

## SDK integration example

WebView plugins achieve greater functionality when they incorporate **SDK methods** for data retrieval, trading and other operations that benefit users.

### Note

The example in this article does not require any pipeline or setup as all dependencies are configured to load straight from **esm.sh**. The code can be saved as an `.html` file and then uploaded directly to a hosting service for quick deployment. Finally, the resulting website URL can be used to **build a WebView plugin**.

### Tip

Feed the example code and **SDK references** to an AI model, then guide the AI to edit the code to **create** and deliver your website idea for a WebView plugin.

The HTML code below is a self-contained example of a website for a WebView plugin. This web app connects to the trading host, exchanges data and executes trade operations in real time.

### Web app example

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8"/>
  <title>SDK integration example — single-file build-less</title>
  <meta name="viewport" content="width=device-width,initial-scale=1"/>

  <!-- Minimal styles -->
  <style>
    html,body{margin:0;height:100%;font-family:Arial,Helvetica,sans-serif}
    #root{height:100%;display:flex;justify-content:center;align-items:flex-start;
      background:#fff;color:#000;padding:10px;overflow:auto}
    button{padding:4px 8px;margin:2px}
    input,select{padding:2px;margin-left:4px}
    pre{margin:0}
  </style>
</head>

<body>
  <div id="root"></div>

  <script type="module">
    /***** CDN imports *****/
    import React, { useState, useEffect, useRef, useCallbck } from "https://esm.sh/react@19.1.0";
    import { createRoot } from "https://esm.sh/react-dom@19.1.0/client";

    /* SDK + helpers */
    import { createClientAdapter }
      from "https://esm.sh/@spotware-web-team/sdk-external-api@0.0.5";
    import {
      cancelOrder, closePosition, createNewOrder,
      executionEvent, getAccountGroupInformation, getAccountInformation,
      getDealList, getDynamicLeverage, getLightSymbolList, getSymbol,
      handleConfirmEvent, modifyOrder, modifyOrderProtection, quoteEvent,
      registerEvent, ServerInterfaces, subscribeQuotes, unsubscribeQuotes
    } from "https://esm.sh/@spotware-web-team/sdk@0.0.9";
    import { take, tap, catchError } from "https://esm.sh/rxjs@7.8.2";
    import { createLogger } from "https://esm.sh/@veksa/logger@1.0.6";

    /***** Example component *****/
    function Example(){
      /* --- connection / adapters --- */
      const [connected,setConnected] = useState(false);
      const adapter = useRef(null);

      /* --- UI state --- */
      const [logs,setLogs] = useState([]);
      const [symbolId, setSymbolId] = useState(1);
      const [quotesSymbolId, setQuotesSymbolId] = useState(1);
      const [leverageId, setLeverageId] = useState(1);
      const [newOrderSymbolId, setNewOrderSymbolId] = useState(1);
      const [newOrderVolume, setNewOrderVolume] = useState(100000);
      const [modifyOrderId, setModifyOrderId] = useState(1);
      const [modifyOrderVolume, setModifyOrderVolume]=useState(100000);
      const [modifyPrice, setModifyPrice] = useState(100);
      const [modifyPositionId, setModifyPositionId]=useState(1);
      const [modifyStopLoss, setModifyStopLoss] = useState(0.1);
```

```

const [modifyTakeProfit, setModifyTakeProfit]=useState(0.1);
const [cancelOrderId, setCancelOrderId] = useState(1);
const [closePositionId, setClosePositionId]=useState(1);
const [closePositionVolume,setClosePositionVolume]=useState(100000);

const log = m => setLogs(p=>[...p, typeof m==="string"?m:JSON.stringify(m,null,2)]);

/* ----- connect on mount ----- */
useEffect(()=>{
  const logger = createLogger(location.href.includes("showLogs"));
  adapter.current= createClientAdapter({logger});
  log("🔌 Connecting...");

  handleConfirmEvent(adapter.current,{}).pipe(take(1)).subscribe();

  registerEvent(adapter.current).pipe(
    take(1),
    tap(()=>{
      handleConfirmEvent(adapter.current,{}).pipe(take(1)).subscribe();
      setConnected(true); log("✅ Connected");
    }),
    catchError(()=>{ log("❌ Error host connection"); return []; })
  ).subscribe();
},[]);

/* ----- helper wrappers ----- */
const handleAccountInformation = ()=> getAccountInformation(adapter.current,{}).pipe(take(1), tap(log)).subscribe();
const
handleAccountGroupInformation=()=>getAccountGroupInformation(adapter.current,{}).pipe(take(1),tap(log)).subscribe();
const handleLightSymbolList = ()=> getLightSymbolList(adapter.current,{}).pipe(take(1),tap(log)).subscribe();
const handleSymbol = ()=> getSymbol(adapter.current,{symbolId:[symbolId]}).pipe(take(1),tap(log)).subscribe();
const handleSubscribeQuotes = ()=>
subscribeQuotes(adapter.current,{symbolId:[quotesSymbolId]}).pipe(take(1),tap(log)).subscribe();
const handleUnsubscribeQuotes = ()=>
unsubscribeQuotes(adapter.current,{symbolId:[quotesSymbolId]}).pipe(take(1),tap(log)).subscribe();
const handleDynamicLeverage = ()=>
getDynamicLeverage(adapter.current,{leverageId}).pipe(take(1),tap(log)).subscribe();
const handleDealList = ()=>
getDealList(adapter.current,{fromTimestamp:0,toTimestamp:Date.now()}).pipe(take(1),tap(log)).subscribe();
const handleCreateNewOrder = ()=> createNewOrder(adapter.current,{
  symbolId:newOrderSymbolId,
  orderType:ServerInterfaces.ProtoOrderType.MARKET,
  tradeSide:ServerInterfaces.ProtoTradeSide.BUY,
  volume:newOrderVolume,
}).pipe(tap(log)).subscribe();
const handleModifyOrder = ()=> modifyOrder(adapter.current,{
  orderId:modifyOrderId,
  limitPrice:modifyPrice,
  volume:modifyOrderVolume,
}).pipe(tap(log)).subscribe();
const handleModifyOrderProtection= ()=> modifyOrderProtection(adapter.current,{
  positionId:modifyPositionId,
  stopLoss:modifyStopLoss,
  takeProfit:modifyTakeProfit,
}).pipe(tap(log)).subscribe();
const handleCancelOrder = ()=> cancelOrder(adapter.current,{orderId:cancelOrderId}).pipe(tap(log)).subscribe();
const handleClosePosition = ()=> closePosition(adapter.current,{
  positionId:closePositionId,
  volume:closePositionVolume,
}).pipe(tap(log)).subscribe();

/* ----- stream quotes / executions ----- */
useEffect(()=>{
  if(!connected) return;
  quoteEvent(adapter.current).pipe(tap(log)).subscribe();
  executionEvent(adapter.current).pipe(tap(log)).subscribe();
},[connected]);

const e = React.createElement; // shorthand

/* first column of controls */
const controlBlock = e("div",{style:{
  display:"flex",flexDirection:"column",alignItems:"flex-start",
  gap:10,flexWrap:"wrap",marginBottom:10}},
[

```

```

e("button",{disabled:!connected,onClick:handleAccountInformation},"getAccountInformation"),
e("button",{disabled:!connected,onClick:handleAccountGroupInformation},"getAccountGroupInformation"),
e("button",{disabled:!connected,onClick:handleLightSymbolList},"getLightSymbolList"),

/* getSymbol */
e("div",null,[
  e("button",{disabled:!connected,onClick:handleSymbol},"getSymbol"),
  " symbolId:",
  e("input",{style:{width:30},value:symbolId,
    onChange:ev=>setSymbolId(+ev.target.value)})),
]),

/* subscribeQuotes */
e("div",null,[
  e("button",{disabled:!connected,onClick:handleSubscribeQuotes},"subscribeQuotes"),
  " symbolId:",
  e("input",{style:{width:30},value:quotesSymbolId,
    onChange:ev=>setQuotesSymbolId(+ev.target.value)})),
]),

/* unsubscribeQuotes */
e("div",null,[
  e("button",{disabled:!connected,onClick:handleUnsubscribeQuotes},"unsubscribeQuotes"),
  " symbolId:",
  e("input",{style:{width:30},value:quotesSymbolId,
    onChange:ev=>setQuotesSymbolId(+ev.target.value)})),
]),

/* getDynamicLeverage */
e("div",null,[
  e("button",{disabled:!connected,onClick:handleDynamicLeverage},"getDynamicLeverage"),
  " leverageId:",
  e("input",{style:{width:30},value:leverageId,
    onChange:ev=>setLeverageId(+ev.target.value)})),
]),

e("button",{disabled:!connected,onClick:handleDealList},"getDealList"),

/* createNewOrder */
e("div",null,[
  e("button",{disabled:!connected,onClick:handleCreateNewOrder},"createNewOrder"),
  " symbolId:",
  e("input",{style:{width:30},value:newOrderSymbolId,
    onChange:ev=>setNewOrderSymbolId(+ev.target.value)})),
  " volume:",
  e("input",{style:{width:60},value:newOrderVolume,
    onChange:ev=>setNewOrderVolume(+ev.target.value)})),
]),

/* modifyOrder */
e("div",null,[
  e("button",{disabled:!connected,onClick:handleModifyOrder},"modifyOrder"),
  " orderId:",
  e("input",{style:{width:30},value:modifyOrderId,
    onChange:ev=>setModifyOrderId(+ev.target.value)})),
  " volume:",
  e("input",{style:{width:60},value:modifyOrderVolume,
    onChange:ev=>setModifyOrderVolume(+ev.target.value)})),
  " modify price:",
  e("input",{style:{width:40},value:modifyPrice,
    onChange:ev=>setModifyPrice(+ev.target.value)})),
]),

/* modifyOrderProtection */
e("div",null,[
  e("button",{disabled:!connected,onClick:handleModifyOrderProtection},"modifyOrderProtection"),
  " positionId:",
  e("input",{style:{width:30},value:modifyPositionId,
    onChange:ev=>setModifyPositionId(+ev.target.value)})),
  " stop loss:",
  e("input",{style:{width:40},value:modifyStopLoss,
    onChange:ev=>setModifyStopLoss(+ev.target.value)})),
  " takeProfit:",
  e("input",{style:{width:40},value:modifyTakeProfit,
    onChange:ev=>setModifyTakeProfit(+ev.target.value)})),
]),

```

```

    )},

    /* cancelOrder */
    e("div", null, [
      e("button", {disabled:!connected, onClick:handleCancelOrder}, "cancelOrder"),
      " orderId:",
      e("input", {style:{width:30}, value:cancelOrderId,
        onChange:ev=>setCancelOrderId(+ev.target.value)}),
    ]),

    /* closePosition */
    e("div", null, [
      e("button", {disabled:!connected, onClick:handleClosePosition}, "closePosition"),
      " positionId:",
      e("input", {style:{width:30}, value:closePositionId,
        onChange:ev=>setClosePositionId(+ev.target.value)}),
      " volume:",
      e("input", {style:{width:60}, value:closePositionVolume,
        onChange:ev=>setClosePositionVolume(+ev.target.value)}),
    ]),
  ]);

  /* log viewer */
  const logPane = e("div", {style:{
    height:"100%", overflowY:"scroll", minHeight:200, background:"#eee",
    padding:6, borderRadius:4}},
    logs.map((l,i)=>e("pre",{key:i, style:{marginBottom:10}},l)));

  return e("div", {style:{display:"flex", flexDirection:"column",
    width:"100%", padding:10, overflow:"auto"}}, [
    controlBlock,
    logPane
  ]);
}

/***** Mount to DOM *****/
createRoot(document.getElementById("root")).render(React.createElement(Example));
</script>
</body>
</html>

```

This table breaks down the typical and important operations in the example:

Operation	Purpose	Instance
Handshake	Opens the WebSocket and completes the confirm → register → confirm sequence.	The first <code>useEffect</code>
Ongoing synchronisation	Streams market data and execution events so that the UI stays current.	The second <code>useEffect</code>
Mounting	Inserts the (React) UI into the page or WebView container.	The <code>createRoot(...).render(...)</code> call at the bottom of the code
SDK actions	Invokes SDK methods, such as <code>getAccountInformation</code> , <code>createNewOrder</code> , <code>modifyOrder</code> , <code>closePosition</code> , etc., in response to user interaction.	Individual button-handler functions inside the <code>Example</code> component

## Handshake

When the plugin loads, it establishes a connection with the cTrader host by creating a `clientAdapter` instance. This adapter enables two-way communication between the WebView plugin and the platform.

```
// The first useEffect, which runs only on initial mounting
useEffect(() => {
  // 1. Open the WebSocket connection
  const logger = createLogger(location.href.includes("showLogs"));
  adapter.current = createClientAdapter({ logger });
  log("🔌 Connecting...");

  // 2. First handshake message ("confirm")
  handleConfirmEvent(adapter.current, {})
    .pipe(take(1))
    .subscribe();

  // 3. Wait until the host registers the client
  registerEvent(adapter.current)
    .pipe(
      take(1),
      tap(() => {
        // 4. Second confirmation required by the protocol
        handleConfirmEvent(adapter.current, {})
          .pipe(take(1))
          .subscribe();

        // 5. Fully in-sync. Unlock the UI
        setConnected(true);
        log("✅ Connected");
      })
    )
    .catchError(err => {
      log("❌ Error host connection");
      log(err);
      return [];
    })
  )
  .subscribe();
}, []); // Empty deps → runs once
```

## Ongoing synchronisation

Once the handshake is complete and the plugin is fully connected, it begins synchronising with the host by subscribing to two continuous event streams: live quotes and trade execution updates. This ensures the plugin receives up-to-date pricing and order activity from the trading environment in real time.

```
// The second useEffect, which depends on the `connected` flag
useEffect(() => {
  if (!connected) return; // do not subscribe until handshake finishes

  // 1. Subscribe to live quotes (price updates)
  quoteEvent(adapter.current)
    .pipe(tap(log))
    .subscribe();

  // 2. Subscribe to trade executions (order fills, cancels, rejections)
  executionEvent(adapter.current)
    .pipe(tap(log))
    .subscribe();
}, [connected]); // rerun this block only once `connected` becomes true
```

## Mounting

After defining the `Example` component, the plugin renders its user interface into the DOM using React `createRoot` API. This step mounts the plugin visually and functionally within the host container, enabling user interaction.

```
// At the bottom. This runs once at load
createRoot(document.getElementById('root'))
  .render(React.createElement(Example));
```

This mounting step completes the setup phase by placing the plugin into the visible interface and allows the logic defined inside `Example()` to begin running, including connection handling, SDK bindings and event subscriptions.

## SDK actions

The SDK provides convenient wrappers for requesting trading-related information, and SDK calls in the code share the same pattern:

```
sdkMethod(adapter.current, params)
  .pipe(take(1), tap(log), catchError(handleError))
  .subscribe();
```

Each of these functions is attached to a button that is only enabled once the plugin is connected. Each button click calls a specific SDK method, such as fetching account data or creating an order.

### Get account information

```
getAccountInformation(adapter, {}).pipe(take(1), tap(log)).subscribe();
```

### Create new market order

```
createNewOrder(adapter, {
  symbolId: 1,
  orderType: ServerInterfaces.ProtoOrderType.MARKET,
  tradeSide: ServerInterfaces.ProtoTradeSide.BUY,
  volume: 100000,
}).pipe(tap(log)).subscribe();
```

### Get list of tradeable symbols

```
getLightSymbolList(adapter, {}).pipe(take(1), tap(log)).subscribe();
```

### Subscribe and unsubscribe to quotes

```
subscribeQuotes(adapter, { symbolId: [1] }).pipe(take(1), tap(log)).subscribe();
unsubscribeQuotes(adapter, { symbolId: [1] }).pipe(take(1), tap(log)).subscribe();
```

Additionally, you can listen to price updates and receive trade execution results using:

```
quoteEvent(adapter.current).pipe(tap(log)).subscribe();
executionEvent(adapter.current).pipe(tap(log)).subscribe();
```

## Logging and debugging output (optional)

All SDK responses and events are logged to a side panel in real time using:

```
const log = m => setLogs(p => [...p, typeof m === "string" ? m : JSON.stringify(m, null, 2)]);
```

The output area is rendered using:

```
logs.map((l, i) => e("pre", { key: i }, l))
```

These logs allow the inspection of the JSON payloads and help to confirm whether an operation succeeded or failed with details.