Logi Plugin SDK

Version 0.0.999.0

Contents

Introduction

This is the technical documentation of the Logi Plugin SDK - a system for integrating third-party applications and services with the Logi client configuration software (Options+ and G HUB). The first part, intended for plugin developers, describes the structure of the plugin archives, the communication protocol used between the plugins and the plugin manager, as well as the SDKs offered for developing plugins in C++ and JavaScript. The second part is intended for developers integrating the SDK with the Logi configuration software. It describes the client library used to communicate with the plugin manager from within the configuration software, and the communication protocol used by the Logi Marketplace client to interact with the plugin manager.

Part I: Plugin Developer Documentation

Chapter 1. Plugin Structure

Plugins are packaged as signed zip archives which may contain the following items:

Item Name	Required	Description
manifest.json	Yes	Manifest is the main entry point for the plugin. Will include a plugin description and will show what components are available in the plugin
localization	No	"localization" is the folder that contains list of yaml files for each supported language
plugin executable	No	Plugin executable is provided per platform basis. It will be run by plugin manager during the start of the system
installer executable	No	Installer executable could be provided per platform. Will be run every time when plugin is installed/updated/removed.
Additional content	No	Plugin could contain any data that is needed for plugin operations including images, audio/video, libraries, etc.

There could be separate package for mac and windows to decrease download time for user.

Chapter 2. Manifest File Structure

Manifest file should be called manifest.json and should contain following fields:

Item Name	Required	Description
plugin_id	Yes	Plugin id is unique identifier of the plugin.
name	Yes	Name of the plugin.
author	Yes	Author of the plugin.
description	Yes	Short description of the plugin
icon	Yes	Icon for the plugin, should be inside plugin package
version	Yes	Version for the plugin, should be in format Major.Minor.Build, where all 3 fields are numbers
URL	No	Plugin homepage
OS	Yes	Specifies applicable operation system for the plugin. This field is array of objects that contain platform and minimum version field. Possible value for this field is:

```
[
    {
        "platform": "mac",
        "minimumVersion" : "10.11"
    },
    {
        "platform": "windows",
        "minimumVersion" : "10"
    }
]
```

Item Name	Required	Description
platformVersion	Yes	Minimum version of the plugin manager
programPath	No	Path for the plugin executable, could be overriden by programPathMac and programPathWin variables. Plugin is allowed to not have any executable at all.
programPathMac	No	Overrides programPath on Mac OS.
programPathWin	No	Overrides programPath on Windows.
multipleAccountsAllowed	No - not supported	Indicates whether it is possible to have several account under the plugin.

application	No	This field is needed for application detection. The main use case for this is hot keys. The following example would show the field structure:

```
{
    "name": "Application Name [Optional]",
    "id": "application_id_<application_name>",
    "detection": {
        "osx": [
            {
                "bundlePath": "/System/Applications/<Application
Name>.app"
            }
        ],
        "win": [
            {
                "executable": "<app exe name>.exe",
                "registryKey": "",
                "registryPath": "HKEY_LOCAL_MACHINE/SOFTWARE/<app
exe registry path>.exe"
            }
        ]
    }
}
```

groups	No	Array of object, each object contains "groupId", "name" and "icon" fields. Allows to group plugin actions by the use case.
actions	No	Array of object, each object contains description for the actions that are provided by plugin
actions[].id	Yes	Id of the action, should be unique in scope of plugin (there should be no control with such id as well).
actions[].icon	No	Icon associated with the action
actions[].tooltip	Yes	Description that describes to user of what the action suppose to do.
actions[].name	Yes	Name of the action
actions[].groupIds	No	Array of strings that shows list of groups where this action belongs to.
actions[].keystroke	No	Indicates that this action is keystroke action, so that calling of this action will be resulted in keystroke command. The keystroke should have following format

```
{
    "osx": {
        "code": 54,
        "modifiers": [
            227
        ],
        "virtualKeyId": "VK_COMMA"
    },
    "win": {
        "code": 54,
        "modifiers": [
            227
        ],
        "virtualKeyId": "VK_COMMA"
    }
}
```

controls	No	Array of object, each object contains description for the analog controls that are provided by plugin. See Control's table below for more details

requiredCapabilities	Yes	Array of strings. API level that plugin requires. Right now only "base" is supported
installationCommands	No	Array of objects of type command, please find description below. Each command will be run after plugin is copied into it's directory from marketplace and before running the plugin.
uninstallationCommands	No	Array of objects of type command, please find description below. Each command will be run after plugin is stopped and before actual removal of the plugin
command.executable	Yes	Executable in the plugin package that should be called
command.parameters	No	Array of string that would be passed to the executable
command.silent	No	Indicates whether console should be shown to user
command.runAsAdmin	No	Indicates whether command should be run under elevated permissions. Consent will be shown to user.
command.platform	No	Should be either "win" or "mac". Indicates whether executable should run on particular platform.

2.1. Control

Analog control structure

Field Name	**Required**	**Description**
id	Yes	Id of the analog control, should be unique in scope of plugin (there should be no action with such id as well).
tooltip	Yes	Description that describes to user of what the analog control suppose to do.
name	Yes	Name of the analog control
groupIds	No	Array of strings that shows list of groups where this analog control belongs to.
minimum	Yes	Minimum value of the analog control, could be overriden by plugin in runtime
maximum	Yes	Maximum value of the analog control, could be overriden by plugin in runtime
defaultStep	Yes	Default step for increasing and decreasing value.
unitOfMeasurement	No	Unit of measure for the value. For example: "px", "m", "cm", etc.
gestureAxisInfoWin	No	Information about gesture, specific to **Windows** platform, see GestureAxisInfo table below for more details
gestureAxisInfoMac	No	Information about gesture, specific to **macOS** platform,see GestureAxisInfo table below for more details

2.2. GestureAxisInfo

Gesture Information structure.

Field Name	Required	Description
invertable	No	Boolean value: *true* or *false*. If *true* then control should respect inverted scrolling setting applied to mouse scroll, and *false* otherwise.
speedControl	No	Boolean value: *true* or *false*. If *true* thumbwheel speed setting affects the control adjustment, and *false* otherwise.
speedFactor	No	Float value: from -1.0 to 1.0, be default is 0.0. if >0 then increases speed of controll adjustment and <0 decreases speed. 0.0 has no effect.
actionDuration	No	Thumbwheel units count that will trigger the next left/right action. This field contains object of type Threshold see below for more details
actionThreshold	No	Duration in milliseconds after action is triggered during which mouse motion is ignored. This field contains object of type ActionDurationMs see below for more details

Threshold

Field Name	Required	Description
minimumSensitivityThreshold	Yes	UInt value. For lowest speed setting - if 0, use default hard coded values.
maximumSensitivityThreshold	Yes	UInt value. For highest speed setting - if 0, use default hard coded values.

ActionDurationMs

Field Name	Required	Description
minimumSensitivityThreshold	Yes	UInt value. For lowest speed setting
maximumSensitivityThreshold	Yes	UInt value. For highest speed setting

There could be separate package for mac and windows to decrease download time for user.

Chapter 3. Localization

Localization of plugin consists of two parts:

1. localization of the text mentioned in manifest
2. localization of the plugin messages that are sent using API.

3.1. Localization of the text in manifest

Plugin should have "localization" folder. In that folder there should be set of "yaml" files for each supported locale, for example "en-US.yaml", "de.yaml", etc. Each of this file will contain key and translated value for this key.

For each user facing item in the manifest follwoing will be performed:

1. Appropriate yaml file will be find according to Options+ or Ghub locale.
2. In case if no yaml file coresponding to locale will be found "en-US.yaml" will be used. In case if this file does not exists as well - no translation will be done.
3. String in manifest will be searched in yaml file as a key, and if it will be found - will be substituted by value.
4. If string will not be found in yaml file - string will be sent to UI as is.

3.2. Localization of the text for API

During the handshake system passes locale to the plugin. In case if locale will be changed and connection with the plugin will be terminated. On reconnection - new locale will be passed to plugin.

Plugin should take care of localizing messages in sent to API. Helper function would be provided in SDK.

There could be separate package for mac and windows to decrease download time for user.

Chapter 4. Localization

Localization of plugin consists of two parts:

1. localization of the text mentioned in manifest
2. localization of the plugin messages that are sent using API.

4.1. Localization of the text in manifest

Plugin should have "localization" folder. In that folder there should be set of "yaml" files for each supported locale, for example "en-US.yaml", "de.yaml", etc. Each of this file will contain key and translated value for this key.

For each user facing item in the manifest follwoing will be performed:

1. Appropriate yaml file will be find according to Options+ or Ghub locale.
2. In case if no yaml file coresponding to locale will be found "en-US.yaml" will be used. In case if this file does not exists as well - no translation will be done.
3. String in manifest will be searched in yaml file as a key, and if it will be found - will be substituted by value.
4. If string will not be found in yaml file - string will be sent to UI as is.

4.2. Localization of the text for API

During the handshake system passes locale to the plugin. In case if locale will be changed and connection with the plugin will be terminated. On reconnection - new locale will be passed to plugin.

Plugin should take care of localizing messages in sent to API. Helper function would be provided in SDK.

Chapter 5. Inter process communication protocol

This section describes protocol for message exchange between plugin manager and the plugin. SDK implementation for this protocol would be described later.

Communication could be done by websocket or named pipe/unix domain sockets. Named pipes should be preferable way of communication

Each message will be packed into following structure:

```
message Envelope
{
    uint32 id = 1;
    google.protobuf.Any message = 2;
    optional ResponseInfo response = 3;
}

message ResponseInfo
{
     Code result = 1; // code
     string what = 2; // description
}
```

Parameters

- **id**: The id of the message, response ot the request will have the same id as an request
- **message**: The message embedded to this request/response
- **response**: In case if this envelope is response - it should contain ResponseInfo structure that will contains code (see Error Codes section) and String description.

5.1. Hello message

```
message PluginHello
{
    string plugin_id        = 1;
    string plugin_code      = 2;
    string plugin_version   = 3;
    int32 protocol_version  = 4;
}
```

Direction
From plugin

Capability
Base

Description
That should be first message that is send by plugin to the system.

Parameters

- **plugin_id**: The id of the plugin, should be the same as in manifest
- **plugin_code**: Security code for the connection
- **plugin_version**: Version of the plugin.
- **protocol_version**: Required version of the protocol.

Return strucutre
Connection will be dropped on failures, following structure could be returned appon success:

```
message ManagerHello
{
    string manager_version          = 1;
    optional string options_version = 2;
    optional string ghub_version    = 3;
    int32 protocol_version          = 4;
    string language_code            = 5;
}
```

`manager_version`	Version of the manager
`options_version`	Options+ version, if installed
`ghub_version`	G HUB version, if installed
`protocol_version`	Protocol version used
`language_code`	Currently used language

5.2. Setting settings

```
message Settings
{
    string setting_name = 1;
    string setting_value = 2;
}
```

Direction
From plugin

Capability
Base

Description
Saving custom settings in the system

Parameters

- **setting_name**: Key name for settings
- **setting_value**: Value for settings

Return strucutre
No custom message will be returned. ResponseInfo contains result of execussion.

5.3. Getting settings

```
message SettingsRequest
{
    string setting_name =1;
}
```

Direction
From plugin

Capability
Base

Description
Request settings from system (previously set by setSettings message) by key

Parameters

- **setting_name**: The key for the setting

Return strucutre

```
message Settings
{
    string setting_name = 1;
    string setting_value = 2;
}
```

`setting_name`	Requested key
`setting_value`	Value that corresponds to the key

5.4. Logging Message

```
message LogEvent
{
    enum LOG_LEVEL
    {
        TRACE = 0;
        DEBUG = 1;
        INFO = 2;
        WARNING = 3;
        ERROR = 4;
    }
    LOG_LEVEL log_level = 1;
    string message =2
}
```

Direction
From plugin

Capability
Base

Description
Adding message to the system log on plugin behalf.

Parameters

- **log_level**: appropriate log level for the event
- **message**: message to be logged

Return strucutre
No custom message will be returned. ResponseInfo contains result of execussion.

5.5. Updating status of analog control

```
message AnalogControlStatus
{
    enum IMAGE_TYPE
    {
        NONE = 0;
        BASE64 = 1;
        FILE = 2;
    }
    string analog_control_id = 1;
    string analog_control_instance_id  = 2;
    optional float min_value = 3;
    optional float max_value = 4;
    optional float current_value = 5;
    optional bool active = 6;
    optional string text = 7;
    optional string image = 8;
    optional IMAGE_TYPE image_type=9;
    optional string error = 10;
    optional Code error_code = 11;
}
```

Direction
From plugin

Capability
Base

Description
Update information about the analog control (whether the analog_control is active or not, min /max values, current value). In case if error field exists and not empty - that would mean that error appears either as a response to previously updated analog control.

Parameters

- **analog_control_id**: id of analog control to send update for.
- **analog_control_instance_id** : instance id of the analog control. Plugin will receive it with visibility event
- **min_value**: minimum value of the analog control, if this value is not provided then **minimum** from manifest will be used
- **max_value**: max value of the analog control, if this value is not provided then **maximum** from manifest will be used
- **current_value**: Current value for the tool
- **active**: Indicates whether analog control is active
- **text**: Text that could be shown on user device
- **image**: image that should be shown on user device.
- **image_type**: Type of the image, file base and base64 inline image are supported
- **error**: String error description
- **error_code**: Error code, check error code section

Return strucutre
No custom message will be returned. ResponseInfo contains result of execussion.

5.6. Updating status of an action cell

```
message ActionCellStatus
{
    enum IMAGE_TYPE
    {
        NONE = 0;
        BASE64 = 1;
        FILE = 2;
    }
    string action_instance_id= 1;
    string action_id = 2
    optional string text = 3;
    optional string image = 4;
    optional IMAGE_TYPE image_type=5;
    optional string error = 6;
    optional Code error_code = 7;
}
```

Direction
From plugin

Capability
Base

Description
This command will be sent from plugin to manager and usually will be a respond for TriggerAction.

This command will be used for such purposes:

1. Update screen image for the devices that have screen
2. In dicate the result of the action for other devices.

In case if error field exists and not empty - that would mean that error appears either as a response to previously called action.

Parameters

- **action_id**: id of action to send update for.
- **action_instance_id**: instance id of the action. Plugin will receive it with visibility event
- **text**: Text that could be shown on user device
- **image**: image that should be shown on user device.
- **image_type**: Type of the image, file base and base64 inline image are supported
- **error**: String error description
- **error_code**: Error code, check error code section

Return strucutre
No custom message will be returned. ResponseInfo contains result of execussion.

5.7. Heartbeat message

```
message Ping
{
}
```

Direction
From system

Capability
Base

Description
The message is send periodically to plugin. Plugin should immidiately response to it.

Parameters
None

Return strucutre
Connection will be dropped in case if plugin will not answer timely

```
message Pong
{
}
```

5.8. Rate limit exceeded

```
message RetryAfter
{
    int interval = 1;
}
```

Direction
From system

Capability
Base

Description
Plugin should not send any events to manager for the specific interval. The interval is in milliseconds.

Parameters

- **interval**: interval in miliseconds

Return strucutre
No custom message will be returned. ResponseInfo contains result of execussion.

5.9. Trigger action

```
message TriggerAction
{
    string action_instance_id = 1;
    string action_id = 2;
    optional string configuration = 3;
}
```

Direction
From system

Capability
Base

Description
Message is actually calling action that plugin provide. This message would be called during key down event, or for other event that indicating start of an action.

Configuration is configuration of this action as it was defined by SendActionConfigurationScheme.

Parameters

- **action_instance_id**: instance id of an action.
- **action_id**: action id as stated in manifest
- **configuration**: configuration for an action

Return strucutre

```
message ActionCellStatus
{
    enum IMAGE_TYPE
    {
        NONE = 0;
        BASE64 = 1;
        FILE = 2;
    }
    string action_instance_id= 1;
    string action_id = 2
    optional string text = 3;
    optional string image = 4;
    optional IMAGE_TYPE image_type=5;
    optional string error = 6;
    optional Code error_code = 7;
}
```

`action_instance_id`	action instance id as stated in request
`action_id`	id of action to send update for.
`text`	Text to show on user device
`image`	image that should be shown on user device.
`image_type`	Type of the image, file base and base64 inline image are supported
`error`	String error description
`error_code`	Error code, check error code section

5.10. Release action

```
message ReleaseAction
{
    string action_instance_id= 1;
    string action_id =2;
    optional string configuration = 3;
}
```

Direction
From system

Capability
Base

Description
This message will be called when key up event appears or similar event for other control. Plugin could ignore this event if the action is not continuous.

Configuration is configuration of this action as it was defined by SendActionConfigurationScheme.

Parameters

- **action_instance_id**: instance id of an action.
- **action_id**: action id as stated in manifest
- **configuration**: configuration for an action

Return strucutre
No custom message will be returned. ResponseInfo contains result of execussion.

5.11. Update analog control

```
message UpdateAnalogControl
{
    string analog_control_id      = 1;
    string instance_id            = 2;
    optional string configuration = 3;

    oneof update_type
    {
        float value                   = 4;
        float delta                   = 5;
    }
}
```

Direction
From system

Capability
Base

Description
Message for updating value of the analog control.

Configuration is configuration of this action as it was defined by SendActionConfigurationScheme.

Parameters

- **analog_control_id**: id of analog control.
- **instance_id**: instance id of analog control
- **configuration**: configuration of an action
- **update_type**: could contain either exact value to set or the delta

Return strucutre

```
message AnalogControlStatus
{
    enum IMAGE_TYPE
    {
        NONE = 0;
        BASE64 = 1;
        FILE = 2;
    }
    string analog_control_id = 1;
    string analog_control_instance_id  = 2;
    optional float min_value = 3;
    optional float max_value = 4;
    optional float current_value = 5;
    optional bool active = 6;
    optional string text = 7;
    optional string image = 8;
    optional IMAGE_TYPE image_type=9;
    optional string error = 10;
    optional Code error_code = 11;
}
```

`analog_control_id`	analog control id
`analog_control_instance_id`	analog control instance id
`min_value`	minimum value of analog control
`max_value`	maximum value of analog control

`current_value`	analog control current value
`active`	indicated whether analog control is active
`text`	Text to show on user device
`image`	image that should be shown on user device.
`image_type`	Type of the image, file base and base64 inline image are supported
`error`	String error description
`error_code`	Error code, check error code section

5.12. Get analog control value

```
message GetAnalogControlValue
{
  string analog_control_id = 1;
  string instance_id = 2;
  optional string configuration = 3;
}
```

Direction
From system

Capability
Base

Description
Plugin manager will request the initial value on the plugin start.

Parameters

- **analog_control_id**: id of analog control.
- **instance_id**: instance id of analog control
- **configuration**: configuration of an action

Return strucutre

```
message AnalogControlStatus
{
    enum IMAGE_TYPE
    {
        NONE = 0;
        BASE64 = 1;
        FILE = 2;
    }
    string analog_control_id = 1;
    string analog_control_instance_id  = 2;
    optional float min_value = 3;
    optional float max_value = 4;
    optional float current_value = 5;
    optional bool active = 6;
    optional string text = 7;
    optional string image = 8;
    optional IMAGE_TYPE image_type=9;
    optional string error = 10;
    optional Code error_code = 11;
}
```

`analog_control_id`	analog control id
`analog_control_instance_id`	analog control instance id
`min_value`	minimum value of analog control
`max_value`	maximum value of analog control
`current_value`	analog control current value
`active`	indicated whether analog control is active
`text`	Text to show on user device
`image`	image that should be shown on user device.

`image_type`	Type of the image, file base and base64 inline image are supported
`error`	String error description
`error_code`	Error code, check error code section

5.13. Set action visibility

```
message VisibilityChanged
{
    string control_id             = 1;
    string action_instance_id     = 2;
    bool visible                  = 3;
    optional string configuration = 4;
}
```

Direction
From system

Capability
Base

Description
Plugin manager will notify the plugin that particular item become visible or hidden.

Parameters

- **control_id**: id of analog control or action.
- **action_instance_id**: instance id of analog control or action
- **visible**: indicates whether action or analog control is visible or not
- **configuration**: configuration of an action

Return strucutre
In case if the id represent action - ActionCellStatus structure should be sent back. In case if the id represent analog control - AnalogControlStatus structure should be sent back.

```
message AnalogControlStatus
{
    enum IMAGE_TYPE
    {
        NONE = 0;
        BASE64 = 1;
        FILE = 2;
    }
    string analog_control_id = 1;
    string analog_control_instance_id  = 2;
    optional float min_value = 3;
    optional float max_value = 4;
    optional float current_value = 5;
    optional bool active = 6;
    optional string text = 7;
    optional string image = 8;
    optional IMAGE_TYPE image_type=9;
    optional string error = 10;
    optional Code error_code = 11;
}
```

`analog_control_id`	analog control id
`analog_control_instance_id`	analog control instance id
`min_value`	minimum value of analog control
`max_value`	maximum value of analog control
`current_value`	analog control current value
`active`	indicated whether analog control is active

text	Text to show on user device
image	image that should be shown on user device.
image_type	Type of the image, file base and base64 inline image are supported
error	String error description
error_code	Error code, check error code section

```
message ActionCellStatus
{
    enum IMAGE_TYPE
    {
        NONE = 0;
        BASE64 = 1;
        FILE = 2;
    }
    string action_instance_id= 1;
    string action_id = 2
    optional string text = 3;
    optional string image = 4;
    optional IMAGE_TYPE image_type=5;
    optional string error = 6;
    optional Code error_code = 7;
}
```

action_instance_id	action instance id as stated in request
action_id	id of action to send update for.
text	Text to show on user device
image	image that should be shown on user device.
image_type	Type of the image, file base and base64 inline image are supported
error	String error description
error_code	Error code, check error code section

5.13.1. Batch updates

```
message VisibilityChangedList
{
    repeated VisibilityChanged visibility = 1;
}
```

Direction
From system

Capability
Base

Description
Plugin manager will notify the plugin that items become visible or hidden.

Parameters

- **visibility**: list of **VisibilityChanged** messages

Return strucutre
AnalogControlStatusList Message

```
message AnalogControlStatusList
{
    repeated AnalogControlStatus statuses = 1;
}
```

`statuses`	list of **AnalogControlStatus** messages

5.14. Request configuration scheme request

```
message PluginActionConfigurationSchemeRequest
{
    string action_id            = 1;
    string analog_control_id    = 2;
    string action_instance_id   = 3;
    optional string username    = 4;
}
```

Direction
From system

Capability
Base

Description
System request configuration for action. This method will be called when user will select action on action picker.

Parameters

- **action_id**: id of action.
- **analog_control_id**: id of analog control.
- **action_instance_id**: instance id of action or analog control
- **username**: optionally selected username

Return strucutre

```
message UserInfo
{
    string username                 = 1;
    optional string printed_name    = 2;
    optional string icon            = 3;
}

message PluginActionConfigurationSchemeResponse
{
    optional bool login_required      = 1;
    optional string selected_username = 2;
    repeated UserInfo users           = 3;
    string action_id                  = 4;
    string analog_control_id          = 5;
    string action_instance_id         = 6;
    optional string json_schema       = 7;
    optional string ui_schema         = 8;
    optional bool loging_required      = 99;

}
```

`loging_required`	indicates whther login is required
`selected_username`	selected username (by default)
`users`	list of logged in user that could have access to that action/analog control
`action_id`	id of action
`analog_control_id`	id of analog control
`action_instance_id`	action instance id/analog control instance id

`json_schema`	json schema, check https://github.com/rjsf-team/react-jsonschema-form
`ui_schema`	ui schema, check https://github.com/rjsf-team/react-jsonschema-form
`loging_required`	temporary field for compatibility, please do not use it

5.15. Submit configuration

```
message PluginActionConfigurationResponse
{
    string action_instance_id   = 1;
    string action_id            = 2;
    string form_result          = 3;
}
```

Direction
From system

Capability
Base

Description
This actions is called when user clicks submit on action configuration panel. On response - additional configuration could be optionally provided.

Parameters

- **action_id**: id of action or analog control.
- **action_instance_id**: instance id of action or analog control
- **form_result**: json result for provided json scheme

Return strucutre
Optionally could PluginActionConfigurationSchemeResponse could be send, in case of empty structure - nothing will be shown on UI.

```
message UserInfo
{
    string username                 = 1;
    optional string printed_name    = 2;
    optional string icon            = 3;
}

message PluginActionConfigurationSchemeResponse
{
    optional bool login_required     = 1;
    optional string selected_username = 2;
    repeated UserInfo users           = 3;
    string action_id                  = 4;
    string analog_control_id          = 5;
    string action_instance_id         = 6;
    optional string json_schema       = 7;
    optional string ui_schema         = 8;
    optional bool loging_required      = 99;
}
```

`login_required`	indicates whther login is required
`selected_username`	selected username (by default)
`users`	list of logged in user that could have access to that action/analog control
`action_id`	id of action
`analog_control_id`	id of analog control
`action_instance_id`	action instance id/analog control instance id

`json_schema`	json schema, check https://github.com/rjsf-team/react-jsonschema-form
`ui_schema`	ui schema, check https://github.com/rjsf-team/react-jsonschema-form
`loging_required`	temporary field for compatibility, please do not use it

5.16. Init login

```
message InitLogin
{
    string redirect_url = 1;
}
```

Direction
From system

Capability
Base

Description
This actions is called when user clicks on login button

Parameters

- **redirect_url**: url for login handler (for oauth2 auth flow).

Return strucutre
No custom message will be returned. ResponseInfo contains result of execussion.

5.17. Authentication Token Received

```
message OauthResponse
{
    string state = 1;
    string token = 2;
    string url   = 3;
}
```

Direction
From system

Capability
Base

Description
This actions is called when user clicks on login button

Parameters

- **state**: state that is received from oauth2 flow.
- **token**: authentication token
- **url**: redirection url

Return strucutre
No custom message will be returned. ResponseInfo contains result of execussion.

5.18. Logout request

```
message Logout
{
    string username = 1;
}
```

Direction
From system

Capability
Base

Description
This actions is called when user clicks on logout button

Parameters

- **username**: username that should be logged out

Return strucutre
No custom message will be returned. ResponseInfo contains result of execussion.

5.19. Logout request

```
message Logout
{
    string username = 1;
}
```

Direction
From system

Capability
Base

Description
This actions is called when user clicks on logout button

Parameters

- **username**: username that should be logged out

Return strucutre
No custom message will be returned. ResponseInfo contains result of execussion.

5.20. User List

```
message GetUserInfoList
{
}
```

Direction
From system

Capability
Base

Description
Request list of logged in users from plugin

Parameters
None

Return strucutre

```
message UserInfo
{
    string username                 = 1;
    optional string printed_name    = 2;
    optional string icon            = 3;
}

message UserInfoList
{
    repeated UserInfo user_info = 1;
}
```

`user_info`	LIst of users

Chapter 6. Error Codes

The following error codes are used to indicate the success state of plugin API calls.

0	`INVALID`	Error codes must be negative, so there should be no error code with code 0.
1	`SUCCESS`	Indicates successfull operation
-1	`INVALID_ARG`	
-2	`INVALID_DEVICE`	
-3	`NO_SUCH_PATH`	
-4	`CANCELLED`	
-5	`NOT_IMPLEMENTED`	
-6	`INVALID_VERB`	
-7	`NOT_READY`	
-8	`FAULTED`	
-9	`UNREACHABLE`	
-10	`UNAUTHORIZED`	
-11	`DUPLICATE_NAME`	
-12	`NOT_FOUND`	
-13	`EXCEPTION`	
-14	`CONFLICT`	

Chapter 7. C++ SDK documentation

C++ SDK contains plugin_client - main SDK class that implement communication protocol and also set of utilities that makes development of plugin simpler.

7.1. plugin_client class

plugin_client is virtual class, so to use it, subclass should be created. To init the class constructor should be called:

```
plugin_client(std::string plugin_id, std::string version, std::string secret);
```

Parameters

- **plugin_id**: The id of the plugin, should be the same as in manifest
- **version**: Version of the plugin.
- **secret**: the secret that found in "token" environment variable

After instatiating plugin_client subclass "connect()" should be called to start the process.

7.1.1. set_log_function()

```
typedef std::function<void(const std::string &)> log_function;
void set_log_function(log_function f);
```

Description
Allow to set custom logging function for the internal logs.

Parameters

- **f**: log function that will get single std::string argument

Return values
None

7.1.2. set_use_text_format()

```
void set_use_text_format(bool use_text);
```

Description
Configuring client to use either json format or protobuf binary format

Parameters

- **use_text**: True means json format, False - protobuf binary format.

Return values
None

7.1.3. set_connect_handler()

```
typedef std::function<void(websocketpp::connection_hdl)> connection_function;
void set_connect_handler(connection_function f);
```

Description
Allowing to set handler for connection. Default handler will send PluginHello message.

Parameters

- **f**: Connection handler

Return values
None

7.1.4. set_disconnect_handler()

```
typedef std::function<void(websocketpp::connection_hdl)> connection_function;
void set_disconnect_handler(connection_function f);
```

Description
Allowing to set handler for disconnection. Default handler do nothing.

Parameters

- **f**: Disonnection handler

Return values
None

7.1.5. is_connected()

```
bool is_connected() const;
```

Description
Indicating whether the plugin is connected to the system

Parameters
None

Return values

`true`	Client is connected to server
`false`	Client is not connected to server

7.1.6. disconnect()

```
void disconnect();
```

Description
Disconnect from server

Parameters
None

Return values
None

7.1.7. wait_for_disconnect()

```
void wait_for_disconnect();
```

Description
Stop current thread untill disconnect operation is finished.

Parameters
None

Return values
None

7.1.8. send_message()

```
uint64_t send_message(const google::protobuf::Message &msg);
```

Description
Sends message to server, if connection is established

Parameters

- **msg**: Message to be send.

Return values
message_id generated for this call

7.1.9. send_message_with_callback()

```
typedef std::function<void(const google::protobuf::Any *, uint32_t, const std::string &)>
                            response_handler_function;
uint64_t send_message_with_callback(const google::protobuf::Message &msg,
                                            response_handler_function on_response,
                                            const unsigned timeout_ms = 10000);
```

Description
Sends message to server, if connection is established. When response is arrived or timeout happen - response handler will be called.

Parameters

- **msg**: Message to be send.
- **on_response**: Handler, that will be called on response.
- **timeout_ms**: Timeout interval in ms

Return values
message_id generated for this call

7.1.10. send_error_response()

```
void send_error_response(uint64_t msg_id, plugin::protocol::ResponseInfo::Code result_code, const std::string
&description = "");
```

Description
Sends error response to server having error code and optional description

Parameters

- **msg_id**: Message id that should be sent in response
- **result_code**: Result code. Check Error codes section
- **description**: OPtional string

Return values
None

7.1.11. send_ok_response()

```
void send_ok_response(uint64_t msg_id);
```

Description
Send ok response.

Parameters

- **msg_id**: Message id that should be sent in response

Return values
None

7.1.12. enable_control_buffering()

```
void enable_control_buffering(const std::map<std::string, std::string> &controls);
```

Description
Register controls for buffering.

Parameters

- **controls**: "{control_id: buffer_key}" map to register actions for some specific buffers.

Return values
None

7.1.13. Callbacks

```
virtual void on_connect() {};
virtual void on_disconnect() {};
virtual void on_manager_hello(const logi::plugin::protocol::ManagerHello &msg, const uint64_t &msg_id) {}
virtual void on_init_login(const logi::plugin::protocol::InitLogin &msg, const uint64_t &msg_id) {}
virtual void on_logout(const logi::plugin::protocol::Logout &msg, const uint64_t &msg_id) {}
virtual void on_trigger_action(const logi::plugin::protocol::TriggerAction &msg, const uint64_t &msg_id) {}
virtual void on_release_action(const logi::plugin::protocol::ReleaseAction &msg, const uint64_t &msg_id) {}
virtual void on_visibility_changed(const logi::plugin::protocol::VisibilityChanged &msg, const uint64_t &msg_id) {}
virtual void on_visibility_changed_list(const logi::plugin::protocol::VisibilityChangedList &msg, const uint64_t
&msg_id) {}
virtual void on_update_analog_control(const logi::plugin::protocol::UpdateAnalogControl &msg,
                                        const uint64_t &msg_id) {}
virtual void on_get_analog_control_value(const logi::plugin::protocol::GetAnalogControlValue &msg,
                                            const uint64_t &msg_id) {}
virtual void on_plugin_action_configuration_scheme_request(
    const logi::plugin::protocol::PluginActionConfigurationSchemeRequest &msg,
    const uint64_t &msg_id) {}
virtual void on_plugin_action_configuration_response(
    const logi::plugin::protocol::PluginActionConfigurationResponse &msg,
    const uint64_t &msg_id) {}
virtual void on_saved_data(const logi::plugin::protocol::SavedData &msg, const uint64_t &msg_id) {}
virtual void on_oauth_response(const logi::plugin::protocol::OauthResponse &msg, const uint64_t &msg_id) {}
virtual void on_get_users_info(const logi::plugin::protocol::GetUserInfoList &msg, const uint64_t &msg_id) {}
```

Description
When messages are send to plugin - callbacks are called in the plugin_client subclass.

7.2. Utilities

7.2.1. open_url()

```
void open_url(const std::string &url);
```

Description
Opens default system browser

Parameters

- **url**: Url to open in the default system browser

Return values
None

7.2.2. save_secret()

```
void save_secret(const std::string &plugin_name, const std::string &name, const std::string &value);
```

Description
Saves the plugin secret in secure storage. That would be keychain for mac and secret storage for windows.

Parameters

- **plugin_name**: User undestandable name of plugin
- **name**: Storage value name
- **value**: Value to push into storage

Return values
None

7.2.3. load_secret()

```
std::string load_secret(const std::string &plugin_name, const std::string &name);
```

Description
Loads value saved by save_secret function

Parameters

- **plugin_name**: User undestandable name of plugin
- **name**: Storage value name

Return values
Value previously saved, or empty value in case of error.

7.2.4. delete_secret()

```
void delete_secret(const std::string &plugin_name, const std::string &name);
```

Description
Delete value saved by save_secret function

Parameters

- **plugin_name**: User undestandable name of plugin
- **name**: Storage value name

Return values
None

7.3. buffer_manager class

Description
Buffer manager keeps in mind all the registered controls and their buffers. The main task of buffer manager is channeling incoming events to corresponding buffers.

7.3.1. is_buffering_enabled()

```
bool is_buffering_enabled() const;
```

Description
Let its users to know if there are some registered for buffering actions;

Parameters

None

Return values

true - if there are registered controls for buffering;

false - otherwise;

7.3.2. register_controls()

```
void register_controls(const std::map<std::string, std::string> &controls);
```

Description
Creates buffers for the specified controls.

Parameters

- **controls**: "{control_id: buffer_key}" map to register actions for some specific buffers.

Return values

None

7.3.3. register_custom_buffer()

```
void register_custom_buffer(const std::string &buffer_name, std::unique_ptr<buffer> buffer);
```

Description
Allows users to provide their own buffers realisation;

Parameters

- **buffer_name**: sugested new buffer name;
- **buffer**: new Buffer obj;

Return values

None

7.3.4. process_event()

```
void process_event(const logi::plugin::protocol::Envelope &event);
```

Description
All the requests and responses should go through this method. If there is no buffer registered for this event it will be dispatched immediately.

Parameters

- **event**: Any request form server or response form client.

Return values

None

7.3.5. set_buffered_message_callback()

```
using buffered_message_callback = std::function<void(const uint64_t msg_id)>;
void set_buffered_message_callback(buffered_message_callback callback);
```

Description
Callback for all the events that weren't immidiately dispatchet to the plugin but were buffered instead; This callback is assigned to all active buffers.

Parameters

- **callback**: Callback to dispatch an event.

Return values

None

7.4. buffer class

Description
Is a general interface for different buffering strategies implementation.

7.4.1. on_update_analog_control()

```
void on_update_analog_control(const logi::plugin::protocol::UpdateAnalogControl &msg, const uint64_t msg_id)
```

Description
Dispatches the firs incomng AnalogControl event and buffers all the following ones.

Parameters

- **msg**: Analog to be buffered.
- **msg_id**: Event id. It can be processed by buffer as well (usualy as additional identifier of the control).

Return values

None

7.4.2. on_message_response()

```
void on_message_response(const uint64_t msg_id)
```

Description

Receives notification that plugin is finished processing event with id.

Parameters

- **msg_id**: Event id.

Return values

None

7.4.3. set_buffered_message_callback()

```
using buffered_message_callback = std::function<void(const uint64_t msg_id)>;
void set_buffered_message_callback(buffered_message_callback callback);
```

Description
Callback for all the events that weren't immidiately dispatchet to the plugin but were buffered instead;

Parameters

- **callback**: Callback to dispatch an event.

Return values

None

7.5. throughput_buffer class

Description

It’s realisation of buffer interface. Buffer adapts to the plugin’s processing speed by sending buffered events as soon as the plugin is finished with previous ones. This approach requires us to not only intercept incoming analog control messages, but outgoing responses as well.

7.6. timer_buffer class

Description

It’s realisation of buffer interface. Buffer accamulates events with a set time interval.

Chapter 8. JavaScript SDK

Setup

1. In your application create folder where the plugin itself will be located - e.x. "js_sdk".
2. Copy in this folder all files from folder src. (It must be two files and one folder with schema) - (in a future release all will be compiling with help webpack - and we will have only one file)
3. In your application and this dependency in package.json: - npm i @bufbuild/protobuf reconnecting-websocket.

Usage

1. In your application import our Client - import Client from 'your path to file client.js' ▯ e.x. import Client from './browser/client';
2. Now we need to say hello our server - create instance of the class Client const clientApp = new Client({}) And in object put your init data ▯ pluginId: '', pluginCode: '', pluginVersion: ''
3. Call function init - clientApp.init();
4. If you need get information from server - please select subscription function which you need - clientApp.onTriggerAction = (msg) ▯ {}

8.1. sendMessage()

```
sendMessage = (type, id, obj) => {}
```

Description
Allow to set custom logging function for the internal logs.

Parameters

- **type**: Protobuf type messag
- **id**: id of the message, should be unique
- **obj**: Object with data

Return values
None

8.2. sendResponseMessage()

```
sendResponseMessage = (type, id, obj, err) => {}
```

Description
Allow to set custom logging function for the internal logs.

Parameters

- **type**: Protobuf type messag
- **id**: id of the message, should be unique
- **obj**: Object with data
- **err**: Error object to sent

Return values
None

8.3. Callbacks

```
onHello() {}
onTriggerAction() {}
onPluginActionConfigurationSchemeRequest() {}
onPluginActionConfigurationResponse() {}
onRetryAfter() {}
onReleaseAction() {}
onVisibilityChanged() {}
onUpdateAnalogControl() {}
onSettings() {}
onSavedData() {}
onOauthResponse() {}
onSettingsRequest() {}
```

Description
While receiving data from the system plugin will call the callbacks.

Part II: Internal Developer Documentation

Chapter 9. Plugin Manager Client library

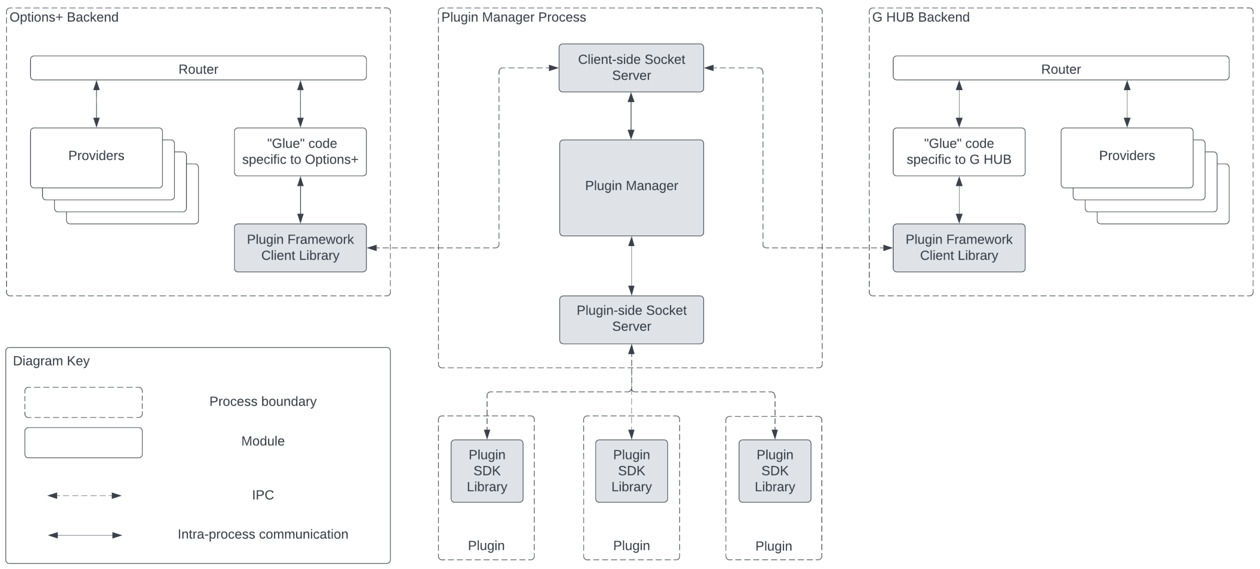

Figure 1. High-level Architecture of the Plugin Framework

9.1. Using the library

9.1.1. Distribution

The Plugin Manager client library is distributed as a conan package with the id `plugin_framework_client_cpp_sdk` in the `logi/stable` private Artifactory.

9.1.2. Entry Points

The header file `logi_plugin_communication.hpp` is the main entry point to the library. Using the library involves creating a class derived from `logi::plugin_framework::LogiPluginCommunication`\`.

9.1.3. Initialization

The `connect` method establishes a connection to the Plugin Manager process, and initiates the handshake protocol between the two sides. If the handshake is successful, the `on_communication_established` method will be called.

Note: On Windows, the library connects to the NamedPipe open by the Plugin Manager process: `\\\\.\\pipe\\logitech_plugin_framework_agent-<USER_NAME_HASH>`, while on macOS it connects to a UNIX Domain Socket (UDS): `/tmp/logitech_plugin_framework_agent-<USER_NAME_HASH>`, where `<USER_NAME_HASH>` is a unique hash based on the username of the currently logged in user.

9.2. API Reference

9.2.1. Initialization methods

connect()

Signature

```
void connect();
```

Description

Connect to the Plugin Manager agent

Parameters

None

Return values

None

disconnect()

```
void disconnect();
```

Description

Disconnect from the Plugin Manager agent

Parameters

None

Return values

None

connected()

```
void connected();
```

Description

Indicates whether the client library is connected to the Plugin Manager

Parameters

None

Return values

true, if the Plugin Manager is connected, false otherwise.

9.3. Plugin admin operations

9.3.1. install_plugin()

```
typedef std::function<void(result_code code, const std::string &text)> simple_result_handler;
result_code install_plugin(std::string plugin_id, simple_result_handler callback = 0);
```

Description

Silently installs a plugin

Parameters

plugin_id

id of the plugin to install

callback

asyncronous reply to the action that contains result code and optional description

Return values

Result code, as stated in error_codes sections, indicates whether sending message was successfull.

9.3.2. remove_plugin()

```
typedef std::function<void(result_code code, const std::string &text)> simple_result_handler;
result_code remove_plugin(std::string plugin_id, simple_result_handler callback = 0);
```

Description
Silently removes plugin

Parameters

- **plugin_id**: plugin_id id of the plugin to be removed
- **callback**: asyncronous reply to the action that contains result code and optional description

Return values
Result code, as stated in error_codes sections, indicates whether sending message was successfull.

9.3.3. plugin_list()

```
typedef std::function<
       void(const logi::protocol::plugin_manager::PluginDetails &plugins, result_code code, const std::string &text)>
       plugin_details_handler;
result_code plugin_list(plugin_details_handler callback);
```

Description
Returns list of installed plugins.

Parameters

- **callback**: callback that will return list of plugin

Return values

Result code, as stated in error_codes sections, indicates whether sending message was successfull.

9.4. Interacting with plugins

9.4.1. invoke()

```
typedef std::function<void(const logi::protocol::plugin_manager::PluginUpdateResult& update_result)>
update_result_handler;
result_code invoke(const logi::protocol::plugin_manager::Invoke &invoke, update_result_handler callback = 0);
```

Description
Call to invoke action or change value of analog control.

Parameters

- **invoke**: contains invoke structure that will have action id, plugin id, button state and rotation position (absolute value or delta)
- **callback**: after action will be returned this structure will contain update state of the action

Return values
Result code, as stated in error_codes sections, indicates whether sending message was successfull.

9.4.2. Handling events from plugins

```
virtual void on_action_state_changed(const logi::protocol::plugin_manager::PluginUpdateResult &update_result) {};
```

Description
Will be called when plugin is reporting a new state for an action.

```
virtual void on_plugin_available(const logi::protocol::plugin_manager::PluginDetails &plugin_details) {};
```

Description
Will be called when plugin is installed or started.

```
virtual void on_plugin_removed(const std::string& plugin_id) {};
```

Description
Will be called when plugin is uninstalled.

```
virtual void on_communication_established() {};
```

Description
Will be called when client library was connected to the plugin manager and handshake successfully finished

9.5. Plugin Configuration

9.5.1. plugin_action_configuration()

```
typedef std::function<void(const logi::protocol::plugin_manager::PluginActionConfiguration &action_configuration,
                               result_code code,
                               const std::string &text)>
        action_configuration_handler;
result_code plugin_action_configuration(
            const logi::protocol::plugin_manager::GetPluginActionConfigurationRequest &configuration_request,
action_configuration_handler
                callback);
```

Description
Returns configuration that is required for particular action.

Parameters

- **configuration_request**: indicates action and plugin to get configuration.
- **callback**: callback that will return configuration of plugin

Return values
Result code, as stated in error_codes sections, indicates whether sending message was successfull.

9.5.2. set_plugin_action_configuration()

```
typedef std::function<void(const logi::protocol::plugin_manager::PluginActionConfiguration &action_configuration,
                               result_code code,
                               const std::string &text)>
        action_configuration_handler;
result_code set_plugin_action_configuration(
            const logi::protocol::plugin_manager::SetPluginActionConfigurationRequest &configuration,
            action_configuration_handler callback = 0);
```

Description
Submit action configuration. This method will be called from UI when user fill the settings form

Parameters

- **configuration**: indicates action and it’s configuration JSON from UI
- **callback**: callback that will return configuration of plugin

Return values
Result code, as stated in error_codes sections, indicates whether sending message was successfull.

9.5.3. action_status()

```
typedef std::function<void(const logi::protocol::plugin_manager::PluginUpdateResult& update_result)>
update_result_handler;
result_code action_status(logi::protocol::plugin_manager::AnalogControlID action_id, update_result_handler callback);
```

Description
Request status for analog control or action

Parameters

- **action_id**: indicates exact action/analog control
- **callback**: will return state of action/analog control

Return values
Result code, as stated in error_codes sections, indicates whether sending message was successfull.

9.5.4. invoke_login()

```
typedef std::function<void(result_code code, const std::string &text)> simple_result_handler;
result_code invoke_login(const std::string& plugin_id, simple_result_handler callback = 0);
```

Description
Start login procedure.

Parameters

- **plugin_id**: id of the plugin
- **callback**: simple handler that have result of login procedure

Return values
Result code, as stated in error_codes sections, indicates whether sending message was successfull.

9.5.5. invoke_logout()

```
typedef std::function<void(result_code code, const std::string &text)> simple_result_handler;
result_code invoke_logout(const std::string &plugin_id, const std::string& username, simple_result_handler callback =
0);
```

Description
Perfrom logout for given plugin and user.

Parameters

- **plugin_id**: id of the plugin
- **username**: username of the user to logout
- **callback**: simple handler that have result of logout procedure

Return values
Result code, as stated in error_codes sections, indicates whether sending message was successfull.

9.5.6. development_mode()

```
typedef std::function<void(bool development_mode, result_code code, const std::string &text)> development_mode_handler;
result_code development_mode(development_mode_handler callback);
```

Description
Asyncronously returns whether development mode is enabled.

Parameters

- **callback**: handler that will return whether dev mode is enabled or if there are any errors calling the action

Return values
Result code, as stated in error_codes sections, indicates whether sending message was successfull.

9.5.7. set_development_mode()

```
typedef std::function<void(result_code code, const std::string &text)> simple_result_handler;
result_code set_development_mode(bool development_mode, simple_result_handler callback = 0);
```

Description
Asyncronously set development mode for plugin manager.

Parameters

- **callback**: handler that will return whether operation was sucessfull.

Return values
Result code, as stated in error_codes sections, indicates whether sending message was successfull.

9.5.8. set_language_changed()

```
typedef std::function<void(result_code code, const std::string &text)> simple_result_handler;
result_code set_language_changed(const std::string& language_code, simple_result_handler callback = 0);
```

Description
Send the plugin manager the language change event, so that it could change internal localisation state.

Parameters

- **callback**: returns the operation status.

Return values
Result code, as stated in error_codes sections, indicates whether sending message was successfull.

9.5.9. set_visibility()

```
typedef std::function<void(const logi::protocol::plugin_manager::PluginStateList &update_result)>
state_result_handler;
result_code set_visibility(const logi::protocol::plugin_manager::PluginActionVisibilityUpdateList visibilityUpdate,
            state_result_handler callback = 0);
```

Description
Change visibility settings for the list of actions/analog controls

Parameters * **visibilityUpdate** contains list of action/analog control with respective visibility value * **callback**: returns the action/analog control state for the each visibility request.

Return values
Result code, as stated in error_codes sections, indicates whether sending message was successfull.

Chapter 10. Marketplace Client Protocol

This section describes the message protocol between the plugin manager and the marketplace web view gallery.

Communication between marketplace and plugin manager is asynchronous. This means that you can send request from web and receive response on the channel without binding between them.

Request: `window.PMCall("{}", "MPGetInstalledPlugins")` Responces can be : `window.PMReceive.addEventListener('message', function (msg) {console.log("plugin msg: "+msg.data)})` Response format is "MessageNameWithoutSpaces <message body right after single space>"

Chapter 11. Types

message PluginInfo { string plugin_id = 1; optional string version = 2; optional bool login_required = 3; optional string description = 4; }

message PluginInstallation { repeated PluginInfo installed_plugins = 1; }

11.1. Updates on plugin installation status

PluginInstallation will be broadcasted to PMReceive event listener - if any change to plugin list will be made. E.g. user installs the plugin using web link - PM Marketplace if opened will receive MPInstalledPlugins with JSON. - if plugin will request login. E.g. plugin requests to login.

11.2. Install plugin

Request: MPInstallPlugin with PluginInfo json. Return: PluginInstallation will be sent if any change to plugin list will be made.

11.3. Update plugin

Request: MPUpdatePlugin with PluginInfo json. Return: noreturn

11.4. Login

Request: MPLogin with PluginInfo json. Return: noreturn

11.5. Get installed plugins

Request: MPGetInstalledPlugins with empty {} json. Return: MPInstalledPlugins

Part III: Appendix

Changelog

Version 0.0.999.0

Changes

Bugs fixed