Using PCore and PConnect Public APIs '24.2

5 September 2025

CONTENTS

Working with Messaging Service

Learn about the Messaging Service and its various use cases.

The Messaging Service serves as a bridge between information publishers and subscribers. It forwards the published information to WebSocket subscribers.

The publishers are typically third party integration services or the Infinity Case engine, while the subscribers are UI components in browsers running Constellation architecture UI.

The Messaging Service is exposed to browsers and publishers through a network IP address. This IP address is passed to the browser during the initial portal load from a Dynamic System Setting (DSS) in Infinity. This IP address is used by publishers for HTTP POST calls and by subscribers for Web Socket Secure (WSS) connections.

In a client/browser, the Messaging Service is accessed through the PCore getMessagingServiceManager() API.

- **Architectural overview**
- **Message schema**
- **Client-side subscriptions**
- **Server-side publishing**

Architectural overview

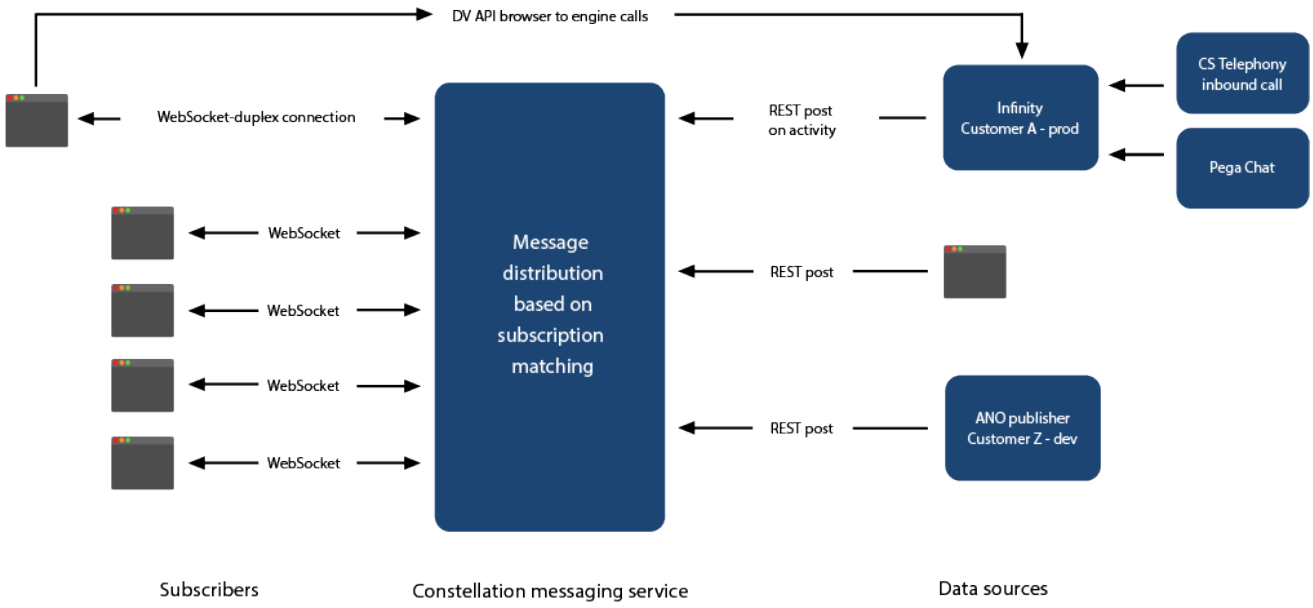
Learn about the components of the Messaging Service and how they are organized.

The server to client push architecture is used to keep UI content updated with event-based information beyond the last server interaction. The messaging service uses WebSocket technology to maintain the connections between the browser and server.

The communication between the Messaging Service and browser must follow the standard messaging format. For more information on the messaging format and schema, see Message schema.

Architecture

Browser components subscribe to topics on the Message Broker through a WebSocket connection. Data sources such as Infinity publish data to the Message Broker through a HTTP REST connection. The Message Broker pushes this new information to those browsers that have registered an interest in this information.



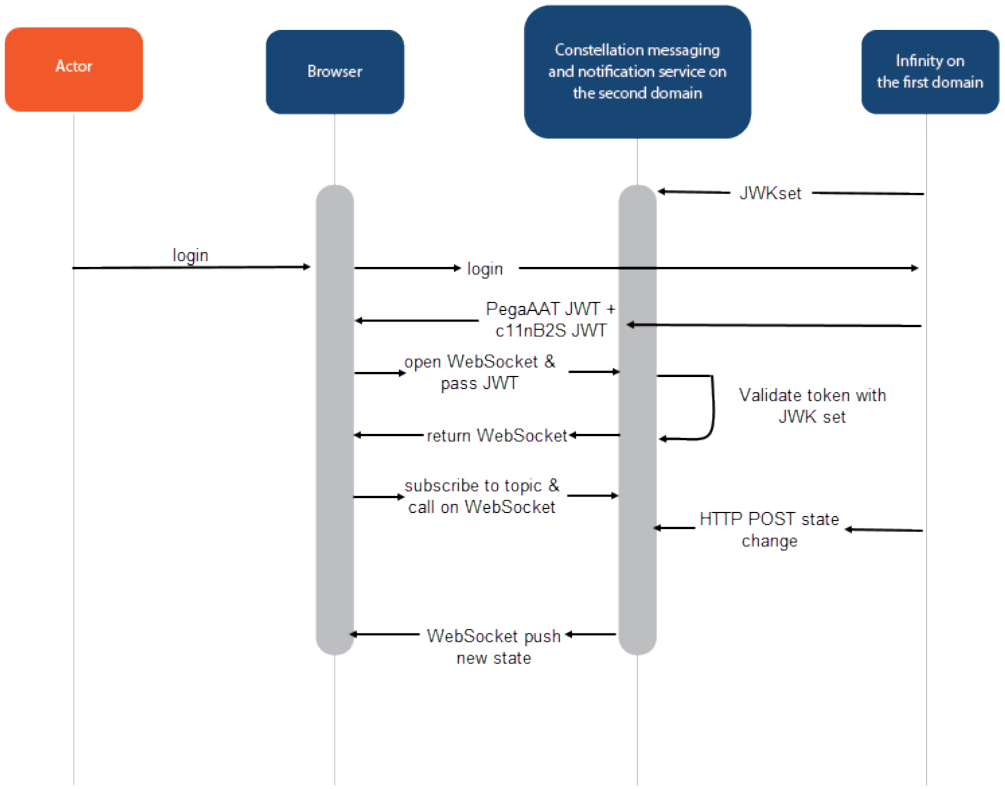
Messaging Service Architecture

Security

In the Messaging Service, the JSON Web Tokens (JWT) and the data to be published are validated to ensure secure WebSocket connections and trustworthy browser calls.

When the WebSocket connection is being created for the browser, the JWT issued by Infinity to the user during their login is validated with the JWKset file issued by Infinity

directly to the Constellation Message Broker. The WebSocket connection is created only if this validation is successful.



Messaging Service Authentication

Message schema

Learn about the standard formats for messages to be sent between the Messaging Service and a browser.

Subscribe format

Apply this format when you want to subscribe to WebSockets.

Format

```
{
  "filter":{
    "matcher":"string: unique name to define subject eq., pulse, interaction, cha
t",
    "criteria":{
      "custom props..":"custom values.."
    }
  },
  "subscribe":{
    "id":"string: client id"
  }
}
```

Usage example

```
{
    "filter":{
      "matcher":"PULSE",
      "criteria":{
        "user":"alex",
        "workId":"EPIC-2021"
      }
    },
   "subscribe":{
    "id":"1214455234_1"
  }
}
```

Unsubscribe format

Apply this format when you want to unsubscribe from WebSockets.

Format

```
{
   "unsubscribe":{
     "id":"string: client id"
   }
}
```

Usage example

```
{
   "unsubscribe":{
     "id":"1214455234_1"
   }
}
```

Message delivery format

Apply this format when you want to send the message to a subscriber matching the specified criteria. If the subscription criteria does not match the criteria of the sent message, the message is discarded.

Format

```
{
     "filter":{
       "matcher":"string: unique name to define subject eq., pulse, interaction",
       "criteria":{
          "custom props..":"custom values.."
       }
     },
     "message":"object : message json"
}
```

Usage example

```
{
    "filter":{
      "matcher":"PULSE",
      "criteria ":{
        "user":"alex",
        "workId":"EPIC-2021"
      }
    },
    "message":{
      "updateById":"admin@constellation.com",
      "messageId":"Pulse P-1021"
    }
}
```

Client-side subscriptions

Learn how a client can use Messaging Service to subscribe to WebSocket events.

For a client, such as a browser component to subscribe to WebSocket event, it must use a Messaging Service API. For example, to obtain updated data from a server or to access events from the server, the data or events are sent to the browser through message schema payloads. The Messaging Service API is used to connect the browser to the server WebSockets.

For more information on the Messaging Service API, see subscribe(filter, messageHandler, contextName, id).

Server-side publishing

Learn how events are published to subscribed clients.

For clients to receive information about data that is updated at the back-end, the server publishes a message through a REST API. In Infinity, server-side publishing is exposed through the `pxC11NPublishMessage` activity. This activity sends a REST call to the Messaging Service.