WebSocket API

This article explains how to use the WebSocket API provided by the Fusion SDK to receive real-time updates about order events and to query active orders. The WebSocket API enables developers to build applications that can respond instantly to order creation, filling, and cancellation events.

Real-world example

```
import { WebSocketApi } from "@1inch/solana-fusion-sdk";

const wsSdk = WebSocketApi.fromConfig({
  url: "wss://api.1inch.dev/fusion/ws",
  authKey: "your-auth-key",
});

wsSdk.order.onOrder((data) => {
  console.log("received order event", data);
});
```

Creation

With constructor

```
import { WebSocketApi } from "@1inch/solana-fusion-sdk";

const ws = WebSocketApi.fromConfig({
  url: "wss://api.1inch.dev/fusion/ws",
  authKey: "your-auth-key",
});
```

With custom provider

Alternatively, you can use a custom provider for WebSocket (by default the ws library is used).

```
import { WsProviderConnector, WebSocketApi } from "@1inch/solana-fusion-sdk";

class MyCustomProvider extends WsProviderConnector {
  // ... user implementation
}

const url = "wss://api.1inch.dev/fusion/ws/v1.0/501";
const provider = new MyCustomProvider({ url });

const wsSdk = WebSocketApi.fromProvider(provider);
```

Lazy initialization

By default, creating an instance of `WebSocketApi` opens a WebSocket connection immediately. To delay this until explicitly required, pass `lazyInit: true` in the constructor. Then, use the `.init()` method to manually start the connection.

```
import { WebSocketApi } from "@1inch/solana-fusion-sdk";

const ws = WebSocketApi.fromConfig({
  url: "wss://api.1inch.dev/fusion/ws",
  lazyInit: true,
});

ws.init();
```

Base methods

on

Description: Subscribe to any event

Arguments:

- [0] event: WebSocketEvent
- [1] cb: Function

Example

```
import {WebSocketEvent} from '@1inch/solana-fusion-sdk'

const ws = ...

ws.on(WebSocketEvent.Error, console.error)

ws.on(WebSocketEvent.Open, function open() {
    ws.send('something')
})

ws.on(WebSocketEvent.Message, function message(data) {
    console.log('received: %s', data)
})
```

off

Description: Unsubscribe from any event.

Arguments:

- [0] event: WebSocketEvent
- [1] cb: Function

Example

```
import {WebSocketEvent} from '@1inch/solana-fusion-sdk'

const ws = ...

function message(data) {
    console.log('received: %s', data)
}

ws.on(WebSocketEvent.Message, message)

ws.off(WebSocketEvent.Message, message)
```

onOpen

Description: Subscribe to an open event.

Arguments:

- [0] cb: Function

Example

```
const ws = ...
ws.onOpen(() => {
    console.log('connection is opened')
})
```

send

Description: Send event to backend.

Arguments:

- [0] message: any message which can be serialized with JSON.stringify

Example

```
const ws = ...
```

```
ws.send('my message')
```

close

Description: Close connection.

Example

```
const ws = ...

ws.close()
```

onMessage

Description: Subscribe to message event.

Arguments:

- [0] cb: (data: any) => void

Example

```
const ws = ...


ws.onMessage((data) => {
    console.log('message received', data)
})
```

onClose

Description: Subscribe to close event.

Example

```
const ws = ...


ws.onClose(() => {
    console.log('connection is closed')
})
```

onError

Description: Subscribe to error event.

Arguments:

- [0] cb: (error: any) => void

Example

```
const ws = ...


ws.onError((error) => {
    console.log('error is received', error)
})
```

Order namespace methods

onOrder

Description: Subscribe to order event.

Arguments:

- [0] cb: (data: OrderEventType) => void

Example

```
import {EventType} from '@1inch/solana-fusion-sdk'

const ws = ...

ws.order.onOrder((data) => {
    if (data.event === EventType.Create) {
        // do something
    }

    if (data.event === EventType.Fill) {
        // do something
    }
})
```

onOrderCreated

Description: Subscribe to order_created events.

Arguments:

- [0] cb: (data: OrderCreatedEvent) => void

Example

```
const ws = ...

ws.order.onOrderCreated((data) => {
    // do something
})
```

onOrderFilled

Description: Subscribe to order_filled events.

Arguments:

- [0] cb: (data: OrderFilledEvent) => void

Example

```
const ws = ...

ws.order.onOrderFilled((data) => {
    // do something
})
```

onOrderCancelled

Description: Subscribe to order_cancelled events.

Arguments:

- [0] cb: (data: OrderCancelledEvent) => void

Example

```
const ws = ...

ws.order.onOrderCancelled((data) => {
    // do something
})
```

RPC namespace methods

onPong

Description: Subscribe to ping response.

Arguments:

- [0] cb: (data: string) => void

Example

```
const ws = ...

ws.rpc.onPong(() => {
    // do something
})
```

ping

Description: Ping healthcheck.

Example

```
const ws = ...
ws.rpc.ping()
```

getAllowedMethods

Description: Get the list of allowed methods.

Example

```
const ws = ...
ws.rpc.getAllowedMethods()
```

onGetAllowedMethods

Description: Subscribe to get allowed methods response.

Arguments:

- [0] cb: (data: RpcMethod[]) => void

Example

```
const ws = ...

ws.rpc.onGetAllowedMethods((data) => {
    // do something
})
```

getActiveOrders

Description: Get the list of active orders.

Example

```
const ws = ...

ws.rpc.getActiveOrders()
```

onGetActiveOrders

Description: Subscribe to get active orders events.

Arguments:

- [0] cb: (data: PaginationOutput) => void

Example

```
const ws = ...

ws.rpc.onGetActiveOrders((data) => {
    // do something
})
```

Types

OrderEventType

```
import { OrderType } from "./types";

export type Event<K extends string, T> = {
  event: K;
  result: T;
};

export type OrderEventType =
  | OrderCreatedEvent
  | OrderFilledEvent
  | OrderCancelledEvent;

export enum EventType {
  Create = "create",
  Fill = "fill",
  Cancel = "cancel",
}

export type CreateOrderEventPayload = {
  transactionSignature: string;
  slotNumber: number;
  blockTime: number;
  action: string;
  commitment: string;
  orderHash: string;
  maker: string;
  order: Jsonify<FusionOrder>;
  filledAuctionTakerAmount: string;
  filledMakerAmount: string;
};

export type FillOrderEventPayload = {
  transactionSignature: string;
  slotNumber: number;
  blockTime: number;
  action: string;
  commitment: string;
  orderHash: string;
  maker: string;
  resolver: string;
  filledAuctionTakerAmount: string;
  filledMakerAmount: string;
};

export type CancelOrderEventPayload = {
  transactionSignature: string;
  slotNumber: number;
  blockTime: number;
  action: string;
  commitment: string;
  orderHash: string;
  maker: string;
  filledAuctionTakerAmount: string;
  filledMakerAmount: string;
};

export type OrderCreatedEvent = Event<
  EventType.Create,
  CreateOrderEventPayload
>;

export type OrderFilledEvent = Event<EventType.Fill, FillOrderEventPayload>;

export type OrderCancelledEvent = Event<
  EventType.Cancel,
  CancelOrderEventPayload
>;
```

RpcMethod

```
export enum RpcMethod {
  GetAllowedMethods = "getAllowedMethods",
  Ping = "ping",
  GetActiveOrders = "getActiveOrders",
}
```

Previous

‹ Filling and canceling Solana Fusion order

Next

FusionOrder module ›