

## WebView plugin SDK basics

To support user data retrieval, trade execution, event handling and more, a WebView plugin (client) must establish reliable communication with the cTrader application (host). The WebView plugin SDK facilitates these interactions with cTrader, and this guide explains the communication pipeline between the client and the host.

### Installation

The WebView plugin SDK is available as an **npm package**. To install the SDK, navigate to your project directory in a terminal window and then run this command:

```
npm install @spotware-web-team/sdk
```

### Client-host synchronisation

#### Note

The relevant protocols are available as reference **messages**.

Before any message can be exchanged, a communication channel must be established. The typical operations include:

1. At the start of communication, an alive event exchange occurs where both the client and the host dispatch an event.
2. The host sends a registration event (payloadType = 2000). Example:

```
window.dispatchEvent(new MessageEvent('message-from-host', { data: {"clientId": "152cf2d4-bfb6-4426-a303-9243b6a0daf4", "payload": {}, "payloadType": 2000} })))
```

3. The WebView plugin sends a confirmation event (payloadType = 2001). Example:

```
window.dispatchEvent(new MessageEvent('message-to-host', { data: {"clientId": "152cf2d4-bfb6-4426-a303-9243b6a0daf4", "payload": {}, "payloadType": 2001} })))
```

4. For each new confirmation event, a new registration event is sent to ensure synchronisation, even if the plugin reinitialises.

#### Note

If either party fails to receive the corresponding alive event, the plugin will not mount. To avoid loops, the plugin must send the confirmation event only once and track the message flow.

#### Warning

Sending multiple confirmation events triggers repeated registration events, potentially resulting in an unintended loop.

### Requests

Once synchronisation is complete, the plugin can interact with cTrader services by calling the **WV plugin SDK methods** that communicate internally with the host.

1. The plugin calls an SDK method. For example, it can request a user's **account information**.
2. The WebView plugin sends a request. Example:

```
{
  "payloadType": 2100,
  "payload": {
    "payloadType": 175,
    "clientId": "asd1-asd1"
  },
  "clientId": "asd2-asd2"
}
```

Internally, it sets two `clientId` fields (inner and outer):

- The inner identifies the specific request within the payload and is used by cTrader to match that request with its corresponding response and any related events.
  - The outer (when used along with the inner) tracks the overall message flow between the client and host.
3. The host listens to requests:

```
window.addEventListener('message-to-host', function (event) {})
```

#### Note

All SDK methods for sending requests and receiving responses are available as reference **server messages**, while details on the repeated fields within requests and responses are provided as **models**.

### Responses

1. The host dispatches a response:

```
window.dispatchEvent(new MessageEvent('message-from-host', { data: {} })))
```

Each response includes the original `clientId` to link it with the request. The response is based on this structure:

```
payloadType: 2101
clientId: "<same-id-as-request>"
```

Real-time or multiple responses generate new events using:

```
payloadType: 2102
clientId: "<same-id-as-request>"
```

2. The client listens for the response:

```
window.addEventListener('message-from-host', function (event) {})
```

## Events and subscriptions

WebView plugins can subscribe to real-time data such as **quotes**, **deal updates**, **execution events** and more.

1. Subscribe to quote data by sending a request:

```
{
  "payloadType": 2100, // indicates a request
  "payload": {
    "payloadType": 601,
    "symbolId": ["EURUSD_ID"],
    "clientMsgId": "quote-sub-123"
  },
  "clientMsgId": "asd2-asd2" // outer message ID
}
```

2. The subscription triggers the host to send periodic responses:

```
{
  "payloadType": 2102, // indicates an event
  "payload": {
    "payloadType": 3, // mapped to QUOTE_EVENT
    "payload": {
      "ask": 85274,
      "bid": 85271,
      "depth": [],
      "high": 85340,
      "low": 84932,
      "sessionClose": 85133,
      "symbolId": 9,
      "tickbar": [],
      "timestamp": 1750162811236,
      "trendbar": []
    },
    "clientMsgId": "asd1-asd1" // inner message ID
  },
  "clientMsgId": "asd2-asd2" // outer message ID
}
```

## Deeplinks

A plugin can ask cTrader to open a **deeplink** (internal URL) by sending this request:

```
payloadType: 2103
clientMsgId: "<client-msg-id>"
```

The plugin then listens for a response:

```
payloadType: 2104
clientMsgId: "<same-id-as-request>"
```

WebView plugins can implement a **supported deeplink** as demonstrated below:

```
// Open EURUSD symbol overview
const openEURUSDInfo = useCallback(() => {
  const url = baseUrl ? `${baseUrl}symbols/EURUSD/info` : 'symbols/EURUSD/info';
  openUrl(url);
}, [baseUrl, openUrl]);
```

## Summary of messages

The table below summarises the protocol messages.

Type	Payload types	Initiation	Use cases
Lifecycle/handshake	2000 (registration event), 2001	Required to initialise plugin communication. Host sends a registration event, plugin responds with a confirmation event.	Channel setup, mounting, communication readiness.

	(confirmation event)		
Event (one-way)	2102	Server-initiated. Events sent without a prior client request. Host generates one <code>clientId</code> ; cTrader backend includes another one.	Market quotes, trade execution updates, price notifications.
Internal navigation request	2103 (OpenInternalUriReq), 2104 (OpenInternalUriRes)	Plugin-initiated request to navigate within the cTrader app. Host processes and confirms via paired response.	Opening overview of a symbol, opening order screen for a symbol.
Request/response	2100 (req), 2101 (res)	Initiated by plugin. Used for single-response operations. Both inner and outer <code>clientId</code> are reused in response.	Getting account, symbol or deal information.
Request/multi-event	2100 (req), then multiple 2102 (events)	One request triggers a stream of related events. Events reuse the original <code>clientId</code> (inner), and new event IDs are generated by the host.	Placing an order, subscribing to updates after a command.
Error response	2500 (PublicErrors)	Host or cTrader uses this to report any failure to process a request. Includes outer and inner <code>clientId</code> .	Invalid payload, unauthorised access, malformed SDK request.

### Error handling

#### Warning

Common reasons for errors include unknown or unregistered plugin, invalid request format and server-side validation failure. If something goes wrong, the host issues a structured error response:

payloadType: 2500

clientId: <original-id>

```
{
  "payloadType": 2500,
  "clientId": "<original-id>",
  "error": "InvalidPayload"
}
```

The SDK handles the error and surfaces it to the plugin creator.