Websockets

Your Game Code - Adding Custom Elements · https://namazustudios.com/docs/custom-code/websockets/

How to build Websocket APIs in Elements

Elements 3.0 offers Standards Compliant Websockets built in using the Jakarta Websocket API 2.1. Websockets are useful in creating high performance bi-directional communication between client and server code. Generally speaking, Websockets are considerably faster than HTTP requests for authoritative code and work well with practically all clients including Web, Unity3d, Unreal, .NET and many other connected services.

Steps to Defining a Websocket Element

To use the Jakarta RS in your own Element, you must perform the following steps:

- Define the Element by annotating the `package-info` type in your code.
- Add all compiled classes and jars into the Element package structure.
- Annotate each Websocket endpoint with the `ServerEndpoint` annotation.

Complete Example

The following example walks through the necessary files to define a simple Websocket echo server.

Step1: Define the Element

{% code title="package-info.java" %}

```
@ElementDefinition
package dev.getelements.elements.sdk.test.element.ws;

import dev.getelements.elements.sdk.annotation.ElementDefinition;
```

{% endcode %}

Step 2: Define the Element

{% code title="EchoEndpoint.java" %}

```
package dev.getelements.elements.sdk.test.element.ws;

import jakarta.websocket.*;
import jakarta.websocket.server.ServerEndpoint;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@ServerEndpoint("/echo")
public class EchoEndpoint {

    private static final Logger logger = LoggerFactory.getLogger(EchoEndpoint.class);

    @OnOpen
    public void onOpen(final Session session) {
        logger.info("Opened {}", session.getId());
    }

    @OnMessage
    public String onMessage(final Session session, final String message) {
        logger.info("Received {}. Echoing.", message);
        return message;
    }

    @OnClose
    public void onClose(final Session session, final CloseReason closeReason) {
        logger.info("Closed {} - {}", session.getId(), closeReason);
    }

}
```

{% endcode %}

Step 3: Ensure All Dependencies are Included

{% code title="pom.xml" %}

```
<?xml version="1.0"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-
4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>dev.getelements.elements</groupId>
        <artifactId>eci-elements</artifactId>
        <version>2.2.0-SNAPSHOT</version>
    </parent>

    <artifactId>sdk-test-element-ws</artifactId>
    <version>2.2.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>dev.getelements.elements</groupId>
            <artifactId>sdk</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>jakarta.websocket</groupId>
            <artifactId>jakarta.websocket-api</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>jakarta.websocket</groupId>
            <artifactId>jakarta.websocket-client-api</artifactId>
            <scope>provided</scope>
        </dependency>
    </dependencies>

</project>
```

{% endcode %}