": "create_topic",
   "request_id": "12345abcde",
   "status": {
      "status_code": 200
   }
   "entity": {
      "@type": "topic",
      "room":{
         "id": "456"
      },
      "body": "Hi, everyone, here's a new topic to discuss."
   }
}
```

Status

The status object in the response message indicates whether the operation completely successfully. It contains a numeric status code, which is patterned after HTTP status codes.

Typical status codes are:

- `200` (Success)
- `400` (Bad request)
- `403` (Forbidden)
- `404` (Not found)
- `409` (Conflict)
- `500` (Internal server error)
- `503` (Service Unavailable)

NOTE: For status_code numbers greater than or equal to 300, the status object will also include additional information, for example:

```
"status": {
   "status_code": 404,
   "error": {
      "code": "ROOM_NOT_FOUND",
      "message": "No room exists with ID 234"
   }
}
```

For more information about possible error codes, see *{list of error codes shared between the REST and real-time APIs in the doc}* (https://www.novell.com/documentation/??).

2.3.3 Methods

The following table shows the supported RPC methods. For more information about request and response entities, see the Data Types section of the REST API documentation.

Method	Request Entity Object	Response Entity Object	Description
create_comment	Comment	Comment	Add a comment to a topic or comment
create_topic	Topic	Topic	Post a new topic to a room.
delete_comment	Comment or String (comment HRef)	CommentReference	Delete a comment
delete_topic	Topic or String (topic HRef)	TopicReference	Delete a topic
get_format_preferences	None	FormatPreferences	Retrieve message formatting preferences
ping	None	None	Ping the server to test the validity of the web socket session
set_format_preferences	FormatPreferences	FormatPreferences	Update message formatting preferences
subscribe_to_stream	EventStream	EventStream	Subscribe to the specified stream
update_comment	Comment	Comment	Update a comment
update_topic	Topic	Topic	Update a topic
unsubscribe_from_stream	EventStream	EventStream	Unsubscribe from the specified stream