HUMAX

IxWebSocketServer SDK
User Guide

Eng Ver 2.0

Ver	Date	Change	Author
1.0	2017 08/29	Draft Version Release	GyoungChul Jung
1.1	2017 09/19	JsonRpcParser update	GyoungChul Jung
1.2	2018 01/05	Sdk version 1.1.0 has been applied.	GyoungChul Jung
2.0	2018 03/26	A new server supporting the http protocol was applied.	Kwonsik Kim

Contents

1 Introduction 6

2 SDK Package Structure 7

2.1 Directory Structure 7

3 How to run WebSocket Server 9

3.1 IxWSS Server API 9

3.2 How to implement simple server 10

3.3 ixwss.conf 11

4 How to run WebSocket Client 12

4.1 IxWSS Server API 12

4.2 How to implement simple client 13

5 IXPlugin 15

5.1 What is IXPlugin? 15

5.2 How does it work? 15

5.3 How to load plugin? 15

5.4 Initialization and destruction 16

5.5 Data processing and sending data 17

5.6 callback functions 17

5.7 ixconfig.conf 17

5.8 Simple example 18

6 IXPlugin with JSON-RPC 18

6.1 JsonObject 19

6.2 JsonRpcPlugin 20

6.3 JsonRpcProcessor 20

6.4 JsonRpcParser 21
6.5 ObjectFactory 22
6.6 ObjectInstance 23
6.7 EventTarget 23
6.8 Integration layer 24
6.8.1 Naming conventions 24
6.8.2 Return codes 24
6.8.3 Data objects 24
6.8.4 Event handling 25
6.8.5 Error handling 25
6.9 Plugin Sequence 25
6.9.1 Plugin loading 25
6.9.2 Client request process 26
6.9.3 Plugin Event Process 27

Picture Contents

Figure 1. SDK directory structure 7
Figure 2. IXPlugin Load 16
Figure 3. Plugin Structure on JSON-RPC 18
Figure 4. JSON-RPC Plugin Class Diagram 19
Figure 5. Sequence of plugin loading 25
Figure 6. Sequence of client request processing 26
Figure 7. Sequence of plugin event processing 27

1 Introduction

IX of IXWSS stands for Interface Extension, and WSS stands for WebSocket Server. In other words, IXWSS can be understood as a component that can easily extend the interface between client and server based on WebSocket/HTTP communication protocol. In a client / server architecture, a communication protocol is basically a standard method, but various data formats may be required for data to be transmitted. IXWSS provides an environment that can be developed using individual plugins so that users can directly process data in various formats.

This document is intended for the user who use the IXWSS SDK. It describe the how to build a plugin-based WebSocket Server using the IXWSSS SDK.

2 SDK Package Structure

2.1 Directory Structure

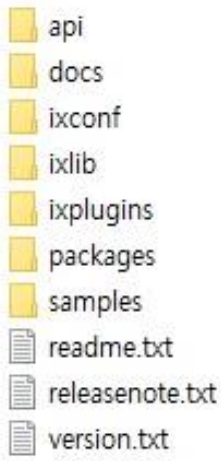


Figure 1. SDK directory structure

- ✓ API

  It provides the contents of IXWSS interface as doxygen documents.

- ✓ docs

  It provides the User Guide document of IXWSS.

- ✓ ixconf

  It provides an example of the necessary configuration files when running IXWSS.

  The files listed below must be present, and the user should create a configuration files with reference to them.

  - ■ ixconfig.conf

    The The information about WebSocketServer & Plugin's setting.

- ✓ ixlib

  It provides the library and header of IXWSS.

  - ■ Library : libixs.so
  - ■ Header :
    - ✓ IxServer.h
    - ✓ IxWebSocketServer.h
    - ✓ IxWebSocketClient.h

✓ ixplugins

It provides the sample plug-in and it's source codes.

The user can refer to the sample plug-in and create the new plug-in the user want.

- Jsonrpc plugin

  The basic sample that processes json data and passes the results and events to the web client.

- Simple plugin

  The basic sample that can verify that the data is passed to the plug-in via the web socket.

✓ packages

It provides the necessary open libraries when running the IXWSS.

- libwebsockets

  This is for the running the IXWSS. (version : 2.0.0)

- libjansson

  This is for the sample plugin named jsonrpc. (version : 4.9.0)

✓ samples

It provides the sample server, client and client web program that run the IXWSS.

It provides a sample client web that transfers data to the sample server program via a web socket. The user can refer to this samples and create the new program.

✓ readme.txt

It provides the readme file.

✓ releasenote.txt

It provides the history of versions.

✓ version.txt

It provides the number of version.

3 How to run WebSocket Server

3.1 IxWSS Server API

The list of IXWSS server interfaces are as follows.

■ Member Functions

함수명	IxWebSocketServer
매개변수	-
리턴값	-
기능정의	Constructor for IXWebSocketServer.

함수명	~IxWebSocketServer
매개변수	-
리턴값	-
기능정의	Destructor for IXWebSocketServer.

함수명	initialize
매개변수	const char *conf_path
리턴값	bool
기능정의	Initialize the server with the configuration file.

함수명	start
매개변수	void
리턴값	bool
기능정의	Start the server.

함수명	stop
매개변수	void

리턴값	bool
기능정의	Stop the server.

ps : See the doxygen document for the detail information on the IXWSS interface.

3.2 How to implement simple server

IXWSS is distributed as a shared library and runs independently of the plugin. How to start IXWSS is following.

1) Two library are required to configure the IXWSS server.
   - ✓ libwebsocket.so
   - ✓ libixs.so
2) To start server, create a server object.
3) When creating the server object, the name of the file including the folder path where the configure file needed for the server operation exists should be passed as a parameter.
4) The IXWSS server sets the server by reading the configuration file from the forwarded path.

The example code is as follows.

```
#include "IxWebSocketServer.h"
#include <iostream>

using namespace std;
using namespace ixs;

int main( int argc, char** argv)
{
    IxWebSocketServer *server = new IxWebSocketServer();
    if ( !server ) return -1;

    if ( !server->initialize("../ixconf/ixconfig.conf") )
    {
        cout<<"initialize fail !!"<<endl;
        return -1;
    }
    if ( !server->start() )
    {
```

```
        cout<<"start fail !!"<<endl;
        return -1;
    }

    for ( ;; )
    {

    }

    if ( !server->stop() )
    {
        cout<<"stop fail !!"<<endl;
        delete server;
        return -1;
    }

    delete server;
    return 0;
}
```

3.3 ixwss.conf

Set the protocol to be connected with the port and protocol that Web Socket Server will allow.

✓ plugin_map ={port}:{protocol}:{sub-protocol}:{plugin.so}

The example is as follows.

```
# IXServer check this file to load plugin when the client request
#
# @ field : plugin_path
# description :
#    set the path of ixplugin library full path
# example >
#    plugin_path=/usr/browser/unittest/ixplugins/bin
#
# @ field : plugin_map
# description :
#    set the port numerber and prototocol and sub-protocol and ixplugin library file
#    that ixs has to load when the web client request. if you don't want to use sub-protocol
#    about some port , please refer to port 9997 in the example below
# example >
# plugin_path=../ixplugins/bin
# plugin_map=9999:http::libixplugin_simple.so
# plugin_map=9999:ws:jsonrpc:libixplugin_jsonrpc.so
# plugin_map=9998:http::libixplugin_jsonrpc.so
# plugin_map=9997:ws::libixplugin_simple.so
#

plugin_path=../ixplugins/bin
plugin_map=9999:http::libixplugin_simple.so
plugin_map=9999:ws:jsonrpc:libixplugin_jsonrpc.so
plugin_map=9998:http::libixplugin_jsonrpc.so
plugin_map=9997:ws::libixplugin_simple.so
```

4 How to run WebSocket Client

4.1 IxWSS Server API

The list of IXWSS Client interface are as follows.

■ Member Functions

Function	IXWebSocketClient
Parameter	const char *name, const char *address, int port, const char *protocol
Return	-
Description	The constructor of IXWebSocketClient. name : Host name address : Ip address port : The number of port protocol : The name of sub-protocol

Function	~IXWebSocketClient
Parameter	-
Return	-
Description	The destructor of IXWebSocketClient.

Function	connect
Parameter	void
Return	bool
Description	Connect with the server.

Function	disconnect
Parameter	void
Return	bool
Description	Disconnect with the server

Function	send
Parameter	const char *data, int dlen
Return	bool
Description	Send data to the server.

Function	addEventListener
Parameter	IXWebSocketClientListener * callback
Return	bool
Description	Register the listener

Function	removeEventListener
Parameter	Void
Return	Bool
Description	Unregister the listener

ps : See the doxygen document for the detail information on the IXWSS interface.

4.2 How to implement simple client

How to start IXWSS client is following.

1) Two library are required to configure the IXWSS server.
   - ✓ libwebsocket.so
   - ✓ libixwss.so
2) To start client, create a client object.
3) When creating the client object, the host name, Ip address, port number and the name of protocol should be passed as a parameter.
4) When calling the function connect(), the client will being connected with the server.

The example code is as follows.

```
#include "IXWebSocketClient.h"
#include <iostream>
#include <string>
```

```
#include <signal.h>

using namespace std;
using namespace ixwss;

class ReceiveData : public IXWebSocketClientListener
{
public:
    ReceiveData() { }
    virtual ~ReceiveData() { }
    virtual int onReceiveData( unsigned char *buffer, unsigned int length )
    {
        cout << "Receive Data: " << buffer << endl;
        return 0;
    }
};

int main( int argc, char** argv)
{
    if ( argc != 3 ) return 0;

    int port;
    string protocol;
    string str(" Some data format !! ");

    port = atoi(argv[1]);
    protocol.assign(argv[2]);

    cout << "Received the value -port: " << port << ", protocol: " << protocol << endl;
    IXWebSocketClient *client = new
IXWebSocketClient("localhost:9000","localhost",port,protocol.c_str());
    ReceiveData *listener = new ReceiveData();

    client->addEventListener(listener);
    client->connect();

    for ( ; ; )
    {
        char c = getchar();
        switch ( c )
        {
            case 'q' :
                client->disconnect(); break;
            case 'r' :
                client->connect(); break;
            case 's' :
```

```
                client->send(str.c_str(),str.size()); break;
            default :
                break;
        }
    }

    client->removeEventListener();
    if ( client ) delete client;
    if ( listener ) delete listener;
    return 0;
}
```

5 IXPlugin

5.1 What is IXPlugin?

IXWSS is designed to process the request of the client after loading plugin through PluginConnector designed for plugin loading of PluginManger. This chapter describes how to make a plugin and how to load the plugin from the PluginConnector.

	If the plugin fails to load because it does not exist, it will send a plugin loading failure message to IXWSS.

5.2 How does it work?

In case of IXWSS, home directory exists by default and the following files exist under the directory.

- ✓ ixplugin.ini : Shared object mapping table to be loaded for port number, protocol and sub-protocol
- ✓ {xxx}.so : Plugin (share object) created by user

5.3 How to load plugin?

*int ixserver_plugin_capability_function (ixserver_plugins_callbacks *cb,*
*ixserver_plugins_plugin_handler **ret)*

IXWSS calls the above function when the plugin is loaded.
The plugin creator should implement the following two things.

1) Keep the cb parameter received by ixserver_plugins_callbacks internally. This callback parameter is used to pass a response or event from the plugin to the client.

2) ixserver_plugins_plugin_handler assigns each function handler that it requires through the return parameter. Each return parameter is called when IXWSS requires the plugin to operate.

This part is a basic mutual promise for making a plugin, and preparation for plugin operation is completed through this process.

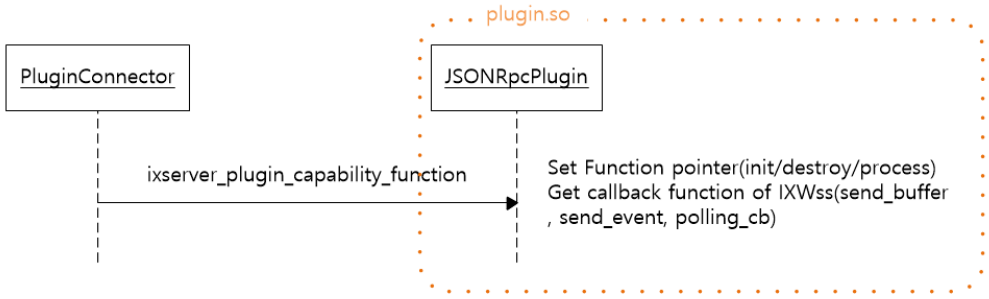


Figure 2. IXPlugin Load

```
static ixserver_plugins_plugin_handler s_plugin_handler;
static ixserver_plugins_callbacks* s_wss_cb = NULL;

int ixserver_plugin_capability( ixserver_plugins_callbacks *cb, ixserver_plugins_plugin_handler **ret )
{
    s_plugin_handler.process = &json_rpc_plugin_process;
    s_plugin_handler.initialize = &json_rpc_plugin_initialize;
    s_plugin_handler.destroy = &json_rpc_plugin_destroy;
    *ret = &s_plugin_handler;
    s_wss_cb = cb;
}
```

5.4 Initialization and destruction

After the plugin is loaded and the necessary information exchange is completed through ixserver_plugin_capability, the initialize function of the plugin handler is called to initialize the plugin. The plug-in developer can do the basic initialization of the plug-in here.

- ✓ *ixserver_plugins_plugin_handler_initialize*

When using the plugin, it performs a task to cleanup the internal resources of the plugin such as releasing resources before closing the shared object. To do this, IXWSS calls the destroy function before the plugin terminates.

- ✓ *ixserver_plugins_plugin_handler_destroy*

5.5 Data processing and sending data

A process handler is a callback function that receives data passed from a client. The plug-in developer processes the received data in the function and obtains the result.

- ✓ *ixserver_plugins_plugin_handler_process*

Call the this IXWSS callback function to pass the results from the process handler to the client.

- ✓ *ixserver_plugins_callback_send_buffer*

Call the this IXWSS callback function to pass the events from the process handler to the client.

- ✓ *ixserver_plugins_callback_send_event*

5.6 callback functions

In order to be able to transmit events generated by the plugin, a callback function that can poll events generated by the plugin should be registered in the PluginAdapter.
When the callback function is registered, the callback function is periodically called to transmit the generated event.

- ✓ *ixserver_plugins_polling_callback*

```
static int json_rpc_plugin_setPollingCallback( ixserver_plugins_object *obj )
{
    s_wss_cb->set_polling_cb( obj, json_rpc_plugin_pollingcallback );
    s_polling_callback_initialized = true;
}
```

5.7 ixconfig.conf

```
# IXServer check this file to load plugin when the client request
#
# @ field : plugin_path
# description :
#    set the path of ixplugin library full path
# example >
#    plugin_path=/usr/browser/unittest/ixplugins/bin
#
# @ field : plugin_map
# description :
#    set the port numerber and prototocol and sub-protocol and ixplugin library file
#    that ixs has to load when the web client request. if you don't want to use sub-protocol
#    about some port , please refer to port 9997 in the example below
# example >
# plugin_path=../ixplugins/bin
# plugin_map=9999:http::libixplugin_simple.so
# plugin_map=9999:ws:jsonrpc:libixplugin_jsonrpc.so
# plugin_map=9998:http::libixplugin_jsonrpc.so
# plugin_map=9997:ws::libixplugin_simple.so
#
```

```
plugin_path=../ixplugins/bin
plugin_map=9999:http::libixplugin_simple.so
plugin_map=9999:ws:jsonrpc:libixplugin_jsonrpc.so
plugin_map=9998:http::libixplugin_jsonrpc.so
plugin_map=9997:ws::libixplugin_simple.so
plugin_path=../ixplugins/bin
map_port_protocol_plugin=9998:jsonrpc:libixplugin_jsonrpc.so
map_port_protocol_plugin=9999:simple:libixplugin_simple.so
```

The process of finding a plug-in is very simple. If the WebSocket Server is listening on a specific port and a client is connected to the port, the Server will create a Session and pass the necessary plugin path and the callback to receive the event before delivering the data. It finds a shared object to be loaded in the received plugin path, loads it, registers an event callback, and communicates with the client.

5.8 Simple example

The simple example codes are existed in /ixplugins/simple folder.

6 IXPlugin with JSON-RPC

IXWSS can easily extend the interface between client and server by plugin method. IXPlugin is an environment provided for developing plugins directly from client. Various plugins can be made according to requirements. This chapter describes the pre-made JSON-RPC Plugin for customers to refer to when creating various plugins.

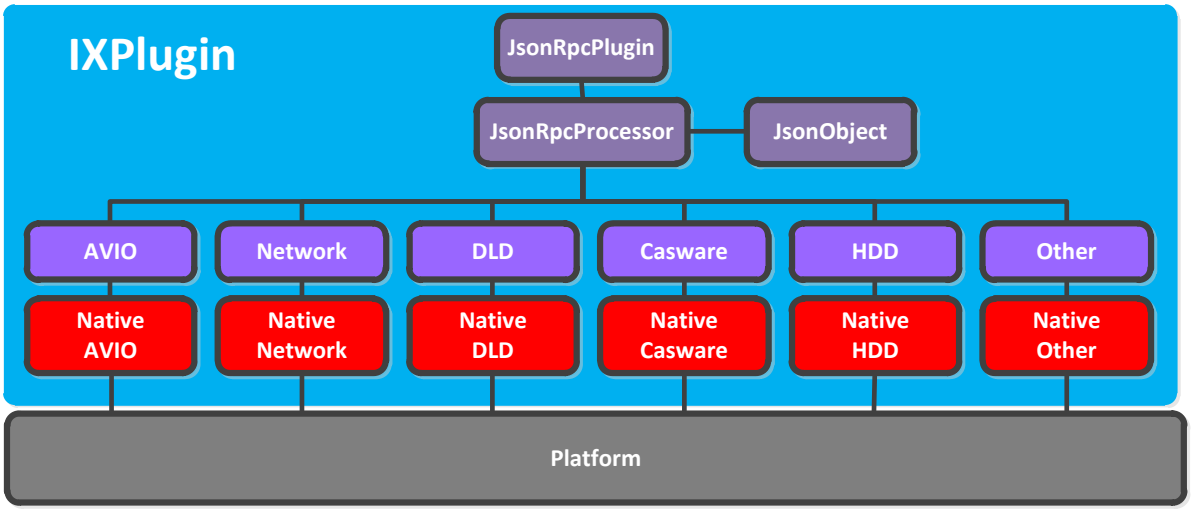


Figure 3. Plugin Structure on JSON-RPC

IXPlugin with JSON-RPC can be represented by the above block. The red highlight is the integration

layer to access platform resources.

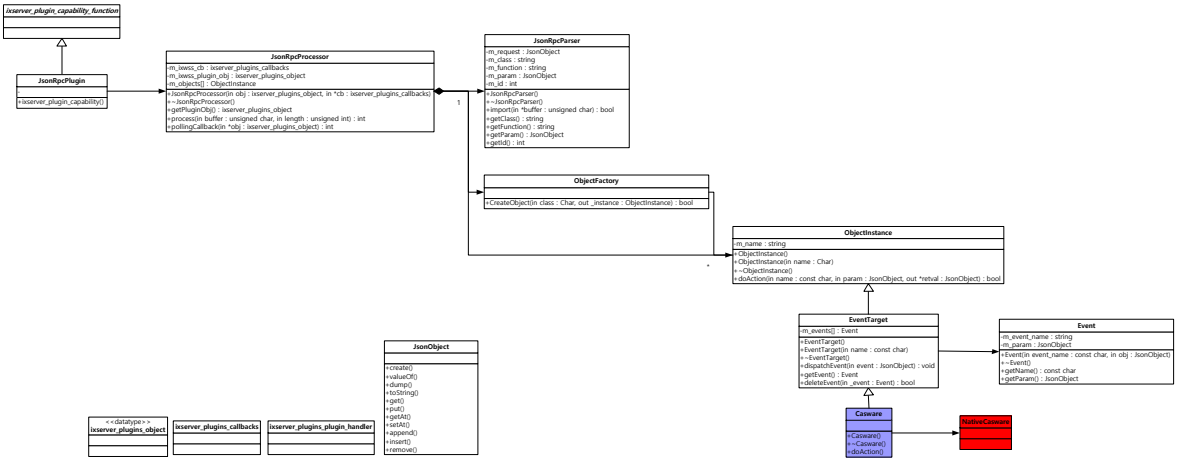


Figure 4. JSON-RPC Plugin Class Diagram

IXPlugin with JSON-RPC consists of the above class modules.

When the plugin is loaded, the ixserver_plugin_Capability () function of JsonRpcPlugin is called to exchange necessary information between the server and the plugin. There is a JsonRpcProcessor class that handles client request and platform events. The json data is managed in the JsonObject class. JsonRpcParser is used to parse json data. The ObjectFactory creates an object to handle client requests. All the generated objects inherit the objectInstance class. Objects that require event processing can manage events by inheriting the EventTarget class.

The classes are described in Sections 6.1 through 6.7.

6.1 JsonObject

JsonObject
+create() +valueOf() +dump() +toString() +get() +put() +getAt() +setAt() +append() +insert() +remove()

It is a data class to manage JSON data.

6.2 JsonRpcPlugin

JsonRpcPlugin
-
+ixserver_plugin_capability()

It is the main class of JSON-RPC Plugin and is responsible for communication with IXWSS. When IXWSS loads this plugin, it calls ixserver_plugin_capability () first.

■ Member Function

Function	ixserver_plugin_capability
Parameter	ixserver_plugins_callbacks *cb, ixserver_plugins_plugin_handler **ret
Return	0/1
Description	When IXWSS loads the plugin, the first function to be called to exchange handlerscb->set_polling_cb : Register polling callback function cb->send_buffer : Pass response data to client cb->send_event : Pass event data to client. ret->initialize : Register plugin initialization function ret->destroy : Register plugin shutdown function ret->process : Register plugin processing function

6.3 JsonRpcProcessor

JsonRpcProcessor
-m_ixwss_cb : ixserver_plugins_callbacks -m_ixwss_plugin_obj : ixserver_plugins_object -m_objects[] : ObjectInstance
+JsonRpcProcessor(in obj : ixserver_plugins_object, in *cb : ixserver_plugins_callbacks) +~JsonRpcProcessor() +getPluginObj() : ixserver_plugins_object +process(in buffer : unsigned char, in length : unsigned int) : int +pollingCallback(in *obj : ixserver_plugins_object) : int

This class handles both the client's request to be sent to the plugin and the response, the event to be delivered to the client.

■ Member Function

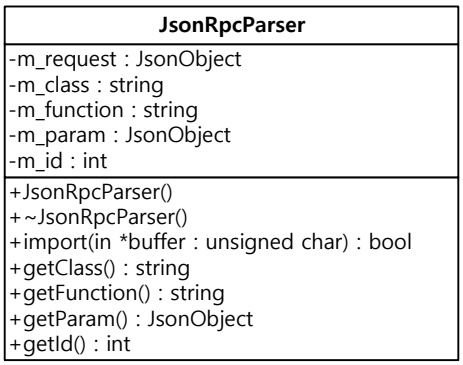
Function	getPluginObj
Return	ixserver_plugin_object

description	Returns the plugin object received from Ixserver.

Function	process
Parameter	unsigned char *buffer, unsigned int length
Return	0/1
Description	It is a function that processes request data transferred from Client.

Function	pollingCallback
Parameter	Ixserver_plugin_object *obj
Return	0/1
description	This function is to be registered as a callback function to be polled, and this function is responsible for sending event data to the client.

6.4 JsonRpcParser



It is a class that parses JSON data delivered from Client according to the specified protocol.

■ Member Function

Function	Import
Parameter	Unsigned char *buffer
Return	bool

description	Enter the data to be parsed

Function	getClass
Return	string
description	Returns the class name.

Function	getFunction
Return	string
description	Returns the function name.

Function	getParam
Return	JsonObject
description	Returns the parameter.

Function	getId
Return	Integer
description	Returns the id.

6.5 ObjectFactory

ObjectFactory
+CreateObject(in class : Char, out _instance : ObjectInstance) : bool

This class is responsible for creating the necessary objects.

■ Member Function

Function	CreateObject
Parameter	const char *_method, ObjectInstance *_instance
Return	boolean
Description	Create and return an object that matches the class name.

6.6 ObjectInstance

ObjectInstance
-m_name : string
+ObjectInstance() +ObjectInstance(in name : Char) +~ObjectInstance() +doAction(in name : const char, in param : JsonObject, out *retval : JsonObject) : bool

It is a class that has as a common attribute of objects to be created to perform client's request.

■ Member Function

Function	doAction
Parameter	const char* _name, JsonObject _param, JsonObject *_retval
Return	boolean
Description	As a function to execute the function requested from the client to the module, _name contains the function name that you want to execute. Parameter information is passed to _param. The result of the operation is passed through _retval.

6.7 EventTarget

EventTarget
-m_events[] : Event
+EventTarget() +EventTarget(in name : const char) +~EventTarget() +dispatchEvent(in event : JsonObject) : void +getEvent() : Event +deleteEvent(in _event : Event) : bool

It is a class that is a common attribute of objects to which event data should be transferred to Client.

■ Member Function

Function	dispatchEvent
Parameter	JsonObject event
Return	void
Description	Event data is saved.

Function	getEvent
Return	Event
description	Get event data to be passed to Client.

Function	deleteEvent
Parameter	Event *_event
Return	bool
Description	Delete the event received by the parameter.

6.8 Integration layer

Explains how to use Native interface integration to use platform resourse.

6.8.1 Naming conventions

The name of the Integration layer API consists of the following structure.

native_<InterfaceName>_<FunctionName>
Example: native_network_saveConfig

6.8.2 Return codes

The return code for function execution in the Integration layer is NATIVE_RETURN type. The definition for the type is in NativeTypes.h.

Return Code	Code	Desc.
	NATIVE_RETURN_OK	This is the type that returns when the internal operation of the plugin is completed normally.
	NATIVE_RETURN_ERROR	It is a type that returns when the internal operation of the plugin can not be completed normally.

6.8.3 Data objects

The structure of the data object for exchanging client JSON data with the Platform layer is defined in the Native header. The data object to be returned from the platform will also be defined in Native and the platform layer will maintain the structure that returns the result by filling this value. We designed this structure to make it useful to change

the result object to JsonObject type.

```
typedef struct{
        int id;
        int idInterface;
        char config[20];
        bool enable;
        char routeInterface[20];
        char destination[20];
        char gateway[20];
        int metric;
}NetworkRouteItem;
```

6.8.4 Event handling

Register the callback function from the integration layer to the platform layer to deliver events that occur on the platform. Callback API naming format and listener configuration API example are as follows.

```
native_<InterfaceName>_setListener

//Example:
typedef void (caswareListener)(CaswareEvent evt_type, void* data);
NATIVE_RETURN native_casware_setListener(caswareListener listener);
```

6.8.5 Error handling

NATIVE_RETURN_ERROR is returned if an error occurs in the execution result of the Integration layer.

6.9 Plugin Sequence

6.9.1 Plugin loading

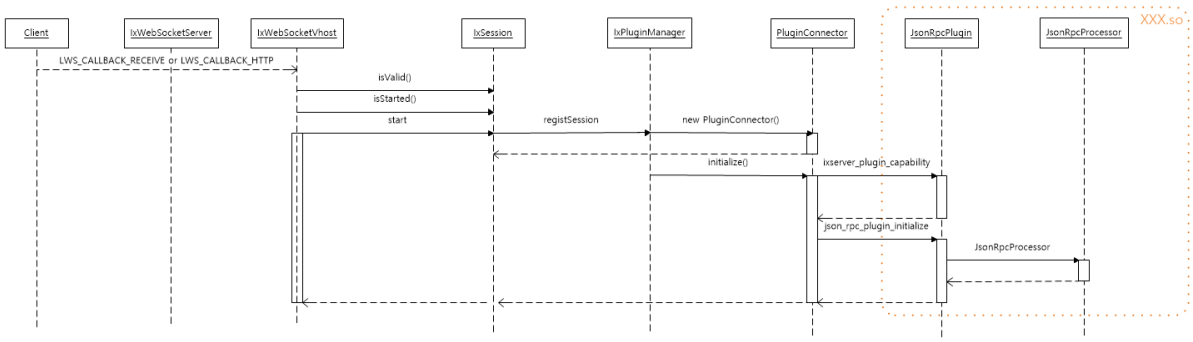


Figure 5. Sequence of plugin loading

When the client is connected to the server, the plugin is loaded in the following order.

1) IxWebSocketVhost checks the validity and start of Session at the time of receiving data from Client.

2) Session is valid and if it does not start Session is started and register Session through PluginManager, creates PluginConnector, and loads plugin file set through configuration file from PluginConnector.

3) The PluginConnector() calls the ixsever_plugin_capability () function of the loaded plugin.

4) plugin exchanges initialization data in ixserver_plugin_capability () function.

Plugin callback:

- ✓ Register the plugin's polling callback function in PluginConnector.
- ✓ Take the send_buffer function of PluginConnector in plugin.

Plugin handler:

- ✓ Register plugin's initialize function in PluginConnector.
- ✓ Register plugin's process function in PluginConnector.
- ✓ Register plugin's destroy function in PluginConnector.

5) PluginConnector calls the initialize function of loaded plugin.

6) Plugin initialize function creates a processor to handle client request & event and object.

6.9.2 Client request process

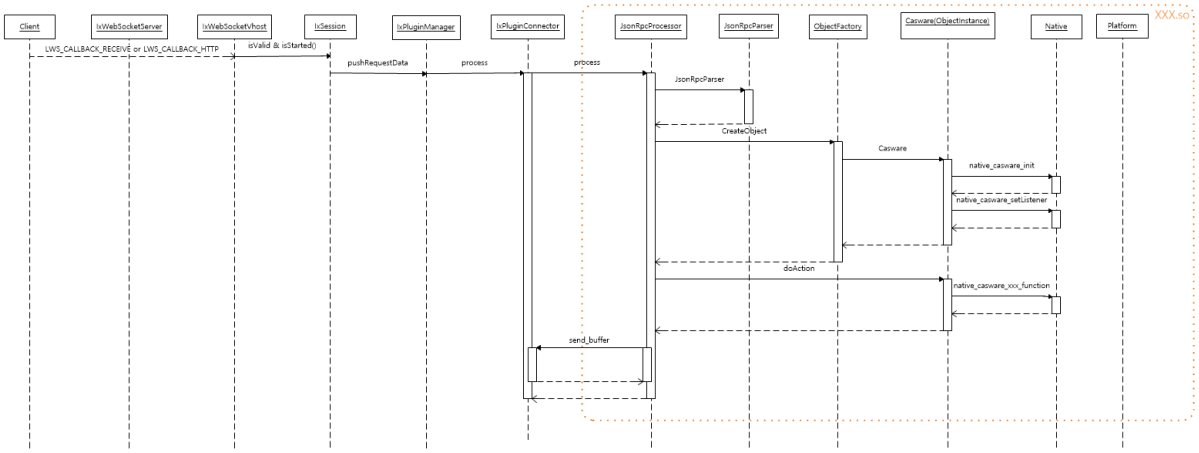


Figure 6. Sequence of client request processing

When a request is made from a client, it is processed in the following order.

1) The server passes the data passed from the client to the session.

2) Session passes the data through the process function of PluginConnector.

3) PluginConnector calls the process function of the previously registered plugin and passes the data.

4) The processor of the plugin parses the request data using the parser.

5) Get the class name of the request data and request the module to the ObjectFactory.

6) Request function is executed by passing function name and param of request data to doAction () function of Object obtained through ObjectFactory.

7) The return data obtained from the execution result is formatted according to the protocol, passed to the send_buffer callback, and the client receives the result.

6.9.3 Plugin Event Process

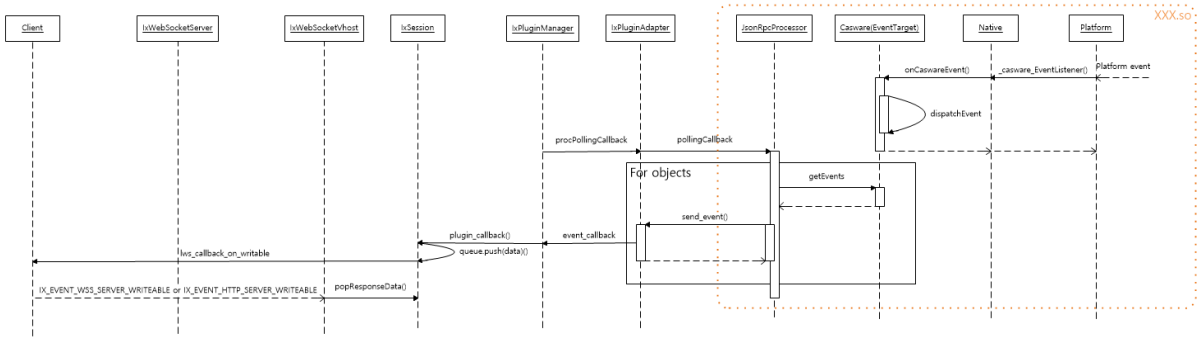


Figure 7. Sequence of plugin event processing

In order to receive a client event when an event occurs on the platform, the following initialization operation must be performed.

1) Register the event listener from the Native Integration layer to the platform layer.

2) The listener is specified so that the event listener registered in 1) becomes the event listener of OjbectClass.

```
int _casware_EventListener(CaswareEvent evt_type, void* data)
{
    g_CaswareListener(evt_type, data);
}
NATIVE_RETURN native_casware_OnCreate()
{
    Platform_addEventListener(_casware_EventListener)
}
NATIVE_RETURN native_casware_setListener(caswareListener listener)
{
    g_CaswareListener = (caswareListener *)listener;
}
```

When events occur on the platform, they are processed in the following order.

1) The event is delivered to the native event callback registered in the platform.

2) The event is delivered to the event callback of each plugin module registered in the native layer.

3) Put the event in the event queue of the object.

4) PluginConnector is called periodically the plugin's polling callback function.
5) In polling callback function, events accumulated in the objects managed by the processor are delivered to client through send_event function of PluginConnector.
6) The event is stored in the queue of the session and calls the libwebsocket function lws_callback_on_writable.
7) In libwebsocket, LWS_CALLBACK_SERVER_WRITEABLE indicates that the data can be sent, and event is sent to the client by calling lws_write function in session.