JAKA

# AddOn Development Guide

JAKA AddOn

English Edition

Document Version: V1.7.2

Product: JAKA AddOn

Type: AddOn Development Guide

# Contents

**1 About AddOn** **2**
1.1 What is AddOn . . . 2
1.2 Why Use AddOn . . . 3
1.3 Basic Functions of AddOn . . . 4
1.4 AddOn Operation Mechanism . . . 4
1.5 AddOn File Composition . . . 8
1.6 Feedback . . . 9

**2 Development Guide** **10**
2.1 Getting Resources . . . 10
2.2 Before You Start . . . 10
2.3 Overview of AddOn Development Tutorials . . . 10
2.3.1 Beginner Tutorials . . . 10
2.3.2 Advanced Tutorials - Specific Cases . . . 11
2.4 Related Reference Documents . . . 11

**3 AddOn Development Process Overview** **12**
3.1 Preparations Before Development . . . 12
3.1.1 Setting Up the Development Environment . . . 12
3.2 Start Developing AddOn . . . 12
3.2.1 Developing Custom Services . . . 13
3.2.2 Developing Custom Commands . . . 13
3.2.3 Developing Custom Webpages . . . 13
3.2.4 Language Setting . . . 13
3.2.5 Writing User Guide . . . 14
3.3 Release . . . 14

**4 Setting up Development Environment** **15**
4.1 Install the JAKA Virtual Machine . . . 15
4.1.1 Step One: Install Virtual Machine Software . . . 15
4.1.2 Step Two: Install the JAKA Mirror File . . . 15
4.1.3 Step Three: Start the JAKA Virtual Controller . . . 16

4.2 Install the AddOn Suite **[change]** (Controller version v1.7.1.28 comes with AddOn package, you just need to install the corresponding controller version and App version.) . . . . 18
4.2.1 Step One: Install JAKA App . . . . 18
4.2.2 Step Two: Connect the Robot Using the App . . . . 19
4.2.3 Step Three: Upgrade the Controller . . . . 20
4.2.4 Step Four: Install the AddOn Suite **[abandoned]** . . . . 20
4.2.5 Step Five: Restart the App . . . . 21

**5 Custom Command Development 22**
5.1 About Custom Commands . . . . 22
5.1.1 What are Custom Commands . . . . 22
5.1.2 Purpose of Custom Commands . . . . 22
5.2 Creating an AddOn Custom Command Package . . . . 22
5.2.1 Step One: Create an AddOn Folder . . . . 23
5.2.2 Step Two: Package & Upload . . . . 24
5.2.3 Step Three: Develop Custom Commands . . . . 25
5.2.4 Step Four: Run the Developed Add-on . . . . 28
5.3 Debugging . . . . 28
5.4 FAQ . . . . 31

**6 Custom Service Development 35**
6.1 About Custom Services . . . . 35
6.1.1 What is a Custom Service . . . . 35
6.1.2 Purpose of Custom Services . . . . 35
6.2 Create AddOn Custom Service Package . . . . 35
6.2.1 Step One: Create AddOn Folder . . . . 35
6.2.2 Step Two: Package & Upload . . . . 36
6.2.3 Step Three: Develop Custom Service . . . . 38
6.3 Testing . . . . 39

**7 Custom Webpage Development 41**
7.1 About Custom Webpages . . . . 41
7.1.1 What Are Custom Webpages . . . . 41
7.1.2 Purpose of Custom Webpages . . . . 41
7.2 Creating JAKA_Web . . . . 41
7.2.1 Method One: Use Dashboard Template . . . . 42
7.2.2 Method Two: Use Front-end Standard Engineering . . . . 47
7.3 Compatibility . . . . 50

**8 Multilanguage Setting 51**

8.1 Multilanguage Setting for Custom Commands . . . 51
8.2 Multilanguage Setting for Custom Interfaces . . . 51

**9 Writing User Guide 53**
9.1 How to publish the manual . . . 54

**10 Use JAKA Dashboard Template 55**
10.1 Import the Template . . . 55
10.2 What You Should Know About Dashboard . . . 55
10.2.1 Layout . . . 55
10.3 List of Templates . . . 56
10.3.1 Control the Program . . . 56
10.3.2 Check the Status of the Program . . . 57
10.3.3 Control the Power Supply . . . 57
10.3.4 Robot Status . . . 58
10.3.5 Jog the Robot . . . 60
10.3.6 Get the Robot Position . . . 60
10.3.7 System Variables . . . 61
10.3.8 I/O Panel . . . 62

**11 Release 63**

**12 Case Study - DH_Grapper 64**

**13 Case Study - Ewellix LiftKit 65**

**14 AddOn Configuration File Specification 66**
14.1 1. File Encoding Requirement . . . 66
14.2 2. File Naming Requirement . . . 66
14.3 3. File Content . . . 66
14.4 4. Field Descriptions: . . . 66

**15 Interaction between AddOn Web Pages and the App 69**
15.1 JAKAController.jakaCall() . . . 69
15.2 window.jakaCallBack() . . . 70
15.3 window.changeLanguageHandler() . . . 71

Controller version: 1.7.2

# 1 About AddOn

**This page aims to explain the basic concepts for AddOn developers.**

**Warning:** Please note:

1. The AddOn mentioned in this tutorial refers to AddOn 3.0 by default.
2. AddOn 3.0 development and usage are supported on JAKA controllers > 1.7.1.x _X64 version.
3. AddOn 1.0 and 2.0 are no longer maintained and provided with development support, but they still work on version 1.7.1.x..

**Tip:** Tip:

This document is intended for readers who have received basic robotics training, have some basic development skills, and are familiar with JAKA' s script language `JKS`.

In addition, experience with programming languages such as node-red, JavaScript, HTML5, CSS3, etc. will be helpful for users to develop and use AddOn.

## 1.1 What is AddOn

JAKA AddOn, as its name, is an add-on software provided by JAKA. It supports users in developing custom command add-on, custom service add-on, custom interface add-on, and composite add-on that includes the above types.

By using AddOn, users can significantly extend the functionality of the robot operating system or customize the existing functionalities according to their preferences.

**Tip:** For distinction, two terms are defined here:

- JAKA AddOn Kit **[abandoned]**: The kit package required during the development environment setup, namely jaka_addon_kit.tar.gz.
- JAKA AddOnXXX Package: A developed package with specific functionality, such as JAKA AddOn Palletizing Package, JAKA AddOn DH Gripper Package, JAKA AddOn Visual Guidance Package, JAKA AddOn Force Control Package, and so on.

**Warning:** Please note:

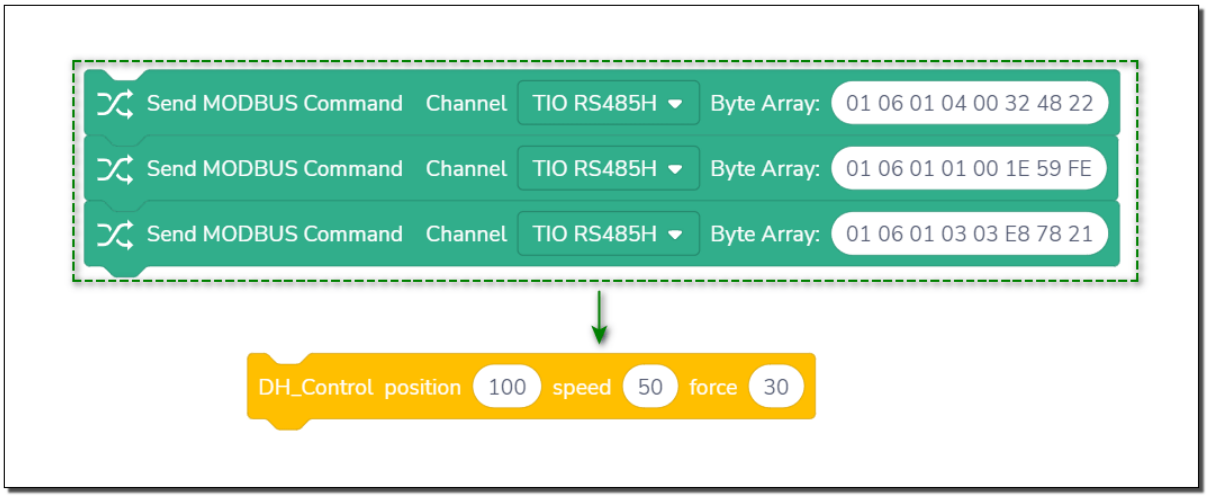
The AddOn package can only be named by English characters and underlines.

## 1.2 Why Use AddOn

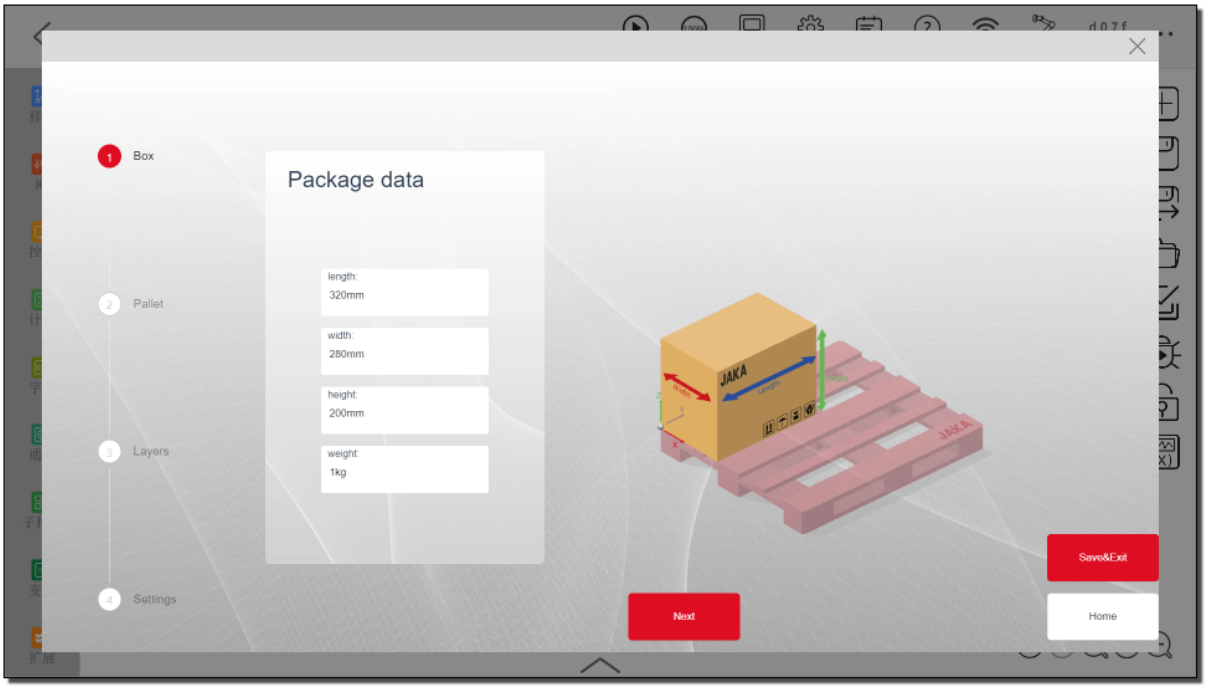
In typical scenarios, when a new third-party tool is connected at the end of the robot, or when performing specific processes (such as palletizing, welding, etc.) in specific usage scenarios, new commands need to be set in the programming interface.

These commands often contain complex, lengthy codes and parameters that are difficult to understand and reuse, adding extra burden to simple actions or programs. For engineers who are proficient in specific crafting processes but not well-versed in robot programming, setting commands is a challenge.

Using AddOn can de-code and greatly simplify the process of setting instructions. It omits the complex settings in the programming step and encapsulates them into a simple, clear, and easy-to-understand instruction, which is easier to operate and more conducive to re-use the programmed instructions, thus fully enhancing efficiency and productivity.




On the other hand, the Process Package AddOn enables users to focus on the execution of specific processes rather than how the instructions are used, which has largely reduced the learning cost and difficulty of use for users.




In summary, using AddOn make it easier and more effective to collaborate with robots.

## 1.3 Basic Functions of AddOn

As mentioned earlier, AddOn has three basic functions: developing custom command add-ons, custom service add-ons, and custom webpage add-ons.

The implementation of these functions empowers the existing systems, allowing users to customize new functions as needed or make changes and upgrades to the existing ones.

JAKA's application development team can develop official JAKA application packages, process packages, and process kits through the AddOn framework for users to download and use. The team can also assist JAKA customers in developing custom functions, services, and webpages to meet their customization needs.

JAKA partners can develop ecosystem-standard add-ons through the AddOn framework to enhance product compatibility and usability.

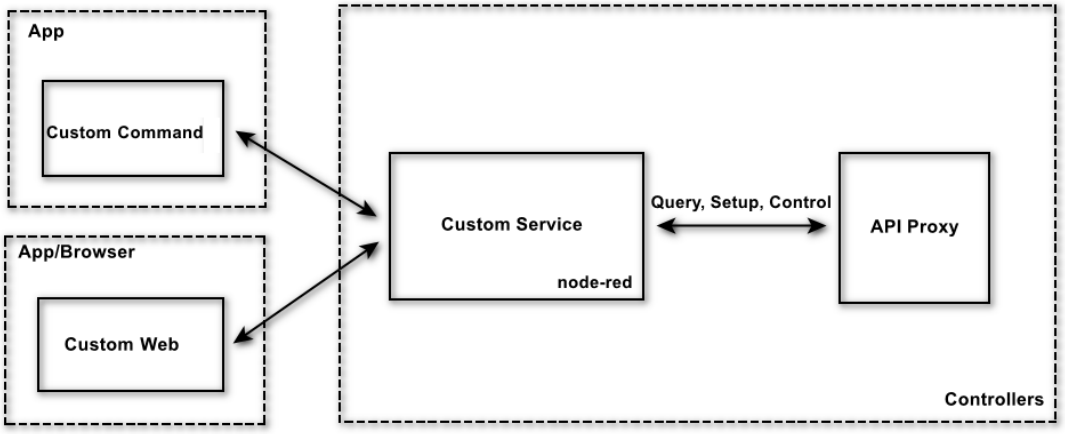
JAKA robot end users can develop custom commands through the AddOn framework to encapsulate and reuse commonly used commands.

## 1.4 AddOn Operation Mechanism

- AddOn Custom Services

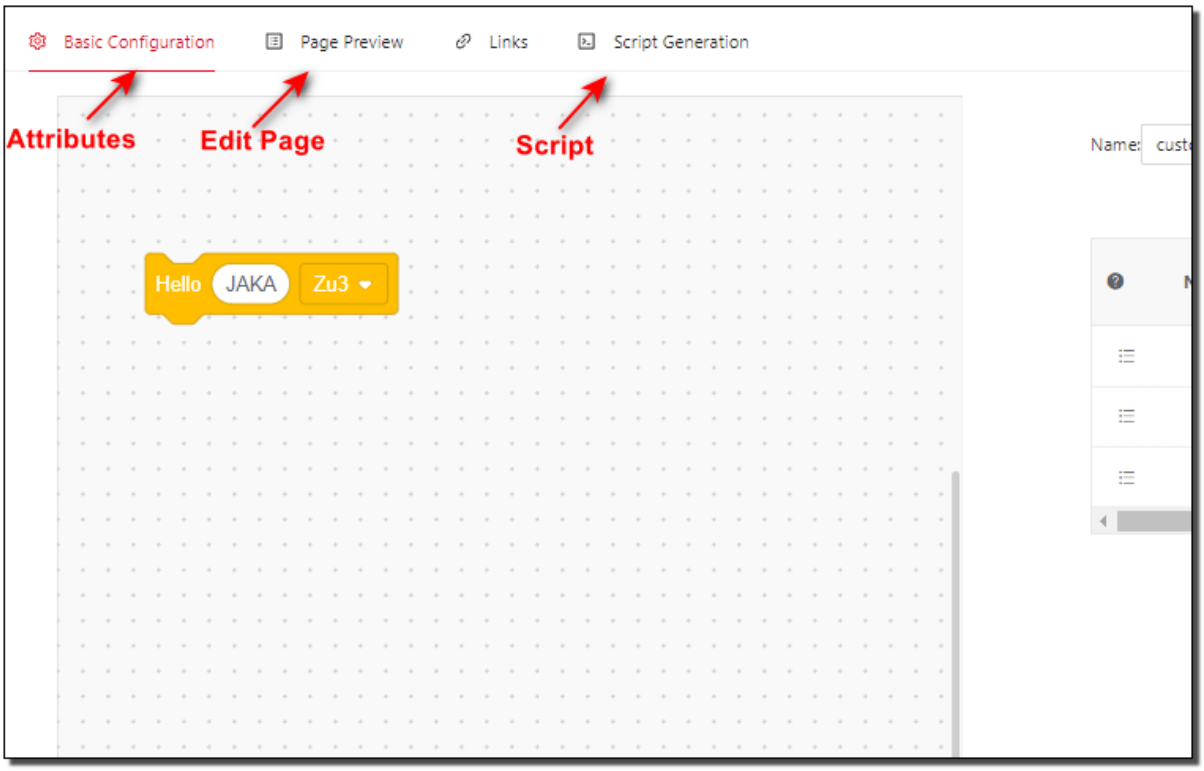
AddOn custom services are developed on the node-red platform and will be run in the controller. When an AddOn is started, the controller starts the node-red process for that AddOn. The custom service can be used as an intermediate layer to provide services for custom commands and custom webpages. The relevant

nodes provided in node-red can be used to query robot information, set robot parameters, and control robot motion.

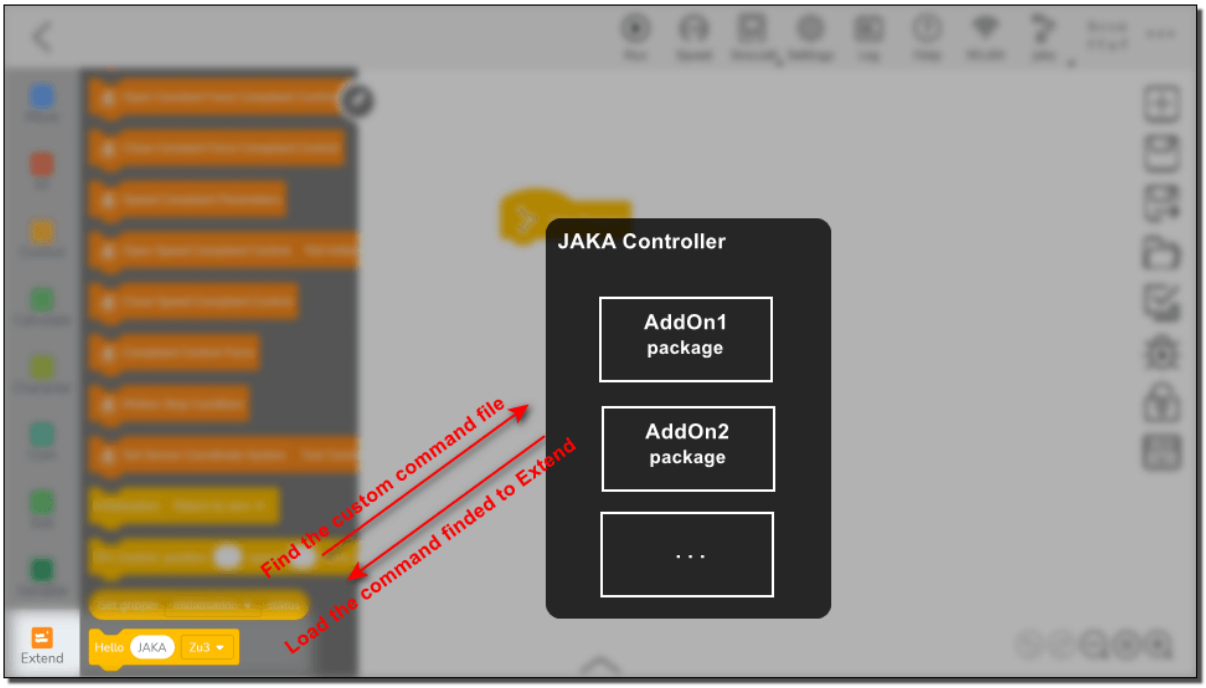



- AddOn Custom Commands

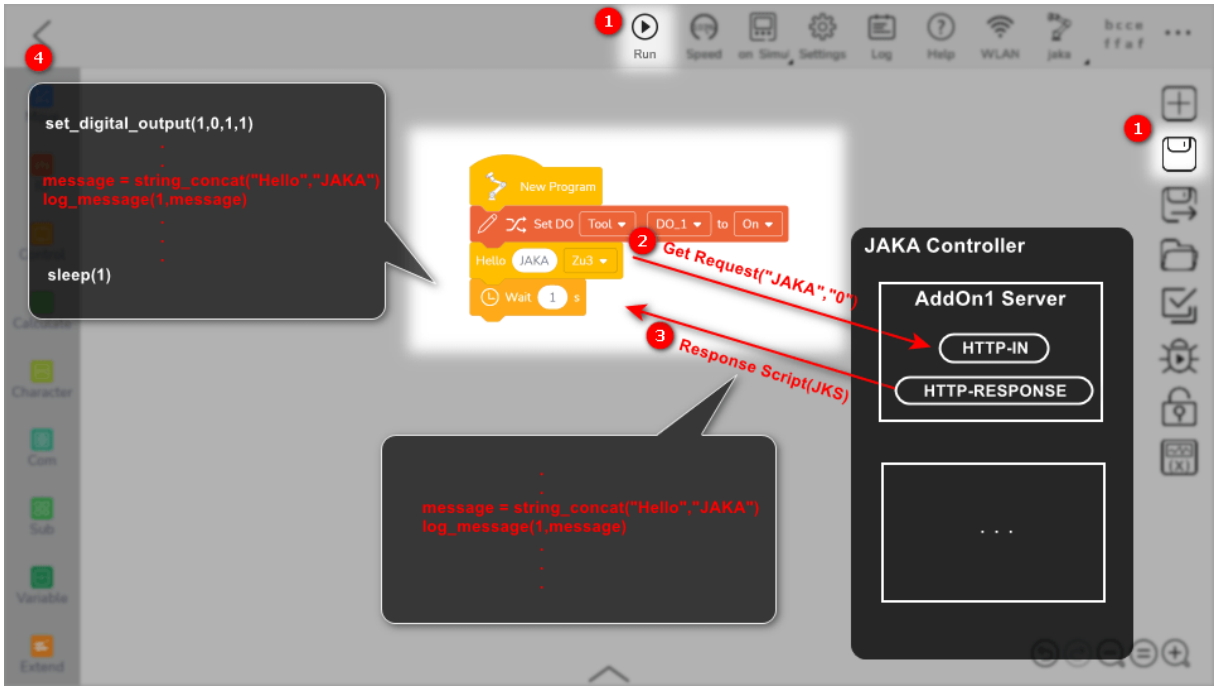
A custom command block consists of three parts, attributes, edit page and script, all of which are defined in the AddOn development tool (node-red).




When the App starts, it looks for the data for the custom command block from the AddOn package. If found, the custom command block will be added into the Program-Extensions bar.

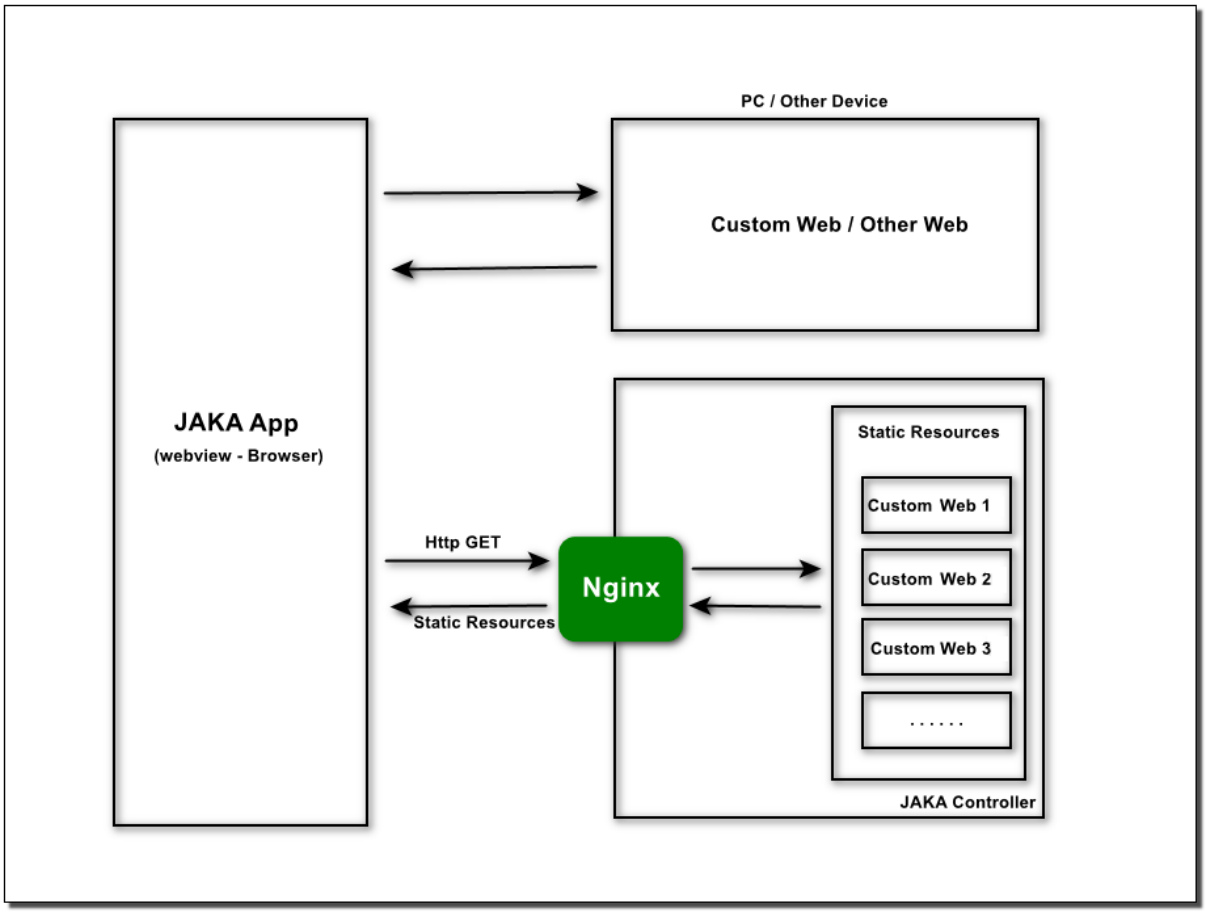



After the custom command block is inserted into the application and when the user clicks Save or Run, the App would make a request to the AddOn service, while the parameters on the command block would be send with it. The AddOn service receives the request, splices the parameters into the script of the custom command, and returns it to the App. Finally, the App inserts the script it receives.

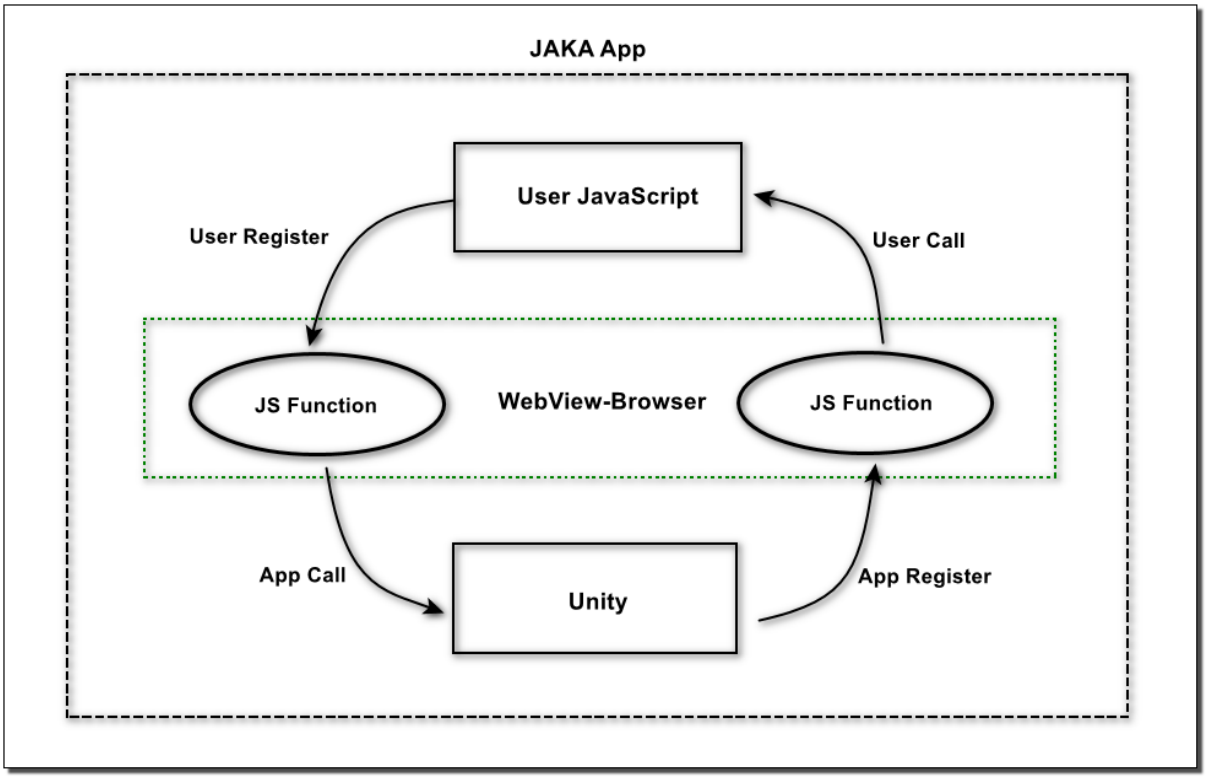



- AddOn Custom Webpages

Custom webpages are usually deployed in a JAKA controller and are proxied through Nginx. It is also possible to deploy them on personal computers or other devices.




Custom webpages can be used not only to access AddOn custom services or services from other devices, but also to interact with the JAKA App by using JavaScript methods injected into the web page by the JAKA App.




## 1.5 AddOn File Composition

The completed AddOnXXX package may contain various files, and below are the most common ones:

```
**Data File**
An AddOn package generally contains a ``flows.json`` file that includes the␣
↪definition of flows and nodes in node-red. The name of this file can be␣
↪customized but needs to be pointed to in the configuration.
```

```
**Configuration File**
Any type of add-on must include an ``xxx_config.ini`` file, i.e.,  :doc:`AddOn␣
↪Configuration File <14_addon_configuration>` . Only packages containing the AddOn␣
↪configuration file can be installed, initialized, and run.
```

```
**Frontend Project**
Under the AddOn directory containing custom pages, there is usually a folder for␣
↪the frontend project, such as client or dist. It is recommended to use "client"␣
↪as the frontend project folder.
```

## 1.6 Feedback

We appreciate your feedback!

If you find any wrong or unclear descriptions in the document, please feel free to submit Issues in our document repository.

# 2 Development Guide

**This page aims to provide AddOn developers with a learning path guide for AddOn development.**

## 2.1 Getting Resources

Developing add-ons requires a series of resources. Here, essential resources for JAKA AddOn development are listed and will be continuously updated to facilitate developers in obtaining complete and up-to-date content.

Links to these resources will be provided again in the subsequent chapters where they are needed. This section serves as a reminder and summary.

1. JAKA Virtual Machine
2. AddOn Kit Package **[abandoned]** (Controller versions after v1.7.1.28 come with the AddOn Kit package, which does not need to be downloaded and installed.)
3. Demos in AddOn development templates and tutorials visit the repository **[under development]**

**Tip:** Note

The kit package will include the latest versions of matching Apps and controllers.

## 2.2 Before You Start

The target reader of this tutorial should have received basic robot operation training and owned a certain level of basic development skills. They should also be familiar with JAKA's script language `JKS`.

In addition, knowledge of programming languages such as node-red, JavaScript, HTML5, CSS3, etc., will be even more helpful for users to develop and use AddOn effectively.

## 2.3 Overview of AddOn Development Tutorials

### 2.3.1 Beginner Tutorials

If you're unsure about where to start, begin by learning the following tutorials:

- Quickly understand the complete process of AddOn development in the *Quick Start* section.
- Set up the AddOn development environment by following the steps in *Setting Up AddOn Development Environment* .

- Learn the basic process and rules of developing custom command AddOn by creating "*JAKACommand* ".
- Learn the basic process and rules of developing custom service AddOn by creating "*JAKAServe* ".
- Learn the basic process and rules of developing custom interface AddOn by creating "*JAKAweb* ".

### 2.3.2 Advanced Tutorials - Specific Cases

The purpose of beginner tutorials is to provide developers with a clear understanding of the process, so they only cover basic functionalities, lacking practical application scenarios.

Advanced tutorials are combined with specific scenario cases, allowing developers to learn about functionalities not mentioned or briefly mentioned in the beginner tutorial.

After understanding the basic development process of AddOn, you can choose a specific application case for learning. These cases include at least two out of three functionalities: custom instruction, custom service, and custom interface, collectively referred to as composite AddOn:

**Tip:** More cases are on the way···

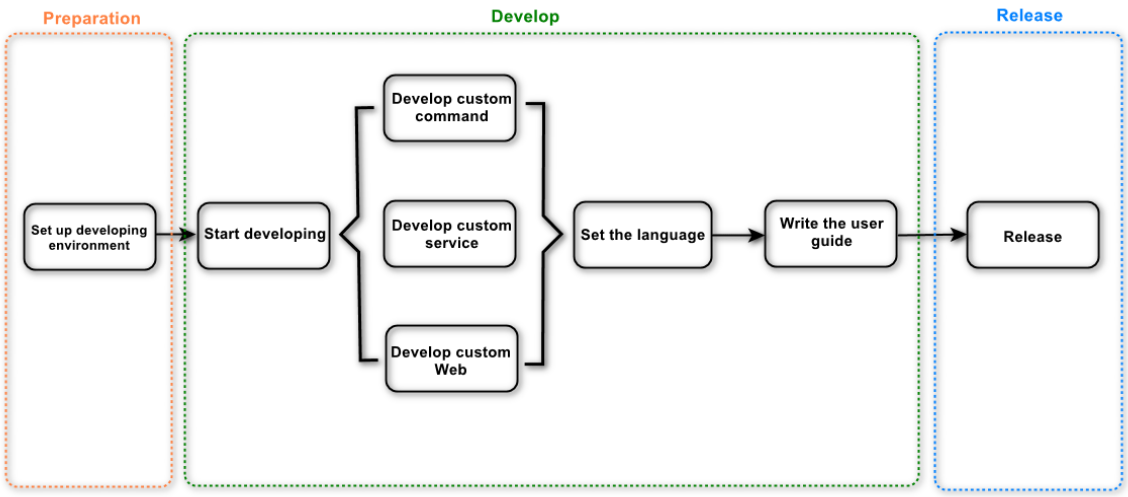
## 2.4 Related Reference Documents

- JAKA Script: Scripting language used to implement custom command.
- *Configuration Files* : Field descriptions for AddOn profiles.
- *JAKA App APIs* : Provides JavaScript functions for interaction between the front end and JAKA App.
- node-red: Development platform and backend service for AddOn.
- JavaScript: Used for scripting and logic in node-red.
- Front-end Technologies: Requires a certain level of front-end technology for developing custom page plugins.

# 3 AddOn Development Process Overview

**This page aims to provide AddOn developers with a brief overview of the development process for an AddOnXXX package, including the basic steps involved in each stage.**

**Tip:** Process Flow:




As shown in the above diagram, the complete development process for the AddOnXXX package consists of three main stages and six major steps. We can break down them into many sub-steps.

The following sections will provide a brief explanation of this process diagram. For detailed explanations and operating instructions of the sub-steps, please refer to the specific chapters in the tutorial.

## 3.1 Preparations Before Development

### 3.1.1 Setting Up the Development Environment

This step aims to provide the necessary development environment for the AddOnXXX package, including installing a virtual machine, upgrading controllers, and upgrading software versions.

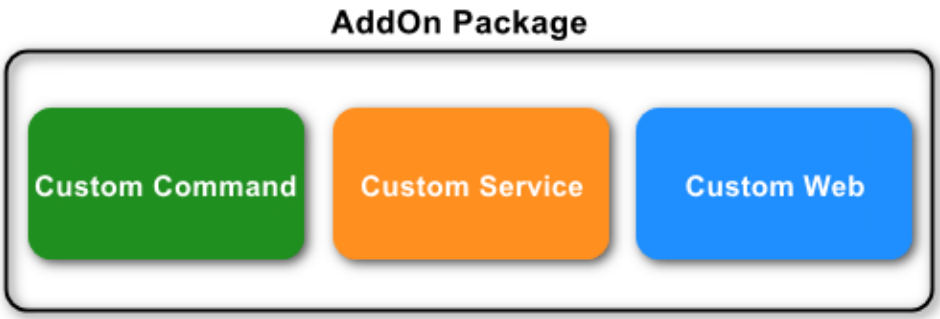
For detailed information, please refer to *Setting Up the Development Environment* .

## 3.2 Start Developing AddOn

After setting up the environment, developers can choose the type they need and refer to the corresponding pages for more detailed instructions.

As mentioned earlier, JAKA AddOn has three functions: custom commands, custom services, and custom webpages.

An AddOnXXX package can include at least one function.




### 3.2.1 Developing Custom Services

The user can run a service in the controller for querying the robot status, setting robot parameters, controlling robot motion, or customizing some functions. The service can run in the background alone, or can be used for custom web pages or custom commands. For details and procedures, see: *custom service development*

### 3.2.2 Developing Custom Commands

The custom command function allows developers to create commands other than the standard commands within the JAKA App. For detailed content and operation steps, please refer to: *Developing Custom Commands* .

### 3.2.3 Developing Custom Webpages

The custom interface function enables developers to create interfaces that meet custom requirements. For detailed content and operation steps, please refer to: *Developing Custom Interfaces* .

### 3.2.4 Language Setting

JAKA AddOn supports several languages. After developing a specific AddOn package, you can set the language according to your needs.

Currently supported languages include: Chinese, English, and Japanese.

For detailed information, please refer to: *Language Setting* .

### 3.2.5 Writing User Guide

Here, “user guide”refers to the documentation that developers need to provide to their users after developing add-ons of a specific applications or with a particular function using JAKA AddOn. This step aims to guide developers on what content should be included in these user guide, what rules to follow, and how to release them for user download or use.

For detailed information, please refer to *Writing User Guides* .

## 3.3 Release

The term “release” here refers to how developers should publish add-ons of a specific applications or with a particular function to provide them to their users for use.

The “release” function is currently under development. After completion, more detailed instruction will be synchronously updated on this website.

# 4 Setting up Development Environment

**This page aims to guide you on how to set up the AddOn development environment. Before you begin, please download the :doc:`development resources <02_about_development>` .**

Developing AddOns requires an appropriate development environment. Related softwares and Apps are essential, including the JAKA virtual machine and compatible software and controller versions.

**Tip:** Tip:

If you already got a controller, then the virtual machine is not needed since you can directly use that controller to develop. Click *InstallTheAddOnSuite* to follow the instructions on installing and upgrading the controller and JAKA App, and then to install the AddOn suite.

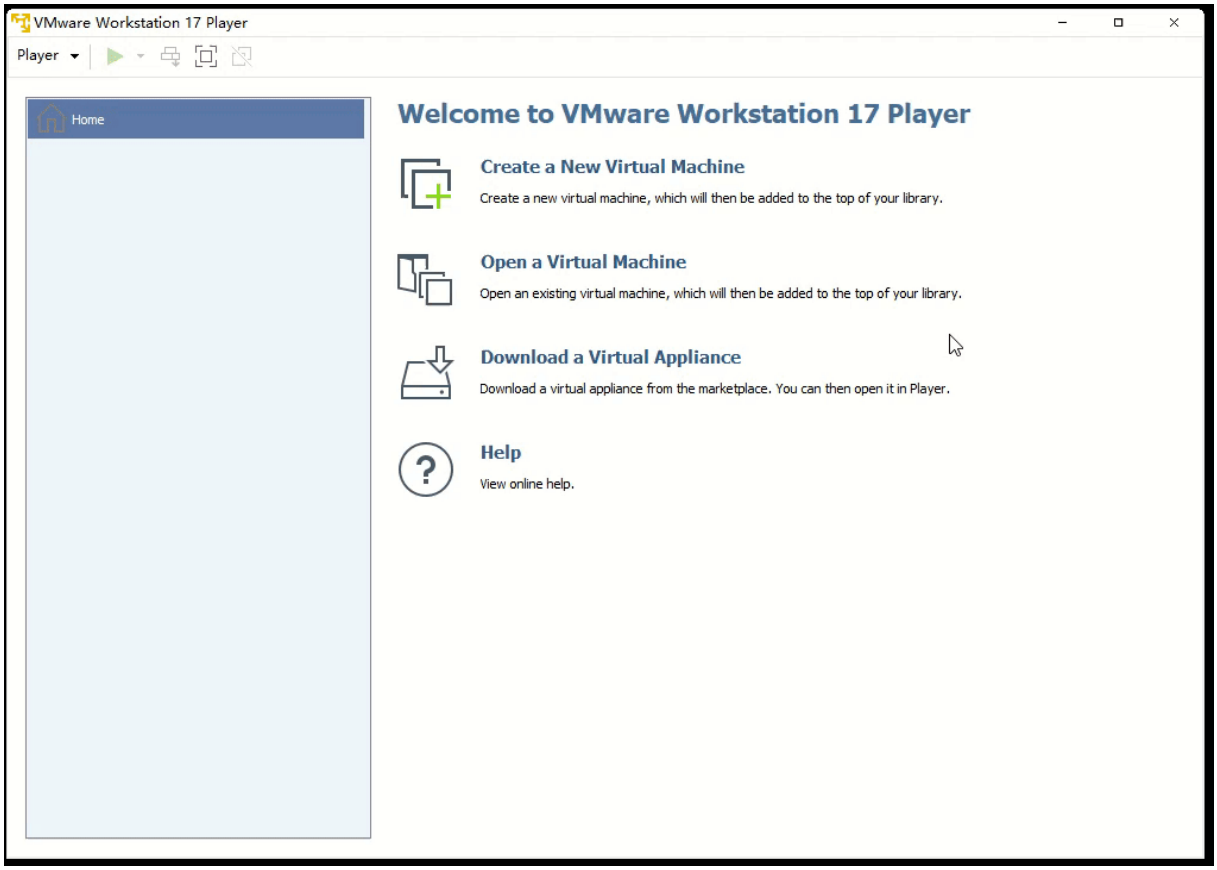
## 4.1 Install the JAKA Virtual Machine

### 4.1.1 Step One: Install Virtual Machine Software

Developers need to first install virtual machine software, such as VirtualBox or VMware Player.

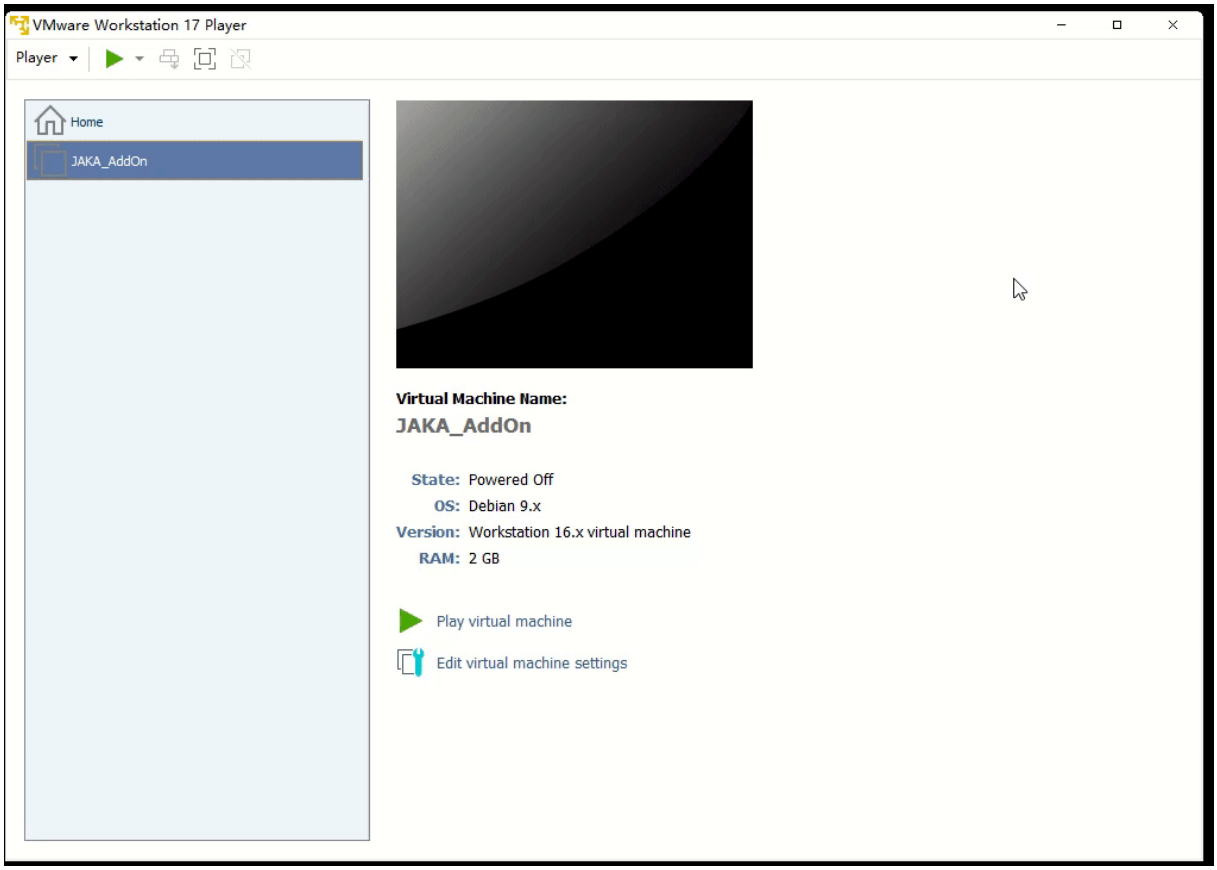
### 4.1.2 Step Two: Install the JAKA Mirror File

Locate the virtual machine mirror file in the *jaka_addon_package* and import it for installation.

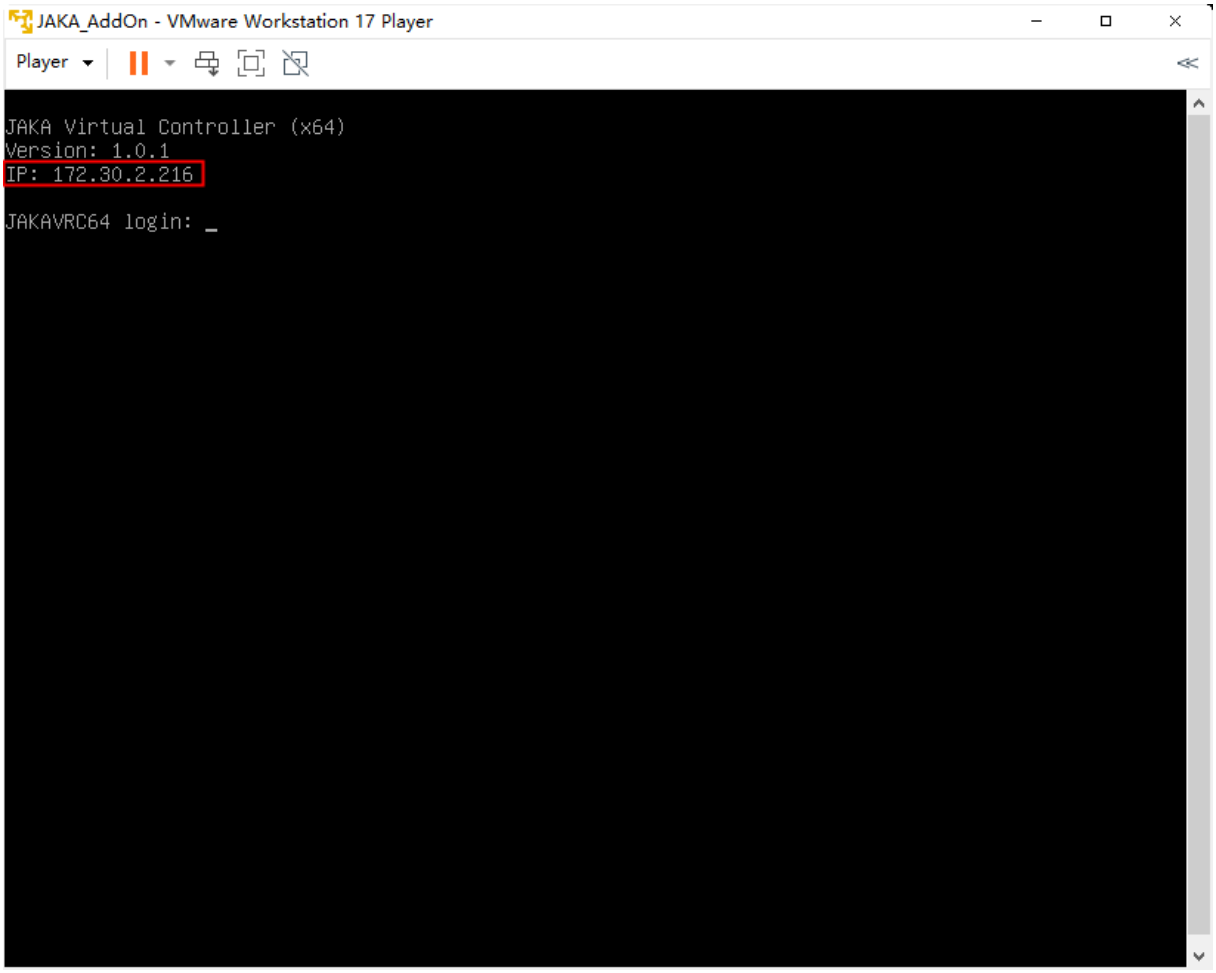



### 4.1.3 Step Three: Start the JAKA Virtual Controller

Click “play virtual machine” to launch it.




At this point, the virtual machine page will display the current robot’ s IP address.




## 4.2 Install the AddOn Suite [change] (Controller version v1.7.1.28 comes with AddOn package, you just need to install the corresponding controller version and App version.)

**Tip:** Please note:

Apps and controllers that are not updated cannot support the AddOn function.

After setting up the virtual machine, you need to upgrade the App and controller to the compatible versions before the AddOn development.

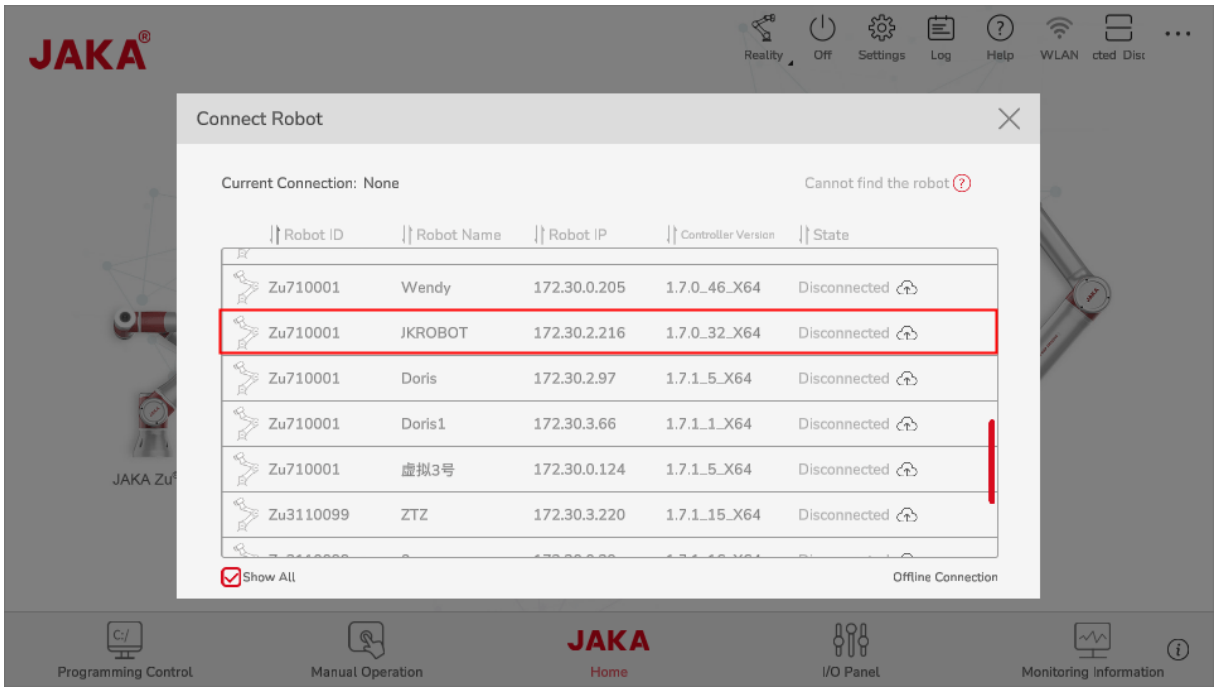
### 4.2.1 Step One: Install JAKA App

In the App folder of the AddOn development environment package, find the installation package and follow the guide to complete the installation of the App.

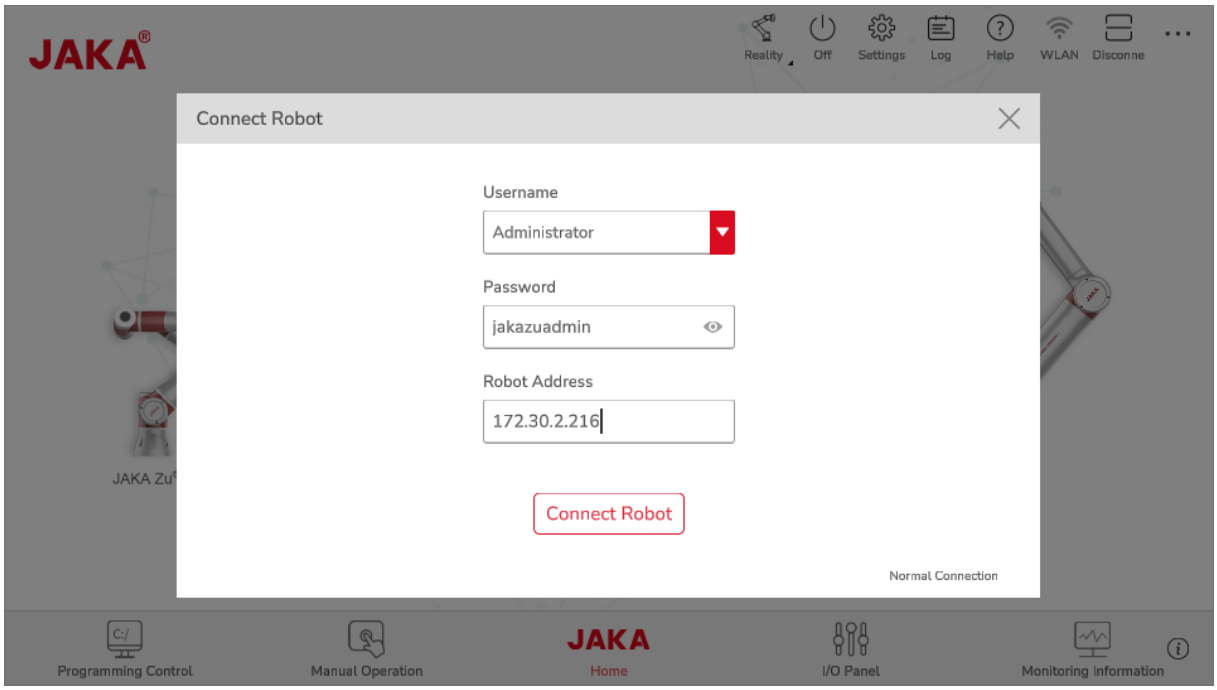
During the installation process, be sure to grant network permissions.

### 4.2.2 Step Two: Connect the Robot Using the App

1. Open the App to connect to the robot. At this point, you should be able to find the virtual robot in the robot list based on its IP.




2. If you cannot find the robot in the list, try using offline connection.
3. For the password, enter the default login password (jakazuadmin).
4. For the address, enter the IP of the virtual robot.

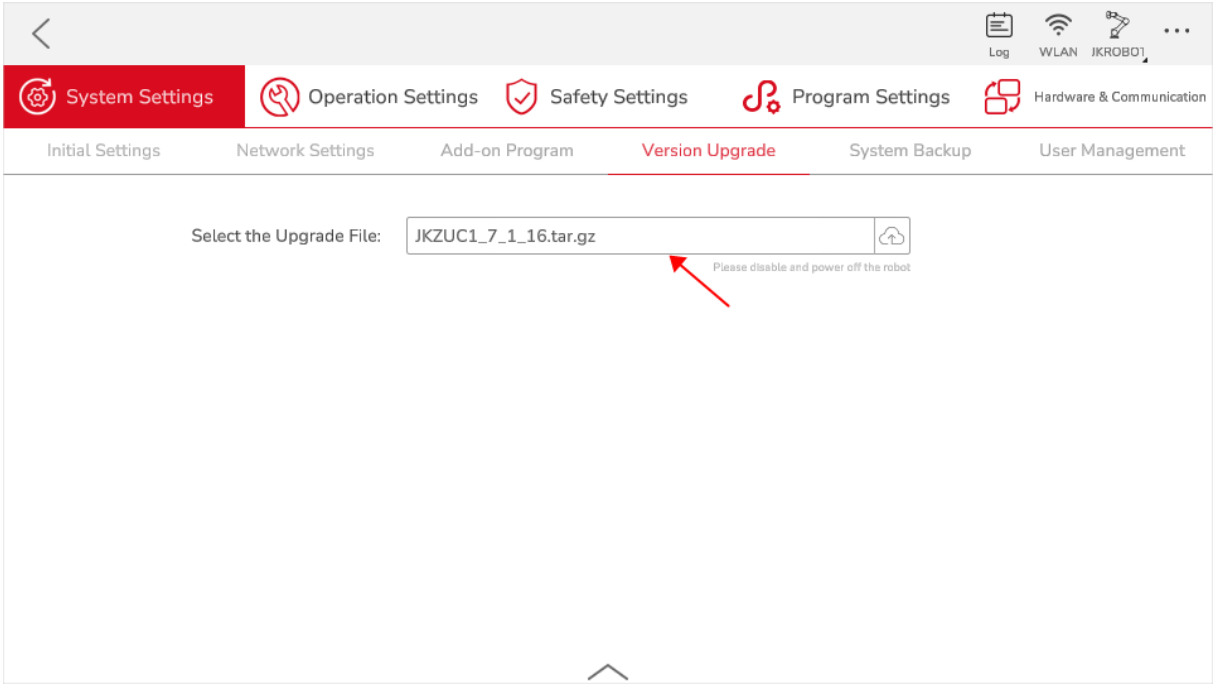



**Tip:** Tip:

If the App is unmatched with the controller' s version, thus the connection has failed, please install the matched version of App first, then upgrade the controller following instructions in Step Three.

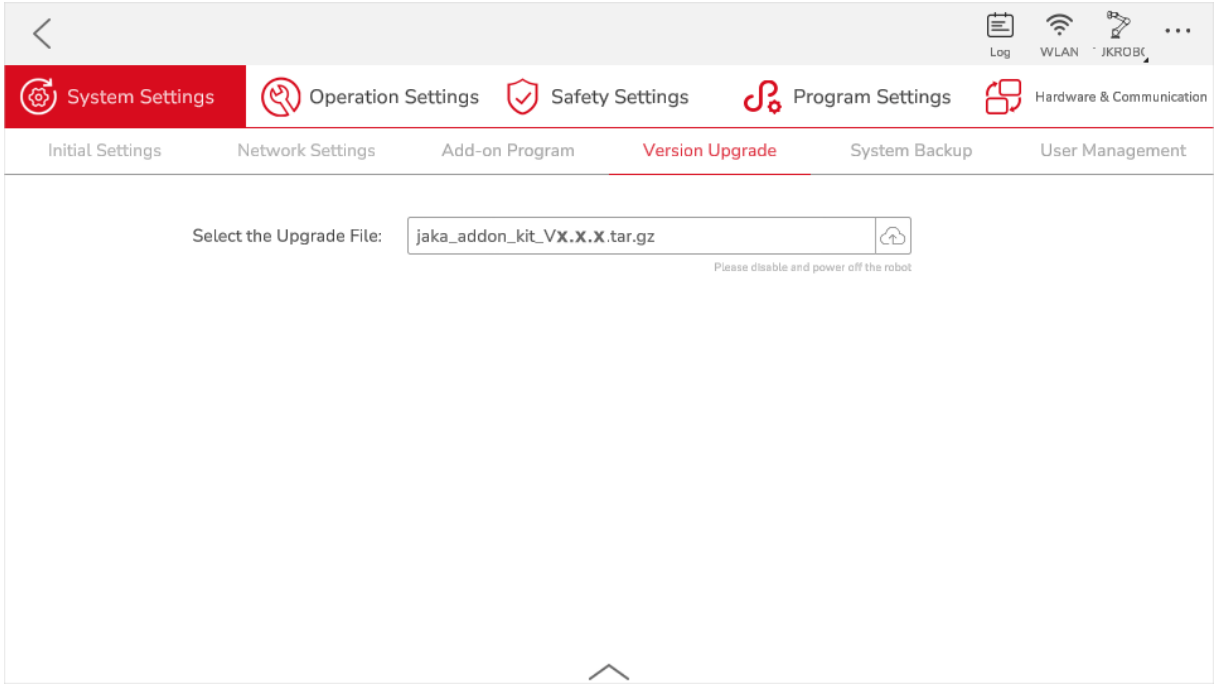
### 4.2.3 Step Three: Upgrade the Controller

Open the settings page, go to System Settings, then the Version Upgrade page. Select the controller package you want to upgrade, click upload, and complete the upgrade.




### 4.2.4 Step Four: Install the AddOn Suite [abandoned]

Open the settings page, go to System Settings, then the Version Upgrade page. Select the AddOn Suite, click upload, and complete the installation.




### 4.2.5 Step Five: Restart the App

**Tip:** Please note:

After completing steps one to four, please restart the App to take effect.

# 5 Custom Command Development

**This page aims to guide developers on how to develop custom command add-ons.**

**Tip:** Please note:

This chapter serves as an introductory tutorial, primarily to familiarize developers with the basic knowledge and development process of custom commands.

For information on how to develop AddOn packages with specific function for specific scenarios, please refer to the subsequent case chapters.

## 5.1 About Custom Commands

### 5.1.1 What are Custom Commands

Custom commands refer to commands that are not in the JAKA App commands directory, but in compliance with JAKA script rules, and developed by developers according to their own needs.

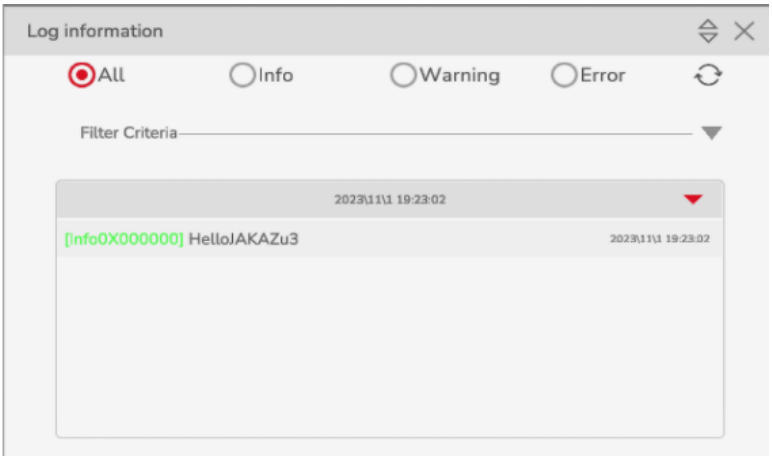
### 5.1.2 Purpose of Custom Commands

As mentioned in the introduction, when a robot' s end effector is connected to a new third-party tool, or when it is used to operate specific processes in a specific work scenario, it often requires writing complex commands and setting numerous parameters in the JAKA App programming interface. This step is complex and not easily reusable, so it is necessary to encapsulate these complex, diverse commands and parameters into a single, straightforward, easy-to-use command block to save time and improve efficiency.

## 5.2 Creating an AddOn Custom Command Package

In this demonstration, we will create a custom command add-on, the function of which is to display the input and selected content from the instruction block in the App log.




**Tip:** Note:

In the following content, "JAKA_Command" refers to the package name of the AddOn custom command we are about to create.

### 5.2.1 Step One: Create an AddOn Folder

First, we need to create a basic AddOn folder, which should contain a `.ini` format configuration file and a json file for data storage.

You can create a new folder, or you can modify the AddOn template under the "Template" directory in the AddOn development package repository.

**Tip:** Directory Structure

```
|——JAKA_Command
   |——AddOn.json
   |——JAKA_Command_config.ini
```

- **AddOn.json** The json file can be empty. Data will be generated and written in the subsequent steps.
- **JAKA_Command_config.ini**

The content of the *configuration file* is as follows: `ini [AddOnInfo] convention = 3.0 name = JAKA_Command description = "Output information in the log" version = 1.0 type = 1 portal = 10006 url = http://localhost/myAddOnUi languagetype = node-red service = AddOn.json serviceenabled = 1`

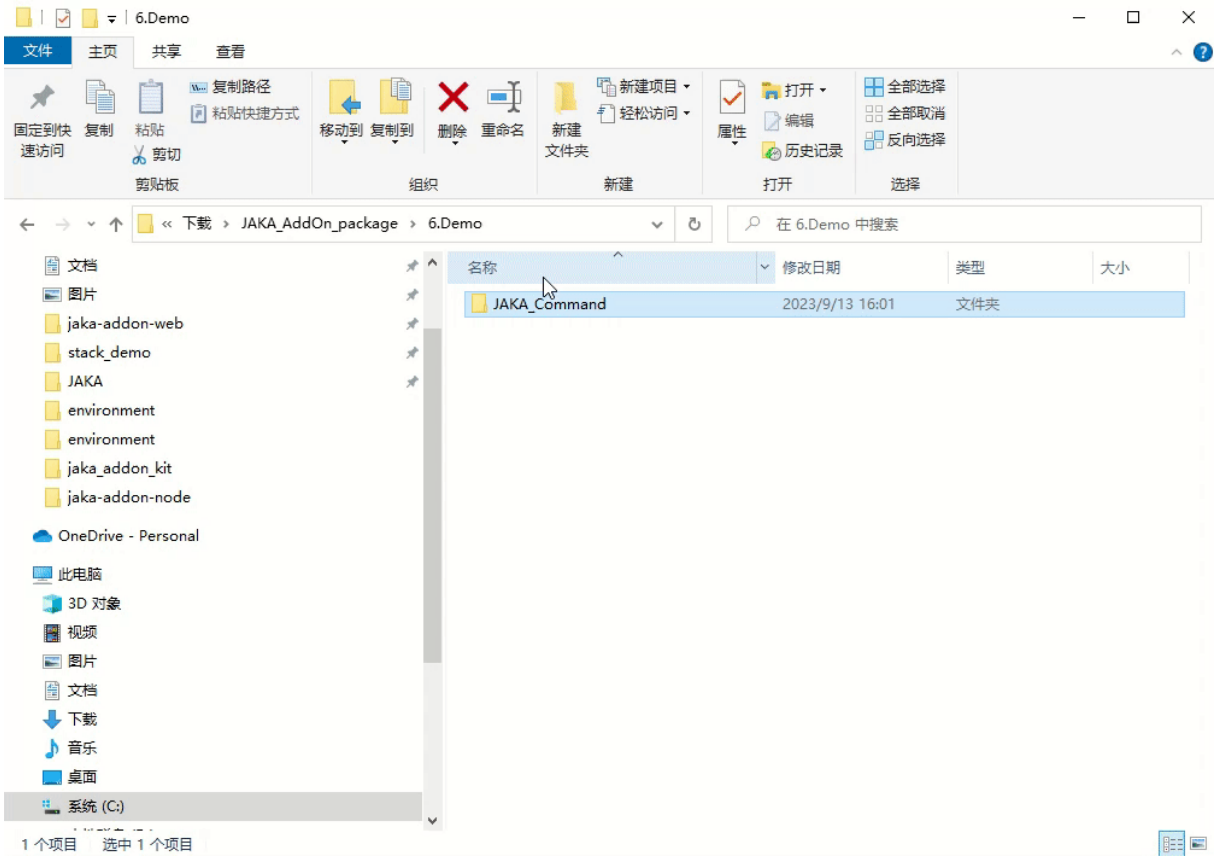
**Tip:** Tip:

Refer to *AddOnConfigurationFileSpecification [Click]* for details.

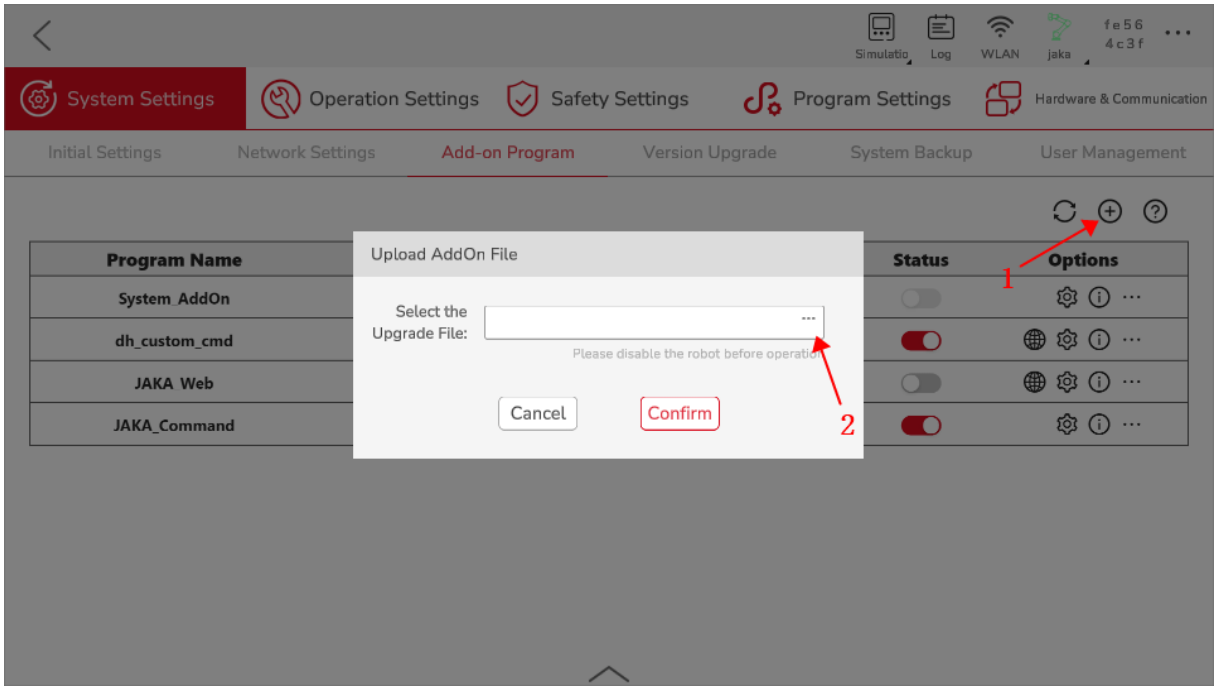
### 5.2.2 Step Two: Package & Upload

Next, we will package the folder created in Step One and upload it to the controller. The required format for packaging is `.tar.gz`.

In Windows, you can use tools like 7z for secondary compression.




In the App, use the attached management page to upload the add-on.



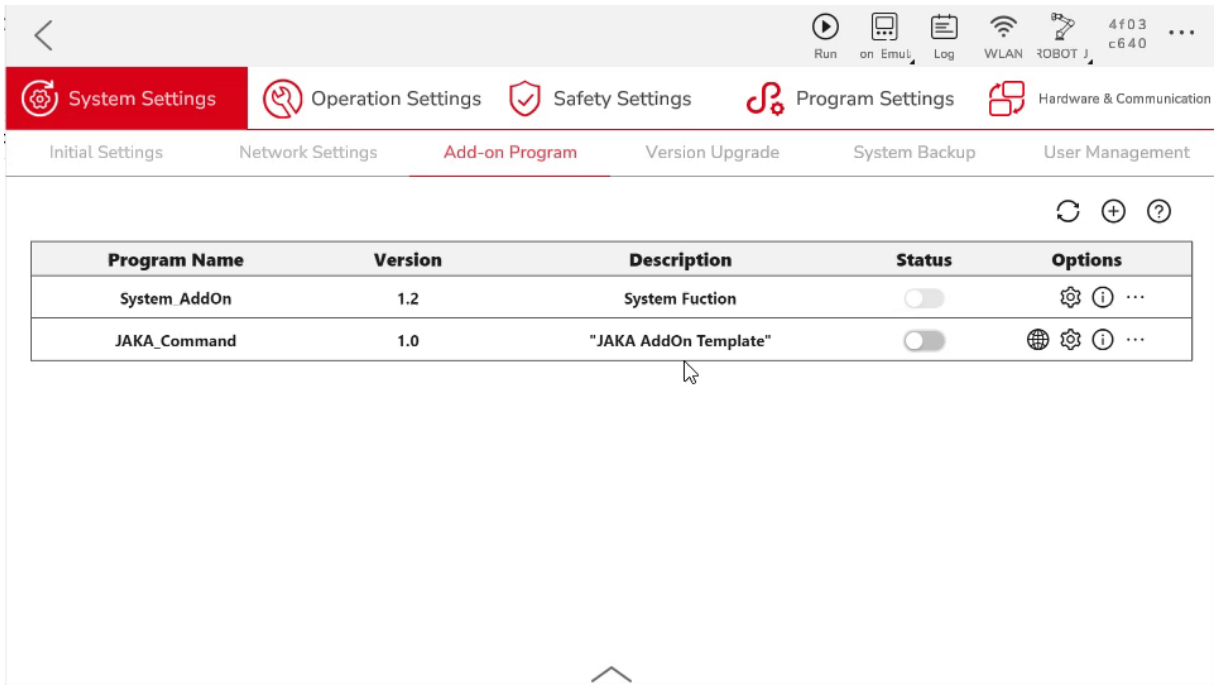

**Tip:** Summary

Up to now, we have fully prepared for the development of AddOn package. An AddOn package of standard format has been made and send to the controller.

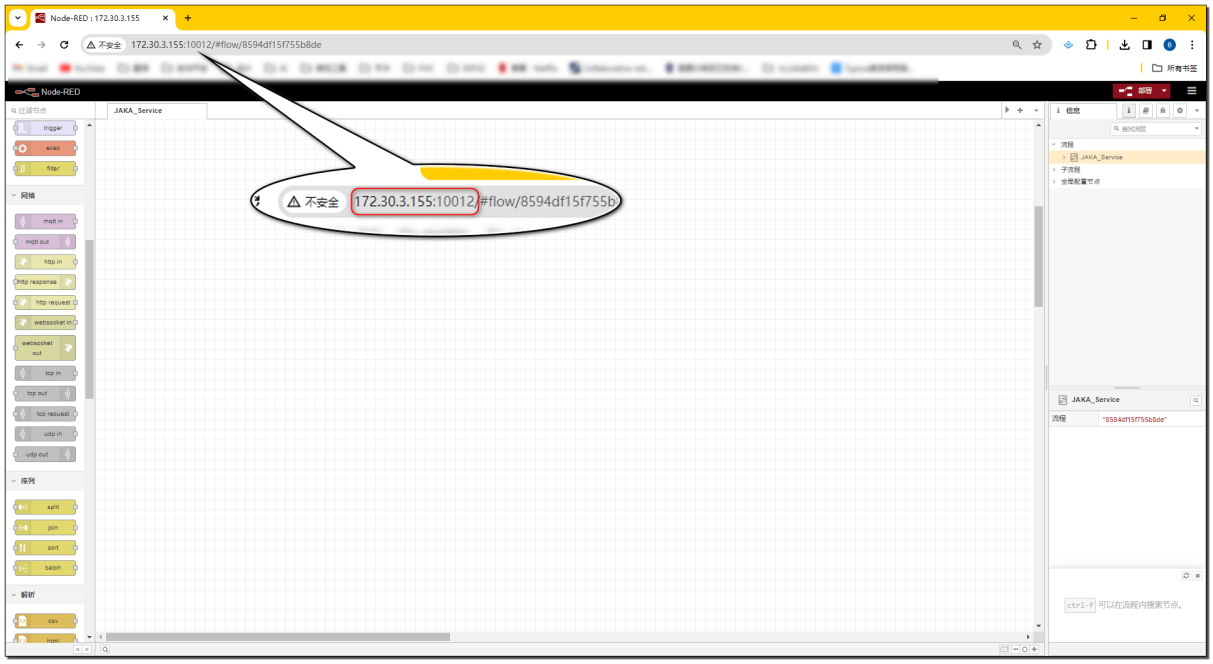
Now, let’s develop the specific function of AddOn package.

### 5.2.3 Step Three: Develop Custom Commands

1. Run the add-on and check the port number it is running on.



2. Open a browser, enter the robot's IP address and port number in the address bar, `IP:Portal`, to access the developer interface.




**Tip:** Summary

When the AddOn is running, the controller would start a node-red that is based on the `AddOn.json` file in AddOn package as an example. We can have access to this node-red through the robot's IP address and the current port number of the AddOn.

**Warning:** Please notice:

AddOn has more than one running port, which would be distributed by the system based on the real time situations. Therefore, the user should check the current port number of the AddOn before using it.

If the user needs to use this port number when programming in node-red, it is suggested to check it the AddOn's configuration file `xxx_config.ini`.

3. In the left toolbar, find the `Custom Commands` node, drag it to the middle, and click "Deploy".

Double-click the `Custom Commands` node to open the command block generation tool page.

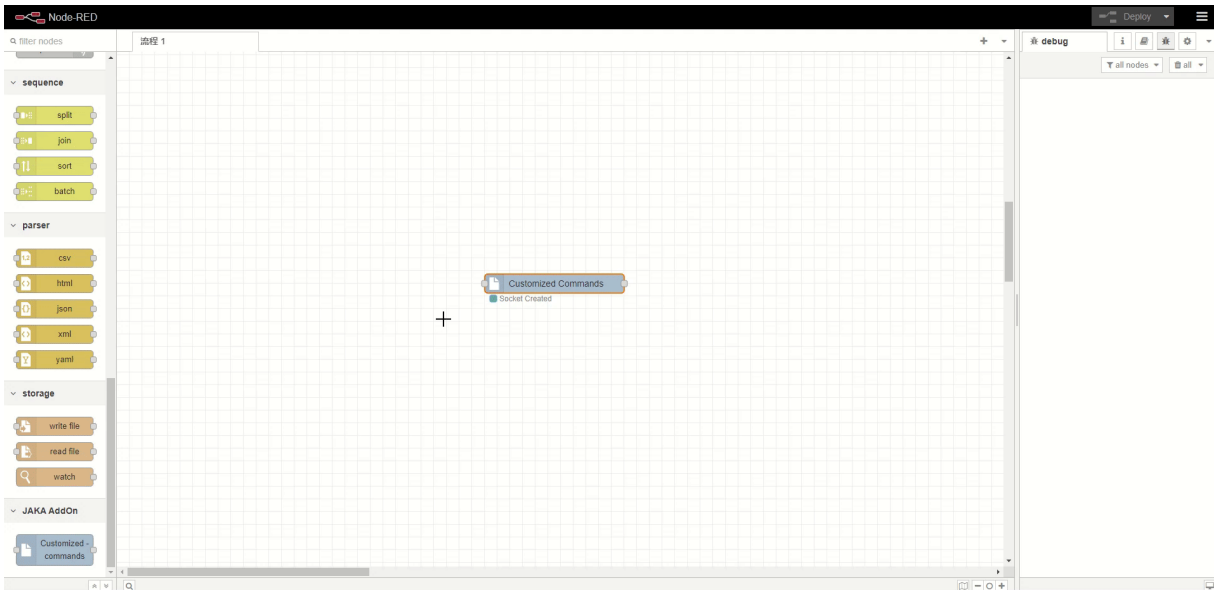
Video: open_coustomer_node.mp4

**Tip:**

The user guide of `Custom Command` node can be viewed in the HELP interface in the upper right corner of the webpage.

1. In the *Basic Configuration* page, define the style and attributes of the command block. In this case, use the attributes provided by the example page without making any changes. Change the instruction block name to *hello_jaka* and set the color to green. In the *Script Generation* page, define the behavior

of the command block. In this case, use the script provided by the example page without making any changes.



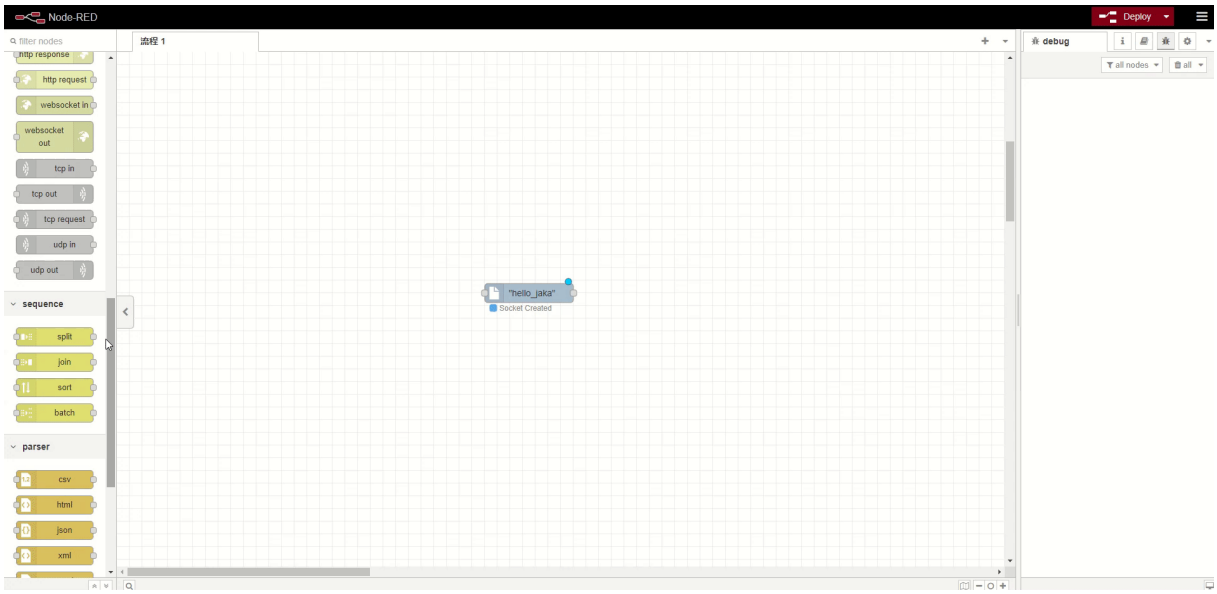

**Tip:**

The script language used here is JAKA Script. For detailed usage, please refer to JAKAScriptManual.

**Tip:** Summary

Up to now, we have a clear definition of custom commands, including its format, property, script, etc. The custom command we made can already be found in the App, but unable to be used. In the following steps, we need to help the App have access to the custom command's script by completing the development in node-red.

5. Use the `http in` and `http response` nodes to complete the flow.



**Warning:** Please Note:

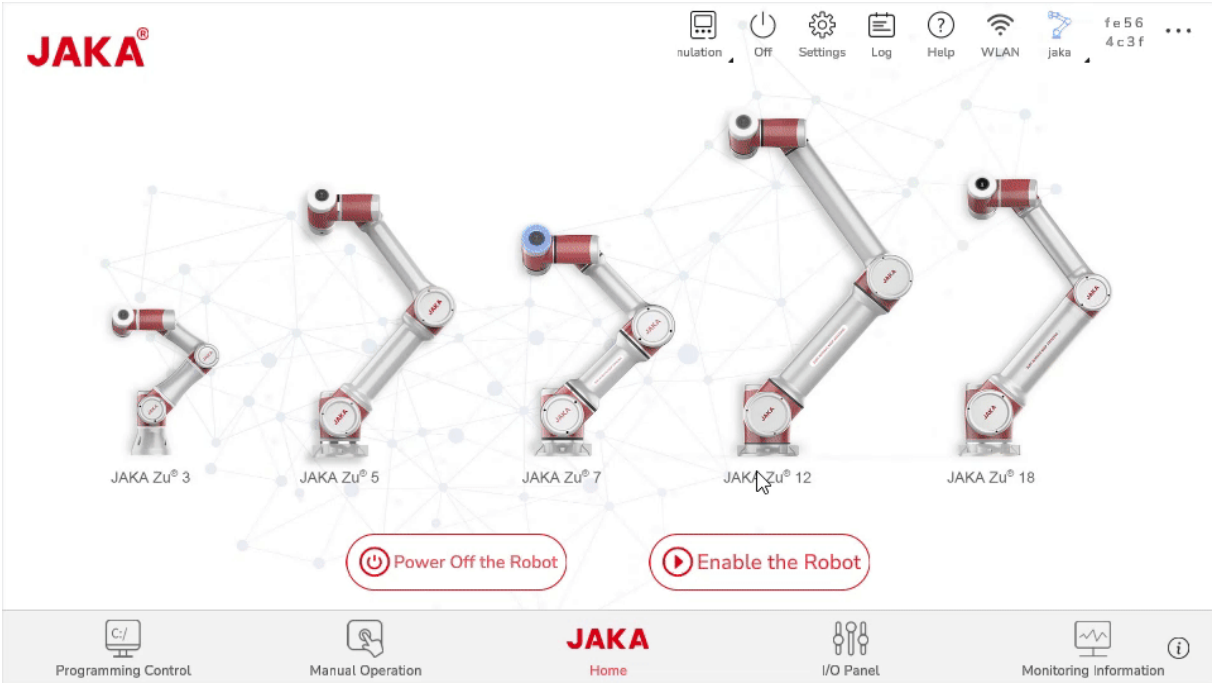
The URL of the `http in` node must be consistent with the name of the custom command block.

**Tip:** Tip

It is fair to regard the custom command as a kind of custom service, which

### 5.2.4 Step Four: Run the Developed Add-on

After deploying the flow in node-red, click the refresh button on the attached programs page. You will find the newly created command block in the extended commands of the programming page in the App.




**Tip:**

Congratulations! You have completed a whole procedure of custom command development. Next, let's find out how to debug and handle some FAQ.

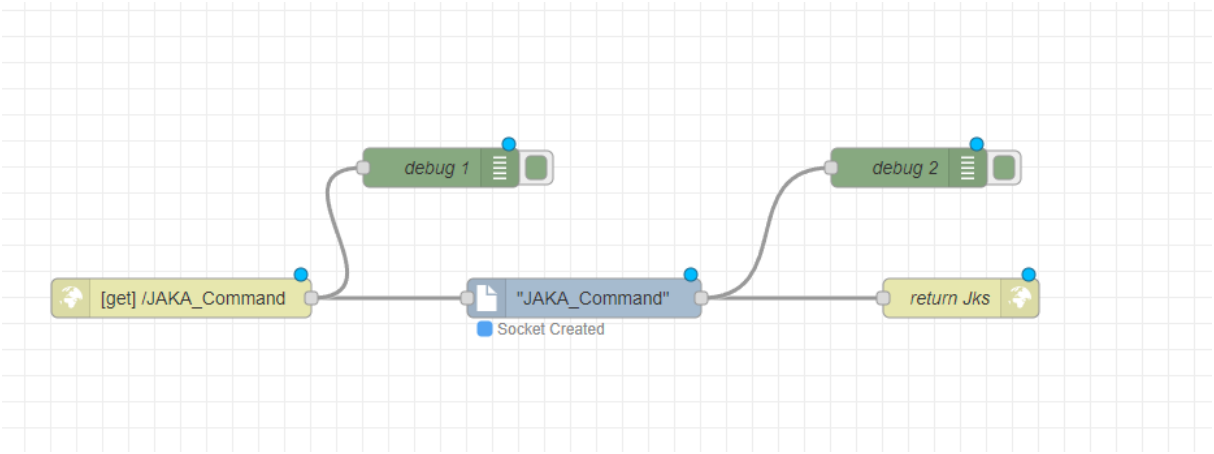
## 5.3 Debugging

**How to Check if the Communication Parameters are Correct?**

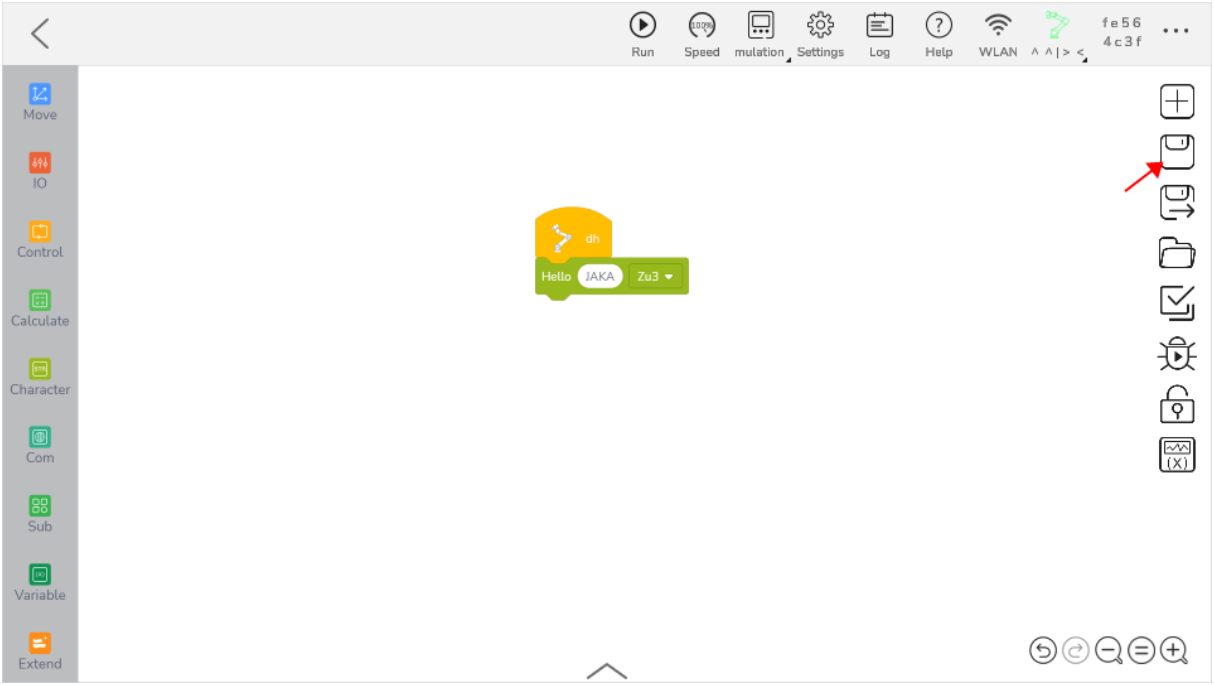
**Tip:**

It is a common debugging method to use debug node in node-red.

Use the debug node in node-red for debugging. It can debug the parameters sent by the instruction block when saving and the returned JKS to see if they are correct.



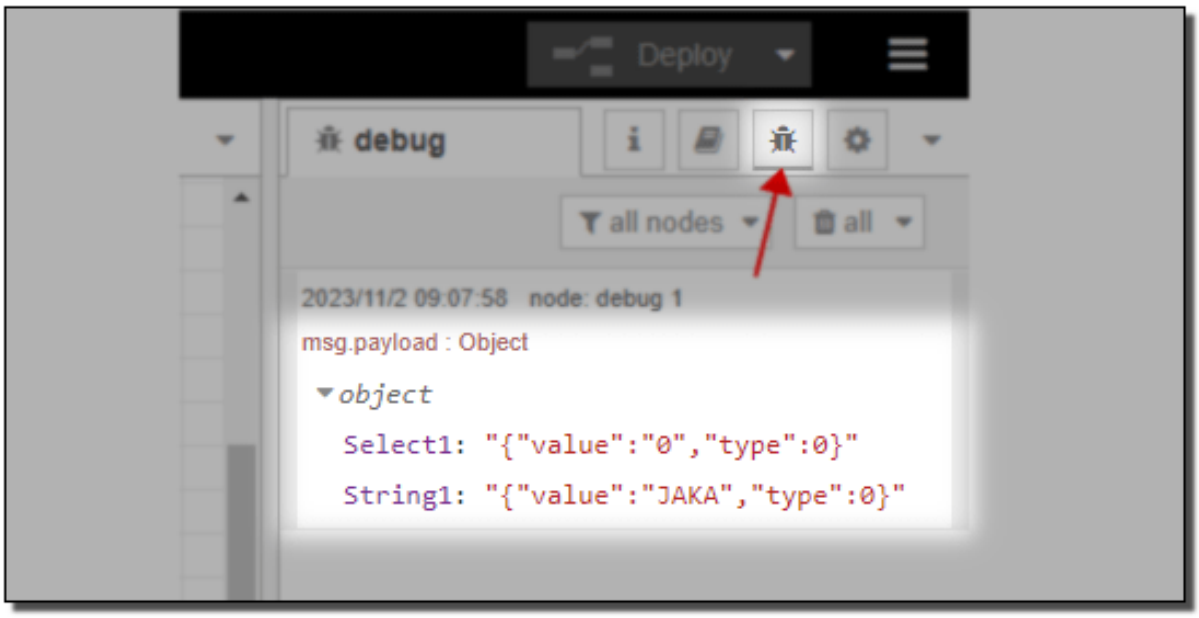

When saving or running the program, the App initiates a GET request.




At this point, debug1 can capture the parameters transmitted in the request. The keys of this object are the names of the attributes on the instruction block, and the values are the specific content of the attributes, including the attribute value and type.

Using the content below as an example, “Select1” is the name of the dropdown box attribute defined, “value” is the attribute value, and 0 indicates that the first option in the dropdown box is selected at this time.

“type” is the type of parameter entered in the instruction block, which is only meaningful for input box type attributes. 0 indicates that the value entered in the input box is manually entered, and 1 indicates that the input box contains content of variable type (data types of instruction blocks such as system variables, calculation results, etc.).




The information printed in debug2 includes the object of the JKS script generated this time.

Among them, `error_code` is the error code for generating the JKS. 0 means normal generation, and -12 means internal error. When this error code is not equal to 0, the App will pop up an error message with the corresponding error code and error information. Here, we only need to pay attention to the "jks" attribute, which is the actual generated JKS script.




**Simulate Script Generation**

- On the script generation page, after entering parameters in the left instruction block, click "Generate JKS Script" to simulate the generation of a script. View the simulated generated code in the code

window below.

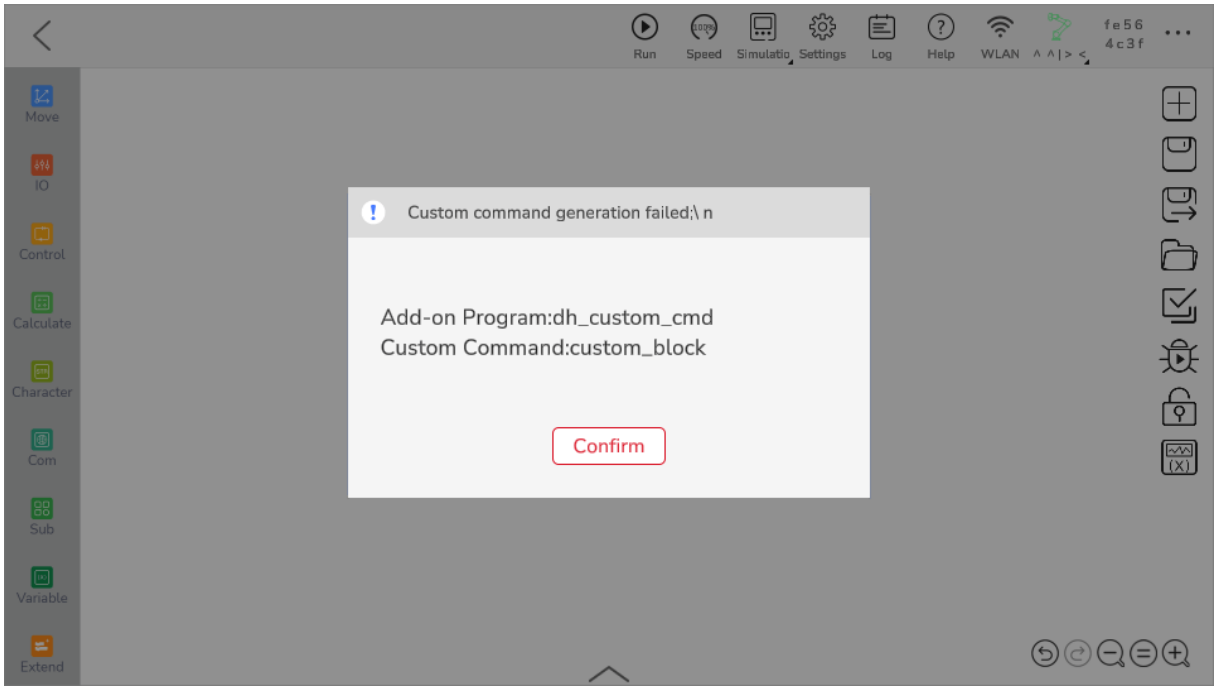
**Tip:** Please Note:

Clicking the "Generate JKS Script" button only displays the simulated generated script in the code window below, and has no other practical effect.

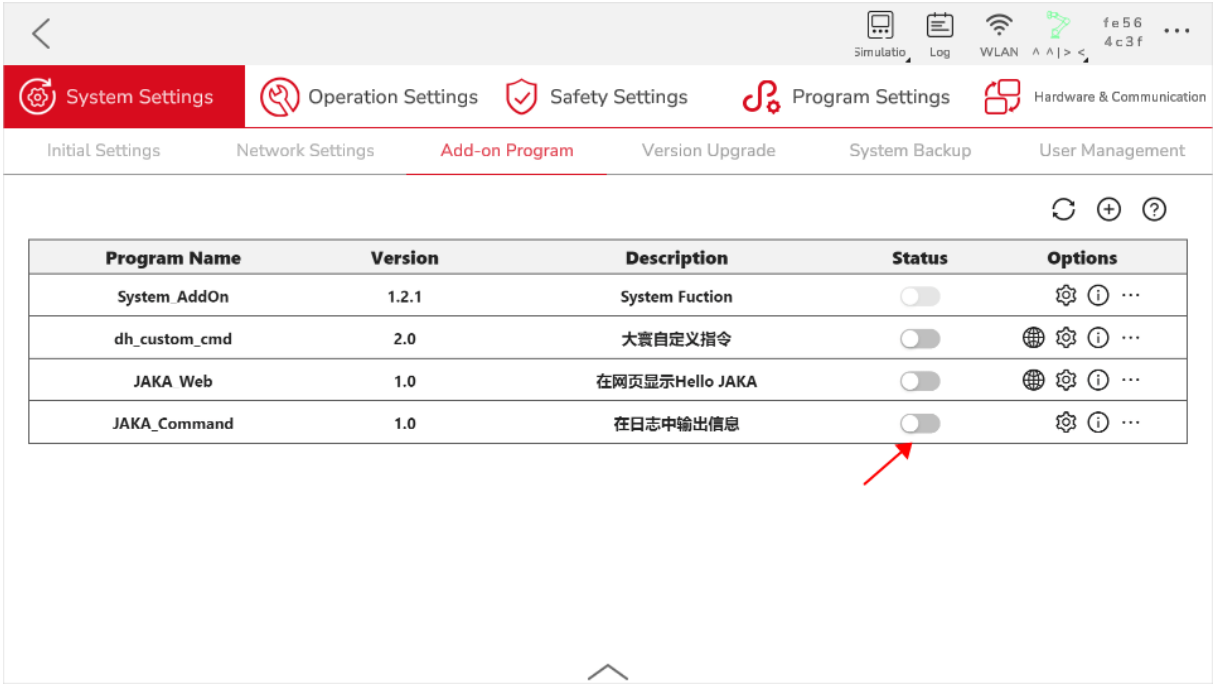
## 5.4 FAQ

During the development process, you may encounter the following issues. Please refer to the following text for specific situations and corresponding debugging methods.

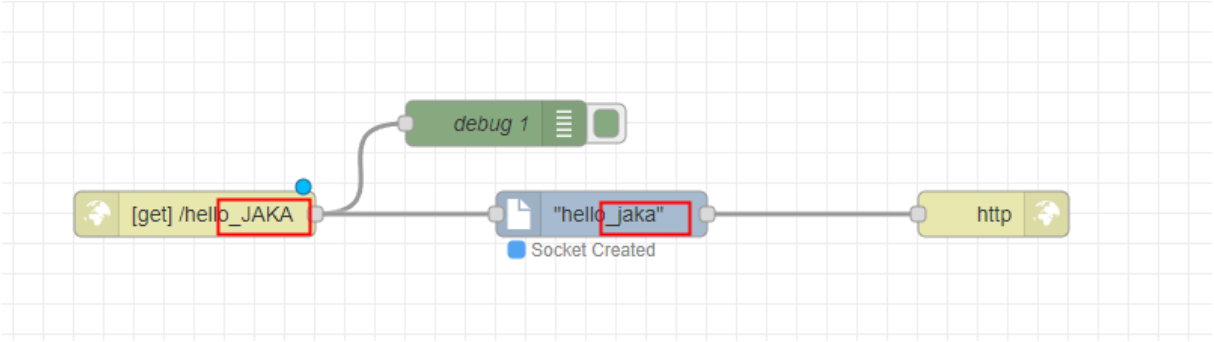
```
**Custom Command Generation Failed**
When using a custom command block in the App programming page, clicking save may␣
↪result in an error message "Custom command generation failed". This may be caused␣
↪by the following reasons.
```



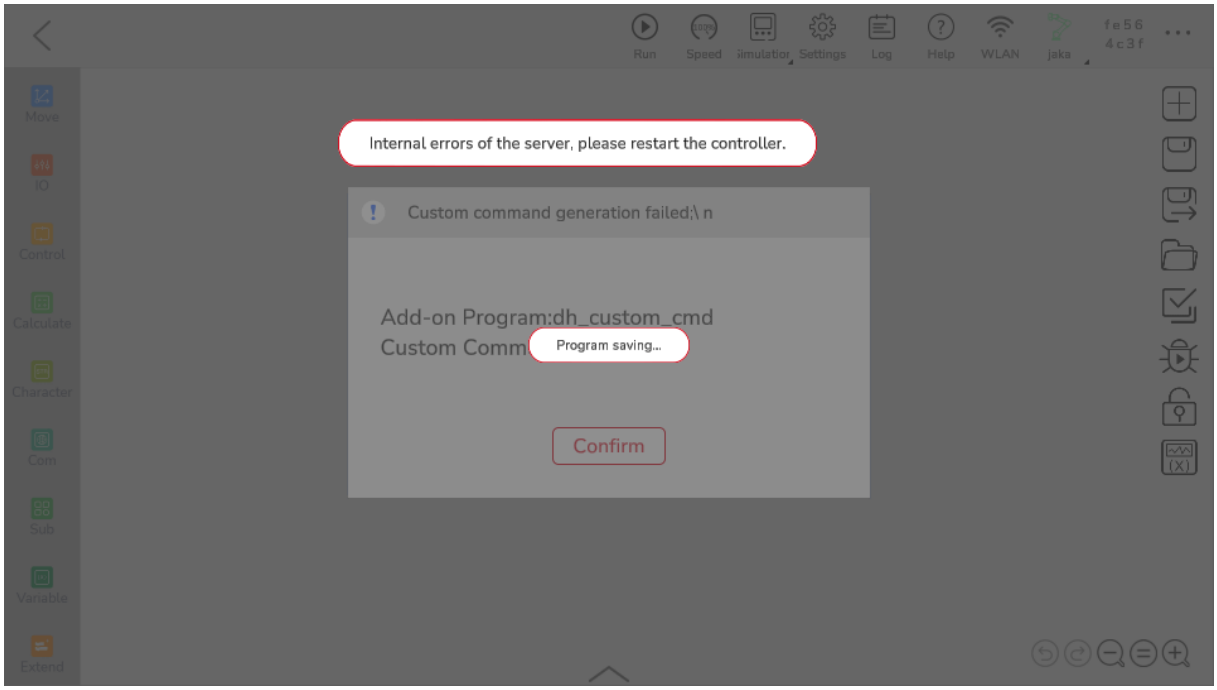

1. Add-on is not running Solution: Find the add-on in the attached programs page and click the run button to start the add-on.




2. The URL in the `http in` node in node-red does not match the name of the instruction block Solution: Ensure that the URL matches the name of the instruction block.




```
**Internal Server Error**
When encountering an internal server error, it is generally due to a timeout in␣
↪receiving the HTTP request in the App.
```

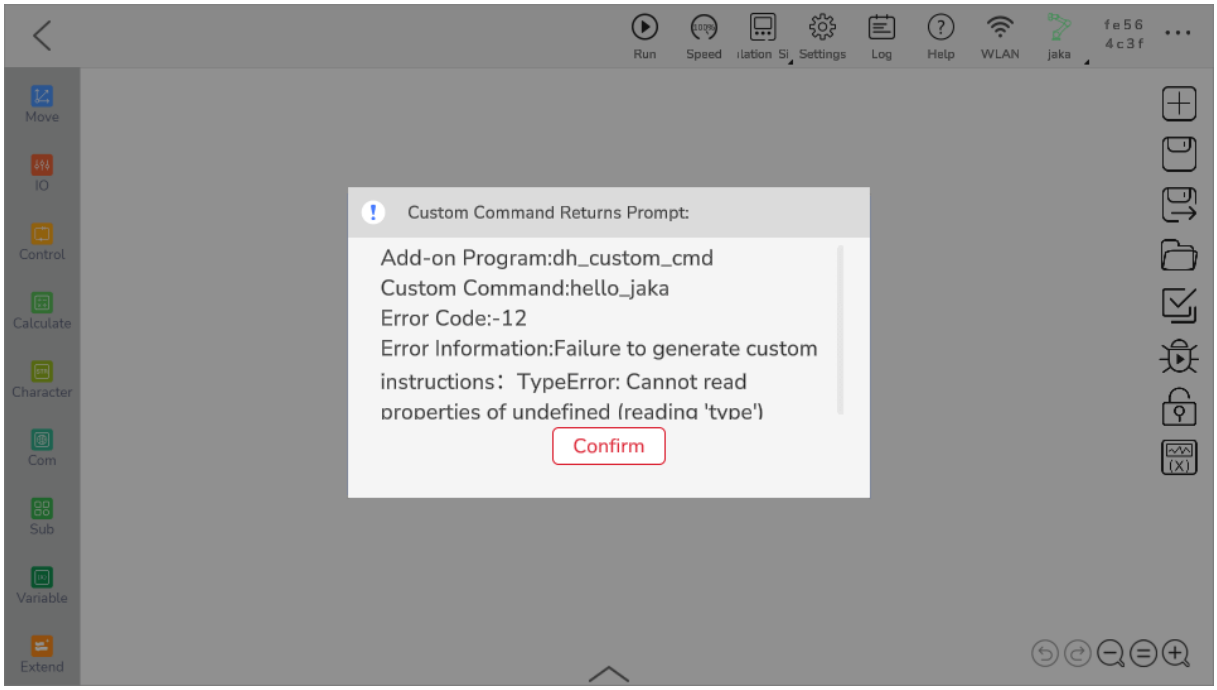



Solution: Check if the `http in` node in node-red is correctly connected to the `http response` node. After connecting, restart the add-on on the App.

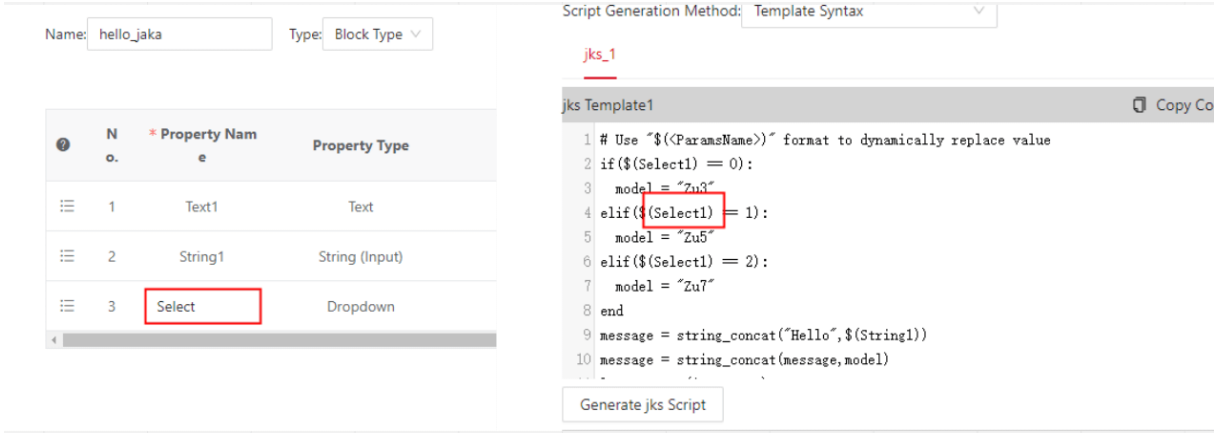
**Warning:**

- Currently, even after reconnecting, you still need to restart the AddOn service to save the program. This is caused by some unknown bugs internally and will be fixed in future versions.
- The description in the error message that instructs to restart the controller is incorrect. Restarting the add-on is sufficient.

**Attribute Matching Failed**




Solution: Check if the attribute names used in the custom command in node-red match those used in the script definition.



# 6 Custom Service Development

**This page aims to guide developers on how to develop custom service add-ons.**

**Tip:** Please note:

This chapter is an introductory tutorial with the main purpose of providing developers with a basic understanding of custom services and its fundamental development process.

If you want to learn how to develop AddOn packages with specific application features tailored to specific process scenarios, please refer to the subsequent complete case study chapters.

## 6.1 About Custom Services

### 6.1.1 What is a Custom Service

A custom service is a service that performs tasks in the background, and users do not directly perceive its existence.

### 6.1.2 Purpose of Custom Services

Custom services are responsible for handling the backend data processing and business logic that is unrelated to user interaction. When combined with Http nodes, custom services can interact with the frontend interface, providing services such as receiving, processing, storing, calculating, and analyzing data for custom commands, custom webpages, or other clients.

## 6.2 Create AddOn Custom Service Package

In this demonstration, we will create custom service AddOn package. The functionality of this AddOn is to provide a service to query the robot’ s power status.

**Tip:** Note:

The JAKA_Service mentioned below is the package name of the AddOn custom command package we are about to create.

### 6.2.1 Step One: Create AddOn Folder

First, we need to create a basic AddOn folder, which should contain a configuration file in `.ini` format and a Json file to store data.

You can create a new one or modify the AddOn template in the AddOn development kit repository under the “Template” directory.

**Tip:** Directory Structure

```
|——JAKA_Service
   |——AddOn.json
   |——JAKA_Service_config.ini
```

- **AddOn.json** The Json file can be empty. Data will be generated and written in later steps.
- **JAKA_Service_config.ini** The configuration file has the following content:

```
[AddOnInfo]
convention = 3.0
name = JAKA_Service
description = "JAKA AddOn Service Demo"
version = 1.0
type = 1
portal = 10006
url = http://localhost/myAddOnUi
languagetype = node-red
service = AddOn.json
serviceenabled = 1
```

**Tip:**

For detailed explanations of configuration options, refer to: *Configuration File Explanation [Click to Jump]*

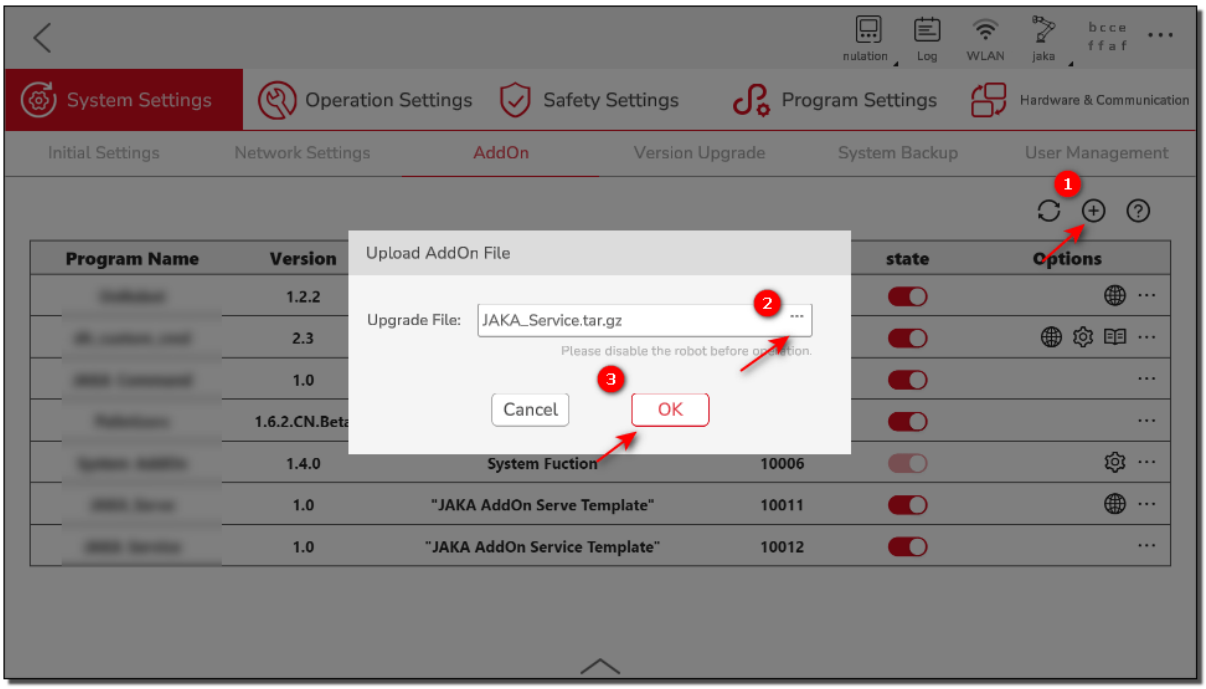
### 6.2.2 Step Two: Package & Upload

Next, we will package the folder created in step one and upload it to the controller for command block development. The packaging format should be `.tar.gz`.

In Windows, you can use tools like 7z for further compression.




Attach the AddOn in the App Management page to upload the AddOn.

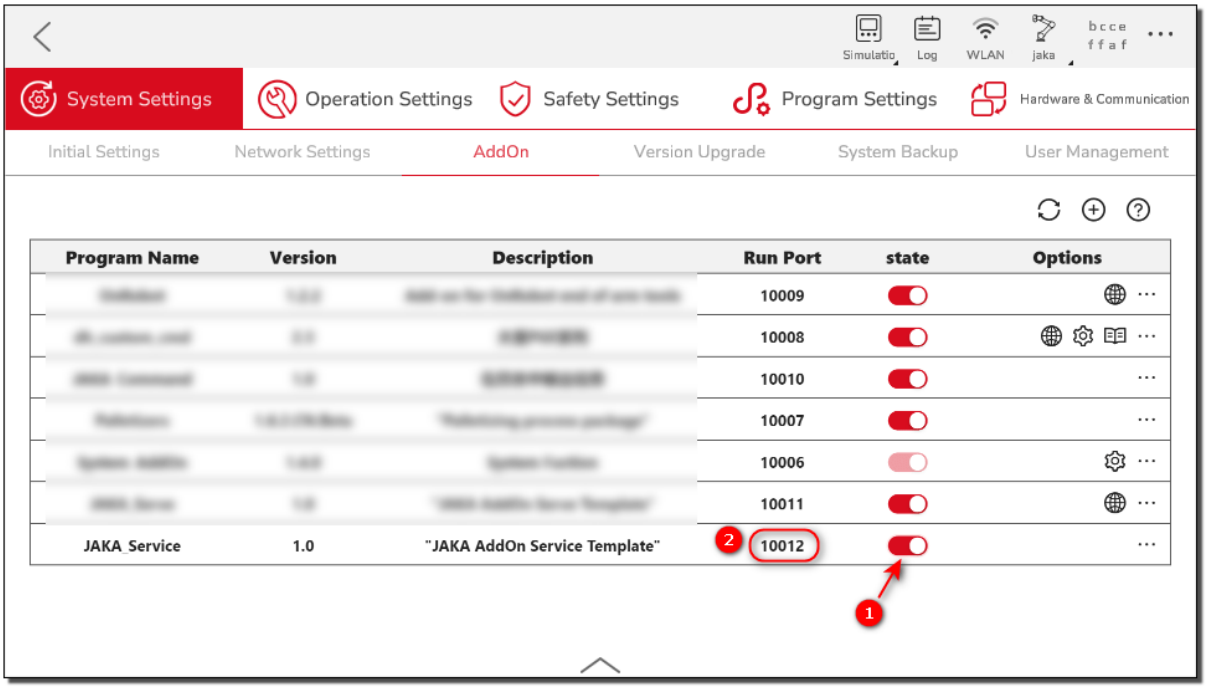



**Tip:** Summary

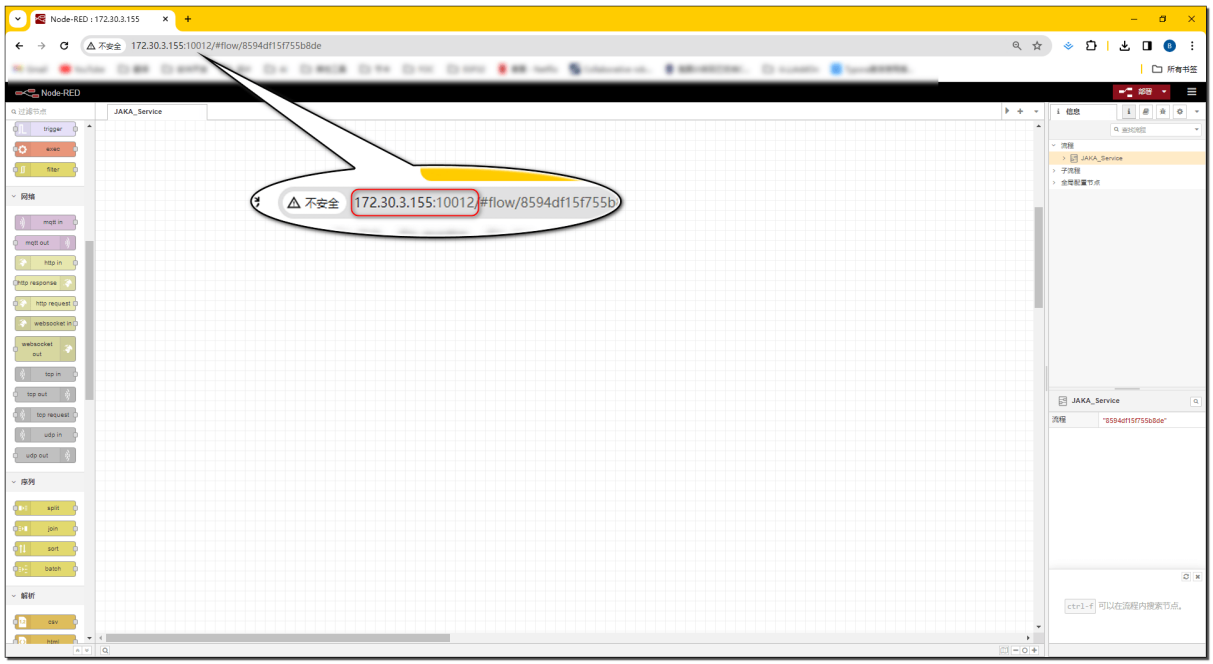
Up to now, we have completed the initialization of the AddOn package, packaged it in the standard AddOn format, and uploaded it to the controller. Next, we can proceed to develop the specific functionality of the AddOn.

### 6.2.3 Step Three: Develop Custom Service

1. Run the AddOn and check the port number on which the AddOn is running.




2. Open a browser and enter the robot's IP + port number (`IP:Portal`) to access the developer interface.




**Tip:** Summary

After the AddOn runs, the controller starts a node-red instance based on the `AddOn.json` file in the AddOn package. We can access this node-red instance by entering the robot's IP and the current running port

number of the AddOn.

**Warning:** Please Note:

The running port of the AddOn is not fixed and can change dynamically based on the system's allocation. Therefore, every time you need to use the AddOn port number, you should check the current port number. If you need to use this port number in your code, it is suggested to check check it from the AddOn's configuration file, `xxx_config.ini`.

3. In the left node bar under the JAKA AddOn directory, find the `Get Robot Information` node, double-click to open the edit page, set the name to "Power Status" and select "Robot Status" -> "Power Status" sequentially.

Video: 配置获取信息节点.mp4

**Tip:**

The usage instructions for the `Get Robot Information` node can be found in the HELP in the upper right corner of the webpage.

1. Add the `http in` node to listen for GET requests.

Video: 添加 Http 监听.mp4

**Tip:** Summary

After connecting with the `http in` node and defining the URL, we can make requests to our service through `Robot_IP:AddOn_Port/Your_URL`. However, we won't receive a response yet, so the next step is to create a response message object and return it to the request.

1. Generate a response message and use the `http response` node to respond to the request.

Video: 添加 Http 响应.mp4

**Tip:** Note

- Initially, we are unsure of the type and data structure of the query result. You can use the `inject` node to manually trigger a query and use the `debug` node to observe the return value.
- The response can only be a string or object type, so boolean values need to be placed inside an object before returning.

## 6.3 Testing

Here, we will use an API interface testing tool to test the service we just created.

Video: 测试.mp4

**Tip:** Congratulations

Now we can use our custom service to query the power status of the robot!

# 7 Custom Webpage Development

**This page aims to guide developers on how to develop custom webpage add-ons.**

**Tip:** Please note:

This chapter serves as an introductory tutorial, primarily to familiarize developers with the basic knowledge and development process of custom webpages.

For information on how to develop AddOn packages with specific function for specific scenarios, please refer to the subsequent comprehensive case chapters.

## 7.1 About Custom Webpages

### 7.1.1 What Are Custom Webpages

Custom webpages refer to pages that are not the existing pages within the JAKA App, developed by developers according to their own needs, and in compliance with web development standards.

### 7.1.2 Purpose of Custom Webpages

In working scenarios, it may be necessary to switch back and forth between different Apps, or between different pages within the JAKA App, or to stop working on a process to debug third-party devices. In such cases, it may have an impact on work pace and efficiency.

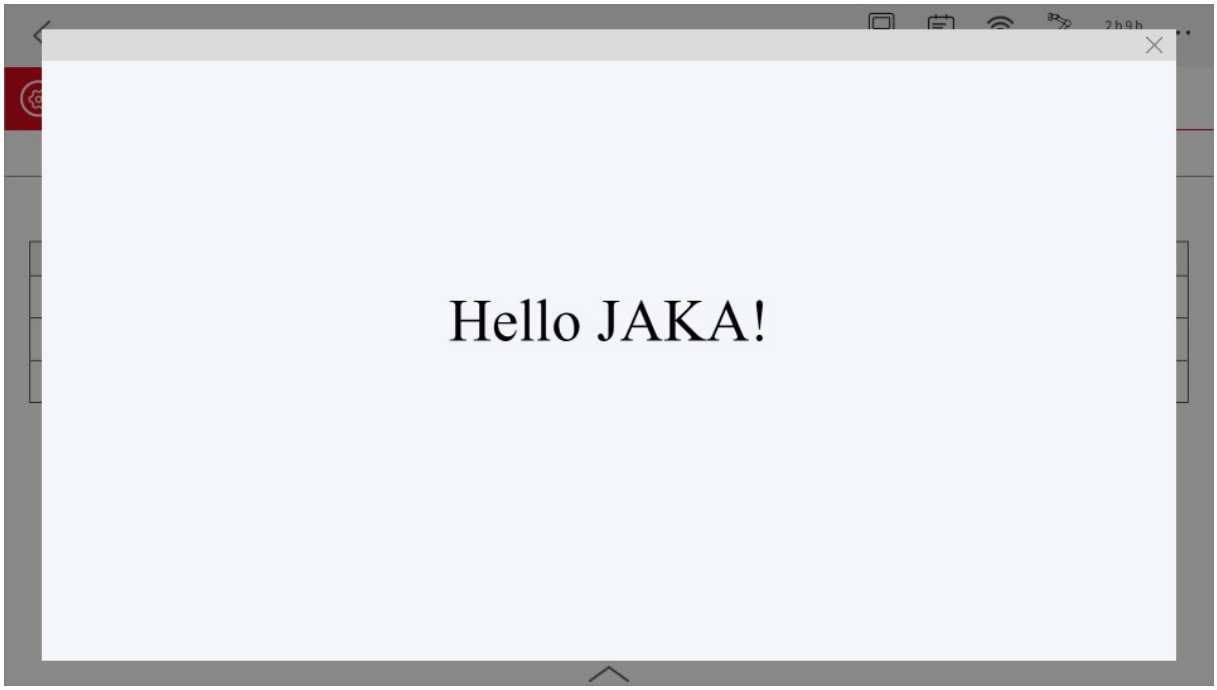
By using AddOn custom webpages, the "back and forth switching" issue can be solved by combining different widgets together to use.

## 7.2 Creating JAKA_Web

Currently, JAKA supports two ways to create a custom web.

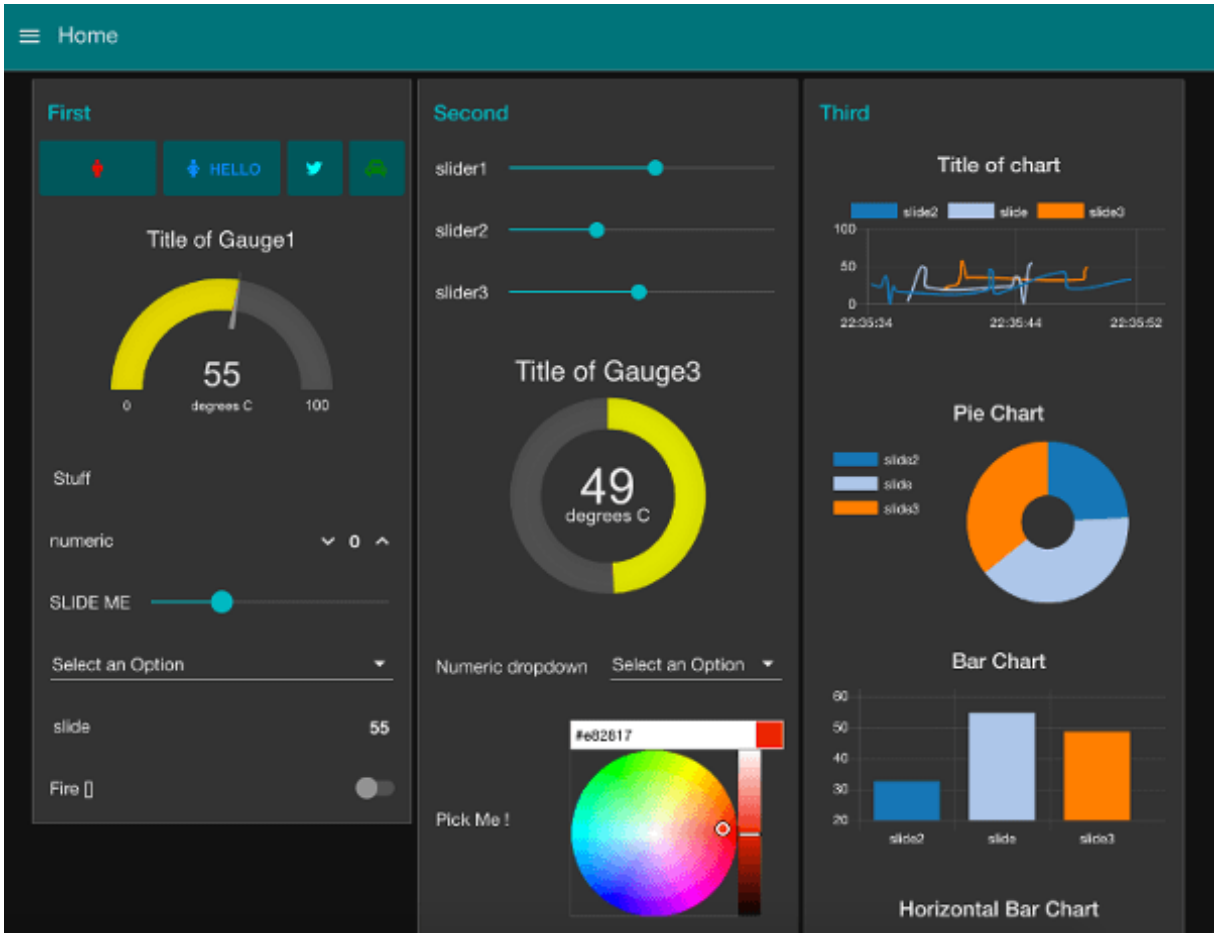
The first way is to use the dashboard (simple code UI template) in Node-RED. This way is easy and fast, as the templates and the widgets within are abundant. But webpage with complex logics and styles might be unable to create. The second approach involves developing and deploying in the form of front-end standard engineering, which allows for the creation of more complex pages.

In this demonstration, we will use both ways to create a JALA custom web AddOn. The function of this AddOn is to display "Hello JAKA!" on the page.




### 7.2.1 Method One: Use Dashboard Template

**About the ``dashboard`` template** This module in Node-RED provides a set of nodes for quickly creating real-time data and control panels. In subsequent tutorials, we won’ t delve too much into the basics of this module. For more details, you can refer to the official documentation node-red-dashboard, or search online for third-party tutorials to learn more.




**Step One: Create an Add-on Folder** First, we need to create a basic AddOn folder. This folder should contain a configuration file in .ini format and a frontend project file. We recommend using "client" as the folder name for the frontend project.

You can create a new folder, or modify the AddOn template under the "Template" directory in the AddOn development kit repository.

**Tip:** Directory Structure

```
|—JAKA_UI
   |—AddOn.json
   |—JAKA_Web_config.ini
```

- **JAKA_UI_config.ini**

  *Configuration File* 内容如下:

```
[AddOnInfo]
convention = 3.0
name = JAKA_UI
description = "Display "Hello JAKA!"
version = 1.0
```

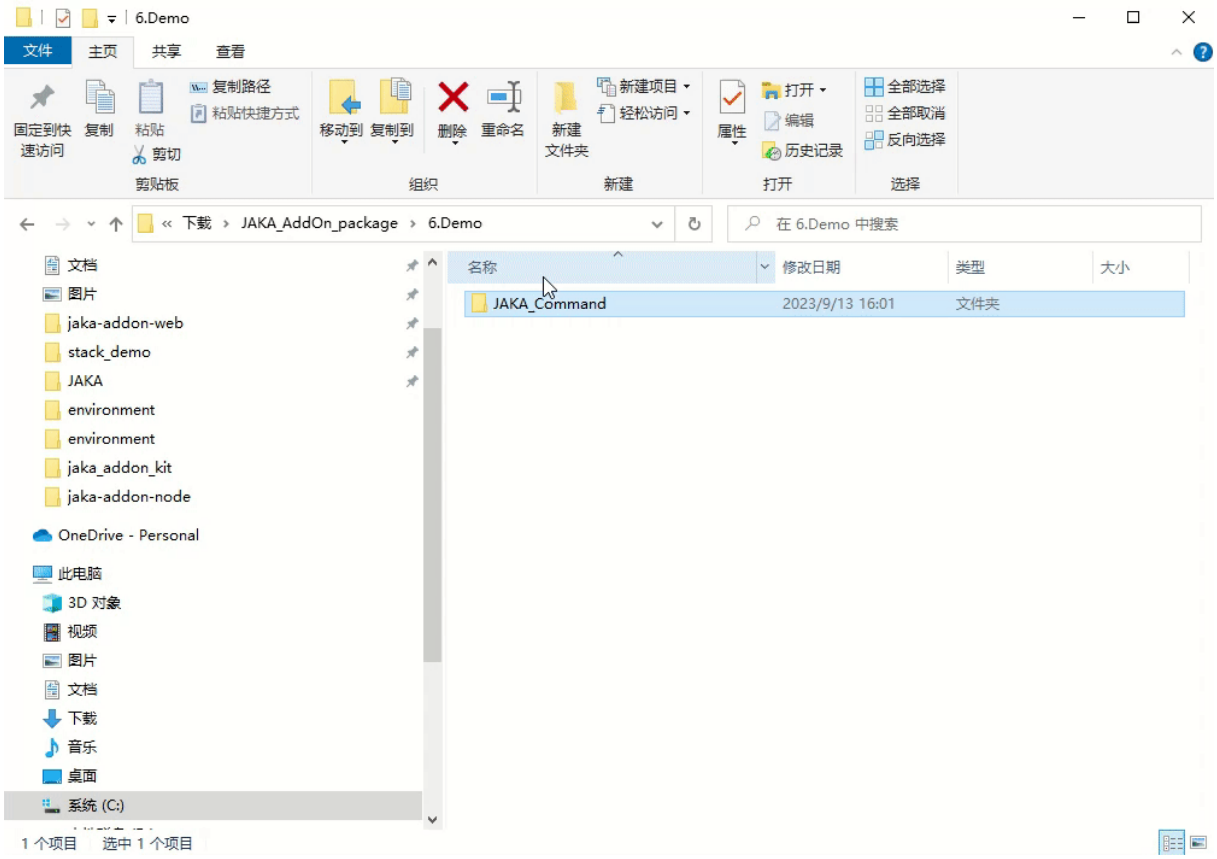
(continues on next page)

(continued from previous page)

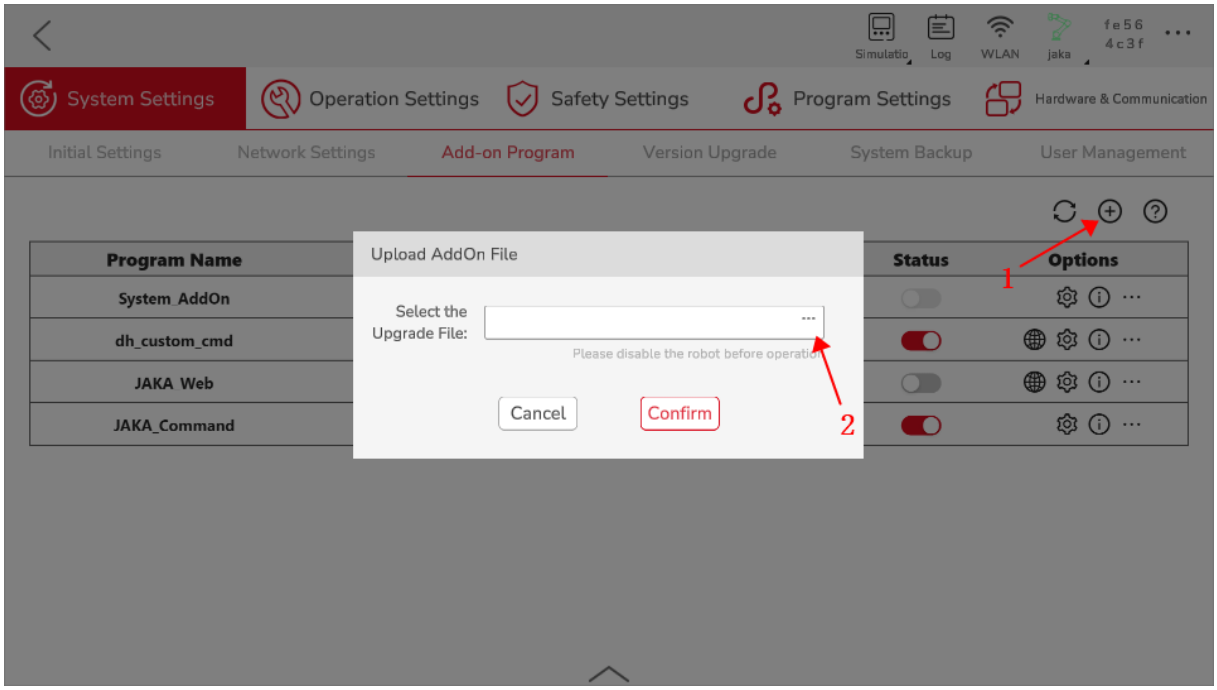
```
type = 3
portal = 10006
url = http://localhost/ui
languagetype = node-red
service = AddOn.json
serviceenabled = 0
```

**Step Two: Package & Upload** Next, we will package the folder created in step one and upload it to the controller for custom Web development. The required format for packaging is `.tar.gz`.

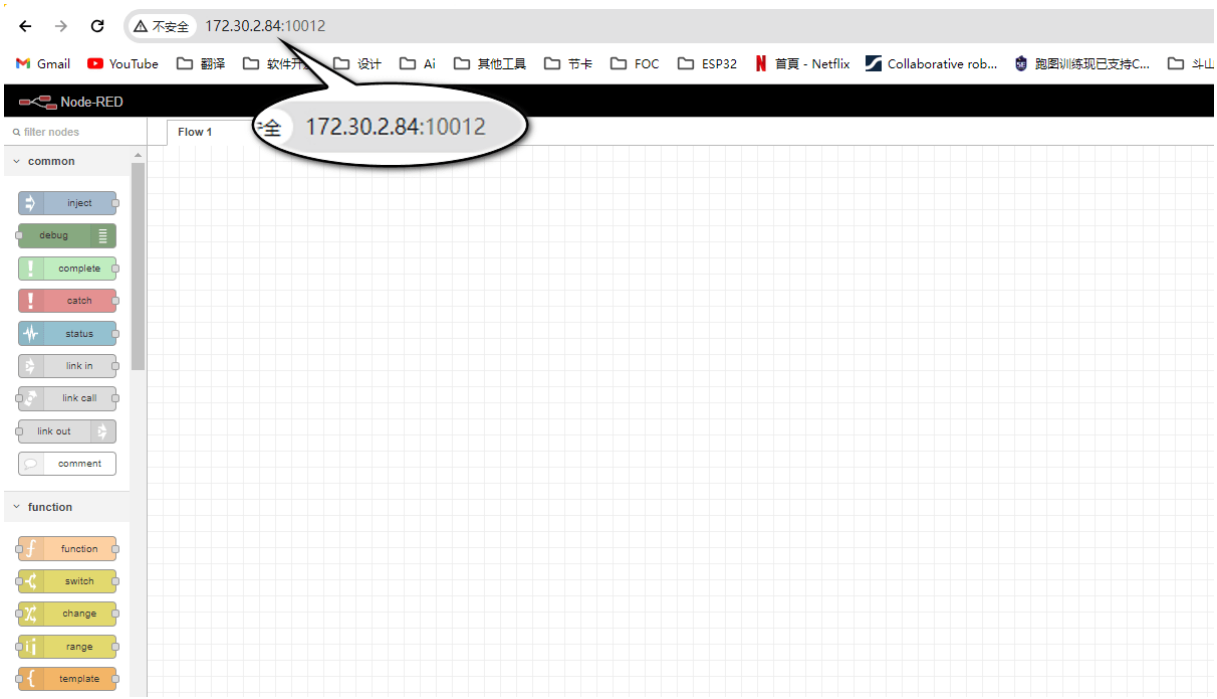
In Windows, you can use tools like 7z for further compression.




Attach the manager page in the App to upload the AddOn.



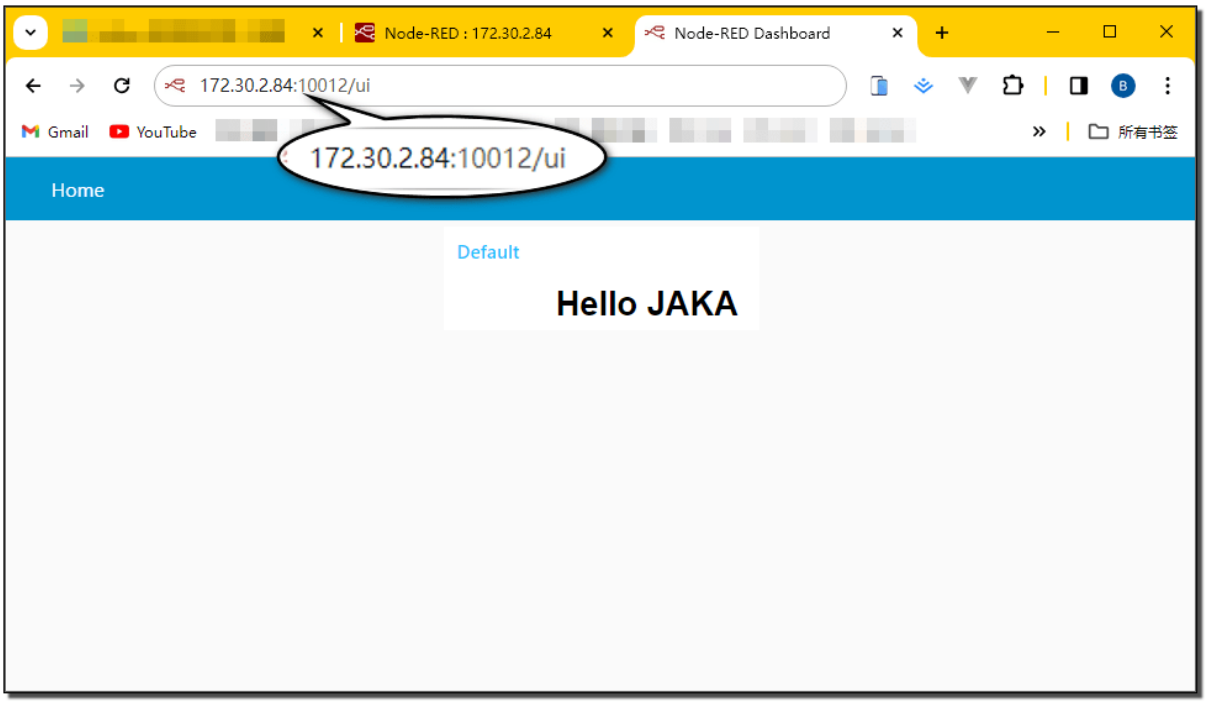

**Step Three: Use Dashboard Widget to Create UI Webpage** After uploading and running the AddOn, you can access the Node-RED development page of the AddOn by entering the IP address of the current machine where the robot is hosted, followed by ":" and the port number on which the AddOn is running, into the browser's address bar. This will allow you to access and interact with the AddOn through Node-RED.




After entering the development page, locate the "dashboard" directory on the left side. We will use a `text` node to display the text "Hello JAKA!" on the page.

Video: useTextNode.mp4

After completing the configuration following the above steps, click on “Deploy.” Once deployment is complete, you can access the UI page of the AddOn by entering the IP address of the current machine where the robot is hosted, followed by “:” and the port number on which the AddOn is running, and then adding “/ui” to the end of the address in the browser’s address bar. This will allow you to access the UI page of the AddOn.




**Tip:** Note

Currently, UI pages created using this method can only be accessed through a web browser and cannot be accessed through the “globe” button within the app.

In the following stages, we will improve the configuration file parsing method to enable access through the “globe” button within the App.

**Tip:** Summary

Here, we’ve learned how to enter the Node-RED components, locate the Dashboard component node, and use the `text` node to display text. We’ve also successfully accessed it through a web browser.

Following this, you can explore the advanced tutorial using *Usw JAKA Dashboard Template* to learn how to use and modify some of the preset Dashboard components we’ve provided.

### 7.2.2 Method Two: Use Front-end Standard Engineering

**Step One: Create an AddOn File** First, we need to create a basic AddOn folder. This folder should contain a configuration file in .ini format and a frontend project file. We recommend using "client" as the folder name for the frontend project.

You can create a new folder, or modify the AddOn template under the "Template" directory in the AddOn development kit repository.

**Tip:** Directory Structure

```
|—JAKA_Web
   |—client
   |   |—index.html
   |—JAKA_Web_config.ini
```

- **index.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Hello JAKA!</title>
</head>
<body style="background-color:#f5f6fa;">
    <p style="font-size: 10vh;text-align: center;margin-top: 20%;"> Hello JAKA!
↪</p>
</body>
</html>
```

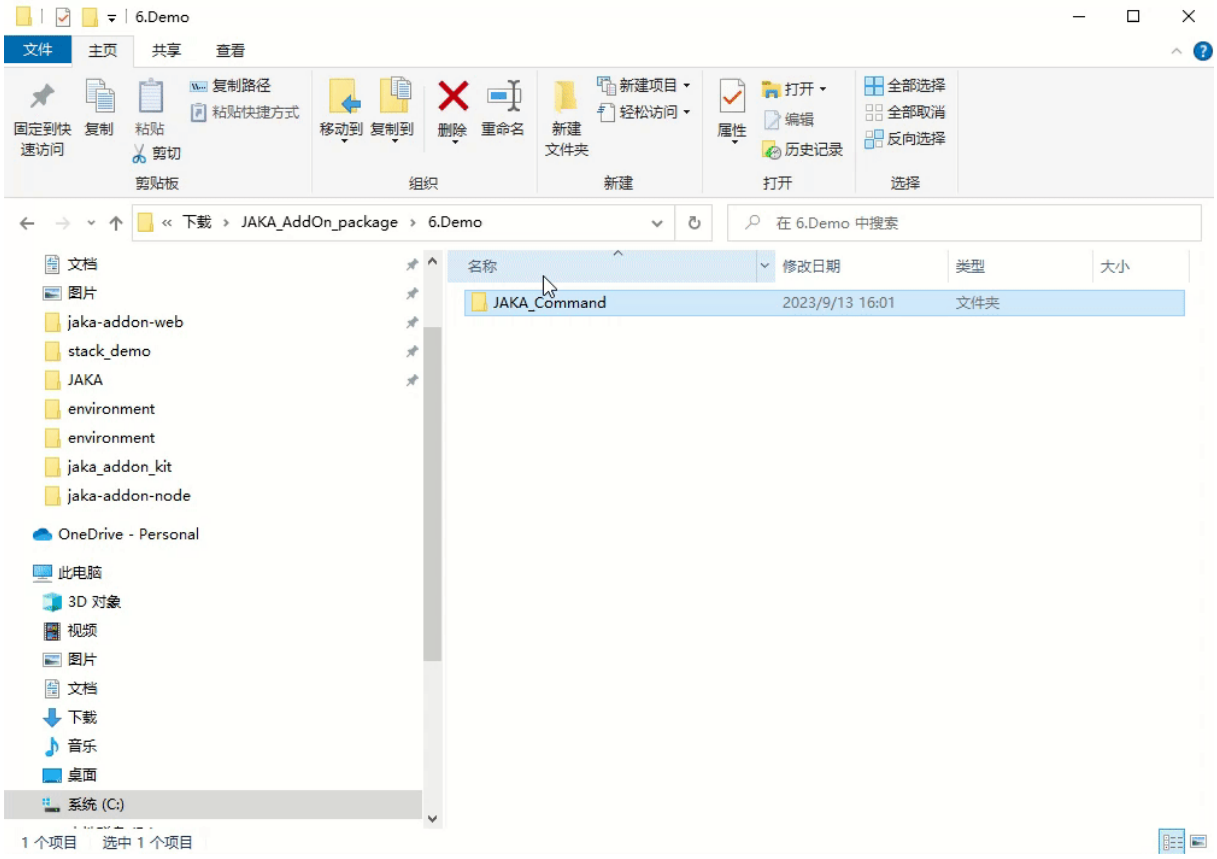
- **JAKA_Web_config.ini**

  *Configuration File* content is as follows:

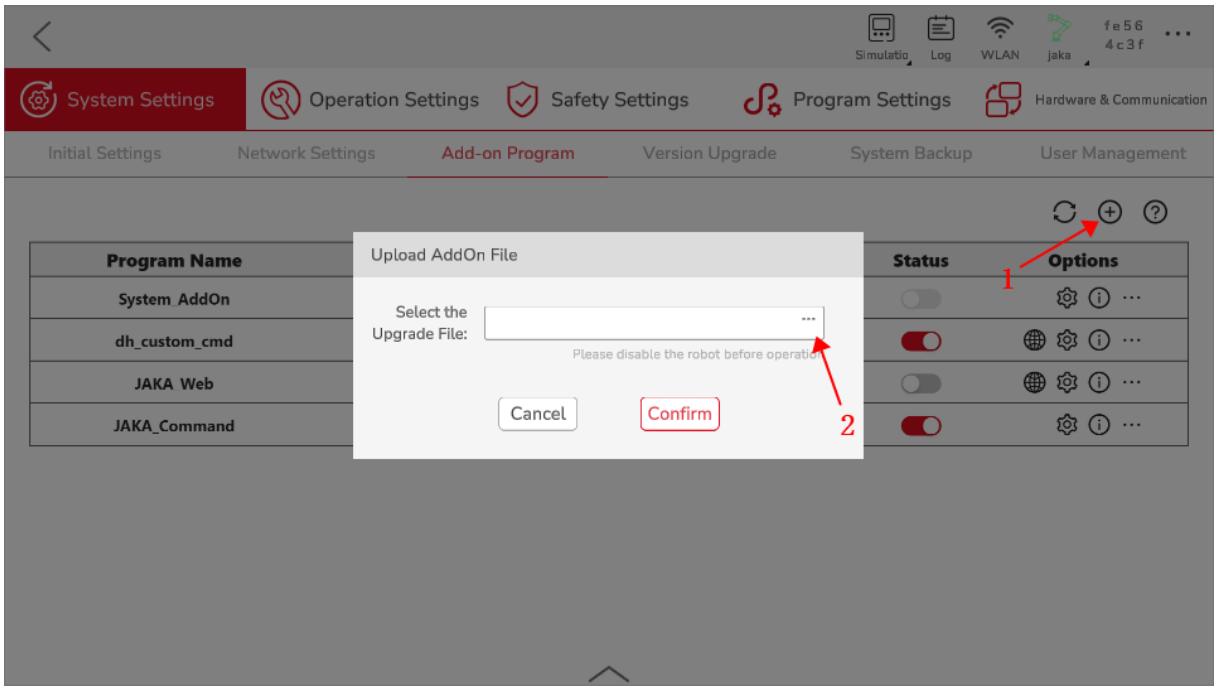
```
[AddOnInfo]
convention = 3.0
name = JAKA_Web
description = Display "Hello JAKA!" on the page.
version = 1.0
type = 3
portal = 10006
url = http://localhost/JAKA_Web/client/
languagetype = node-red
service = null
serviceenabled = 0
```

**Step Two: Package & Upload** Next, we will package the folder created in step one and upload it to the controller for custom Web development. The required format for packaging is `.tar.gz`.

In Windows, you can use tools like 7z for further compression.




Attach the manager page in the App to upload the AddOn.



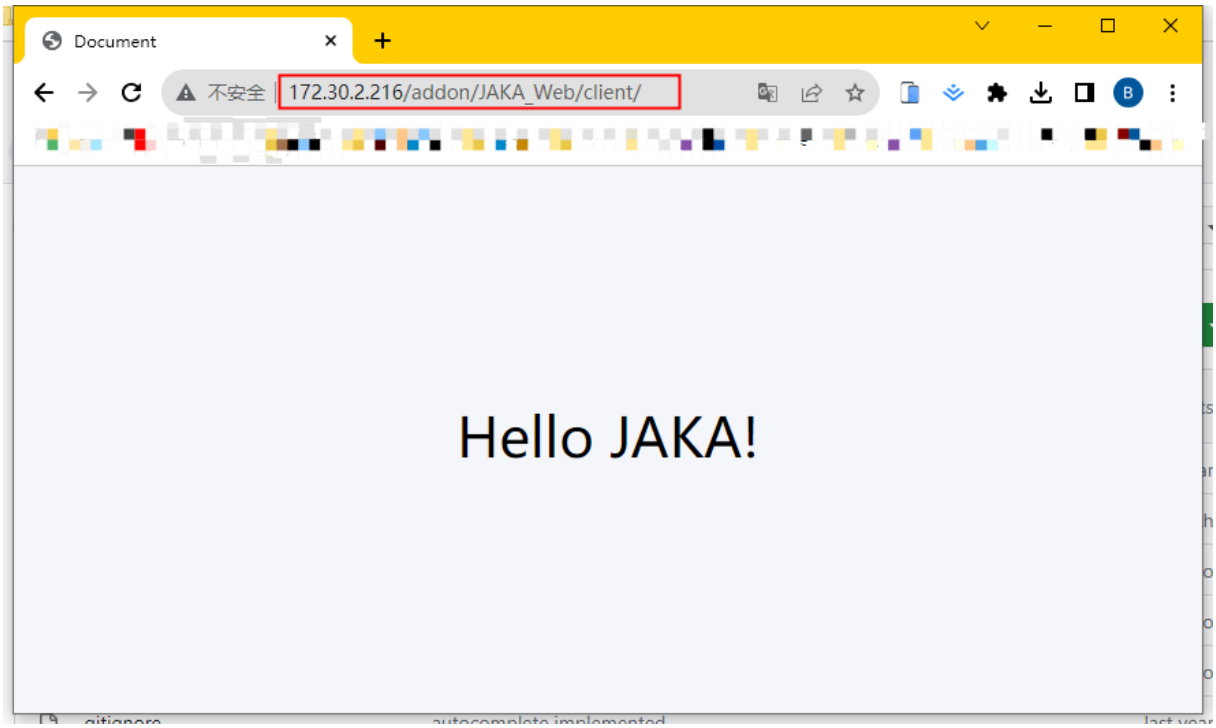
**Step Three: Access the Webpage**

Below are two methods to access the previously created web.

- Method 1: In the AddOn, find JAKA_Web and click the globe icon to access the web.




- Method 2: Enter `http://172.30.2.216/addon/4.3-JAKA_Web/client/` in your browser, replacing the link IP with the current robot's IP.



## 7.3 Compatibility

When developing custom webpages, pay attention to the version of the target browser to avoid incompatibility due to version differences.

- The web view plugin used in JAKA App to display custom webs uses the browser engine `Chromium:74.0.3729.157`.

# 8 Multilanguage Setting

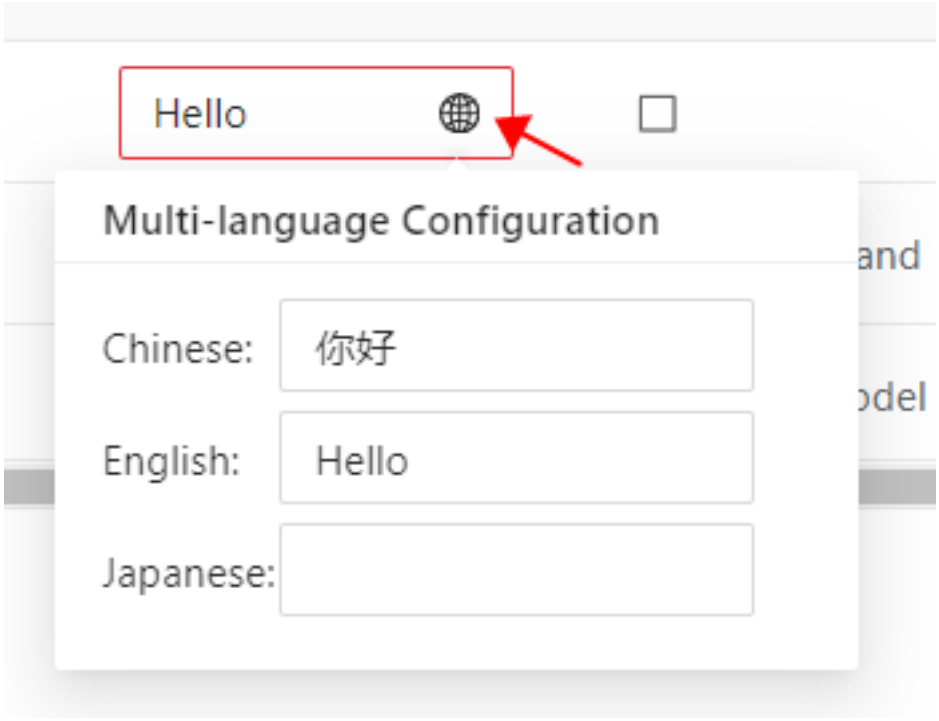
The JAKA AddOn supports multilanguage setting. After completing the development of a certain package's function, you can set up multilanguage options as needed.

Currently supported languages include Chinese, English, and Japanese.

The specific steps for configuration are as follows:

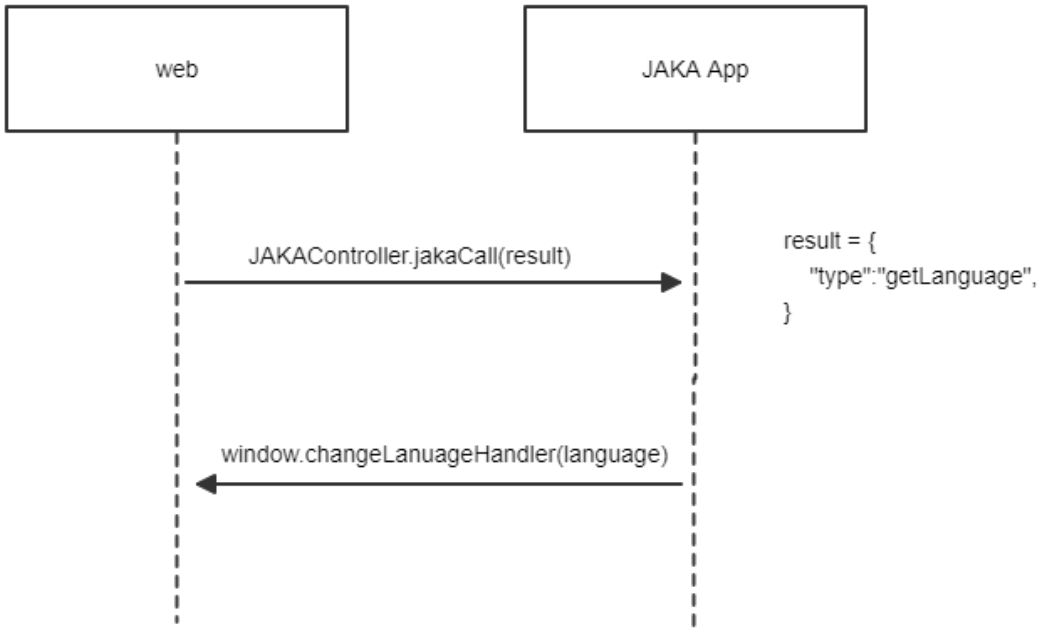
## 8.1 Multilanguage Setting for Custom Commands

When developing custom commands, there will be a small globe icon on the right of the content input box that supports multilanguage. Clicking on it will allow you to configure multilanguage options.




## 8.2 Multilanguage Setting for Custom Interfaces

When a custom interface is opened in the JAKA App, it needs to retrieve the current language of the app in order to synchronize the language of the user's custom page with that of the app. This can be achieved using the following method.




In the web initialization JavaScript code, call the method `JAKAController.jakaCall(result)`. After calling this, the JAKA App will invoke the `window.changeLanguageHandler(language)` function and pass in the current app language.

**Example Code:**

```
let result = {
    "type": "getLanguage",
}

result = JSON.stringify(result);

JAKAController.jakaCall(result);

/**
 * Callback function from the App after triggering getAppLanguage.
 * @param {*} language Current language passed in by the App.
 */
window.changeLanguageHandler = (language) => {
    console.log(`Current language is {language}`)
}
```

# 9 Writing User Guide

**This page aims to guide developers on what content should be included in the user guide for a completed AddOnXXX package, what rules to follow, and how to release it for user download or use.**

Generally, a user-oriented add-on user guide should include the following content to ensure users can install, configure, and use the add-on smoothly:

**Title and Introduction** Specifically includes:

- Add-on Name
- Add-on Version
- Basic Information of the add-on

**Installation Guide** Specifically includes:

- System Requirements

List information about the required operating system, software environment, hardware requirements, etc..

- Installation Steps

Provide step-by-step installation instructions, including any issues that may be encountered during the installation process and their corresponding solutions.

**Configuration Guide** Specifically includes:

- Add-on Configuration Options

List the configuration options that users can customize.

- Configuration Steps

Provide step-by-step configuration instructions, including the role of each configuration option and recommended settings.

- Default Configuration

If the add-on has default configuration options, detailed information about the default configuration should be provided.

**Use Instructions** Specifically includes:

- Main Functions

List the main features and uses of the add-on.

- Examples of Use

Provide practical examples of use to help users understand how to use the add-on in their own workflows.

**Troubleshooting** Specifically includes:

- Frequently Asked Questions

List common problems users may encounter and their corresponding solutions.

- Add-on Logs

If the add-on generates log files, explain how to view and analyze these logs to resolve issues.

**Updates and Upgrades** Specifically includes:

- Update Guide

Provide instructions on how to obtain and install add-on updates.

- Version History

List the version history of the add-on and the changes in each version.

**Support and Contact Information** Specifically includes:

- Provide contact information for support, such as technical support hotline, email address, social media links, or forum links.

**Release**

Provide information about how will the AddOn packages be released and where to obtain them.

**Additional Information** Specifically includes:

- Attachment information, such as plugin dependencies, known issues, or warnings.

**Tip:** Tip:

In addition to the above content, it is important to ensure that the user guide's language should be concise, easy to understand, and suitable for the target reader of the add-on.

## 9.1 How to publish the manual

```
**standalone document**
Currently, JAKA supports to provide the user guide along with the packages in the␣
↪form of PDF.Any changes in the future will be updated here.
```

# 10 Use JAKA Dashboard Template

**This page aims to give a detailed guide on how to use the Dashboard function template to create a JAKA AddOn Webpage.**

We have pre-set some robot operation templates in the JAKA Node-RED Examples. Users can directly use these templates to create webpages, or modify these templates based on their needs. These templates combine the widget (like input, button and dropdown) with JAKA Custom Service together to achieve specific functions.

## 10.1 Import the Template

Users can get all the pre-set templates in Node-RED, and directly use them after deployment. Users can also import the required widgets in different templates for combination, deployment and use.

Video: 如何找到组件.mp4

## 10.2 What You Should Know About Dashboard

### 10.2.1 Layout

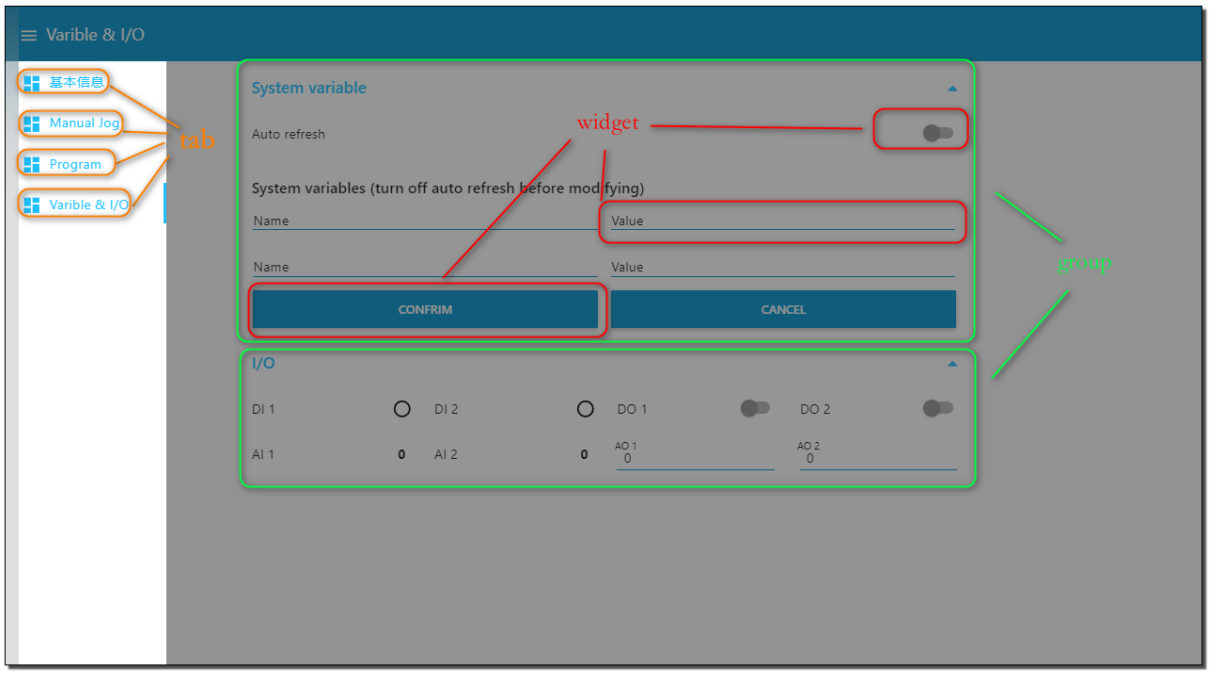
The dashboard layout should be considered as a grid.

Every **Tab** can be seen as a webpage. One Tab can contain multiple Groups, and one Group can contain multiple Widgets. When editing Widget, users can designate its Group. When editing Group, users can designate its Tab.

Each **Group** element has a width - by default 6 'units' (a unit is 48px wide by default with a 6px gap).

Each **Widget** in the group also has a width - by default, 'auto' which means it will fill the width of the group it is in, but you can set it to a fixed number of units.

The layout algorithm of the Dashboard always tries to place items as high and to the left as they can within their container - this applies to how groups are positioned on the page, as well as how widgets are positioned in a group.




When the template has been imported, the widgets within might be displaced. Users can reset its layout, or adjust it to whatever is preferred.

Video: layout 调整.mp4

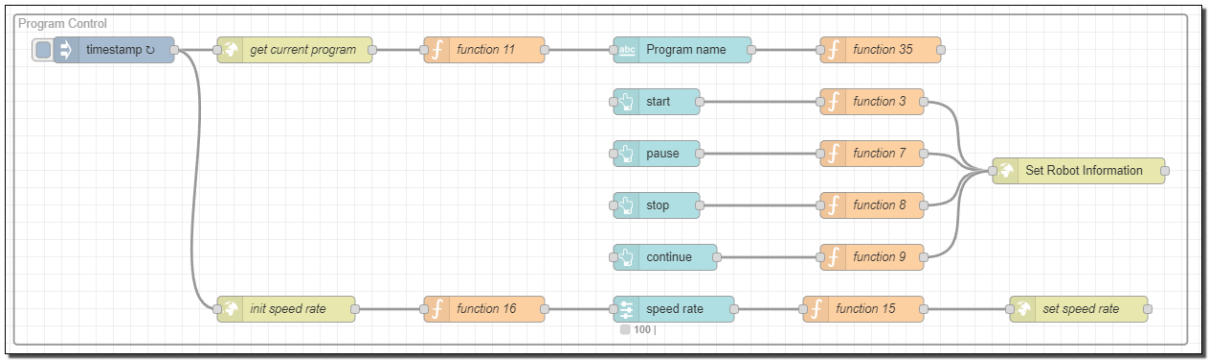
## 10.3 List of Templates

### 10.3.1 Control the Program

**To start, pause, resume and stop the program**

This template contains the use of input, button and slider.

Video: 程序控制演示.mp4

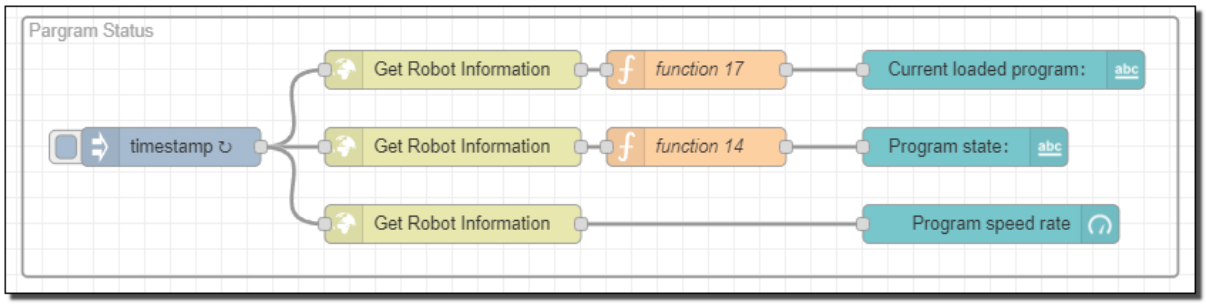


### 10.3.2 Check the Status of the Program

**To check the currently loading program and its running speed and status.**

This template contains the use of gauge.

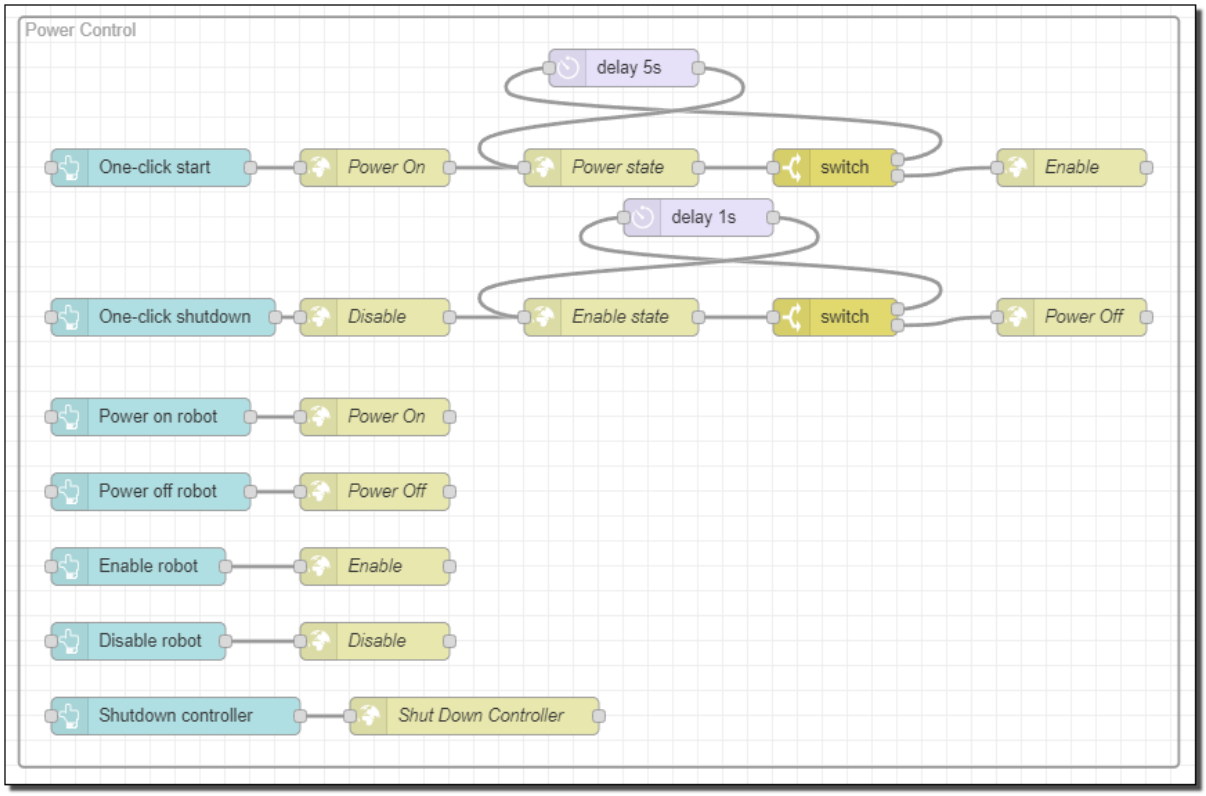
Video: 程序状态演示.mp4




### 10.3.3 Control the Power Supply

**To control the robot’ s power supply and status**

Video: 电源控制演示.mp4




The “One-click start up” would power on the robot first. After so, it automatically cycles to detect whether the robot has been successfully powered on or not. If the answer is yes, it would continue to enable the robot. Users can modify the time period of cyclic detection in “delay”.

### 10.3.4 Robot Status

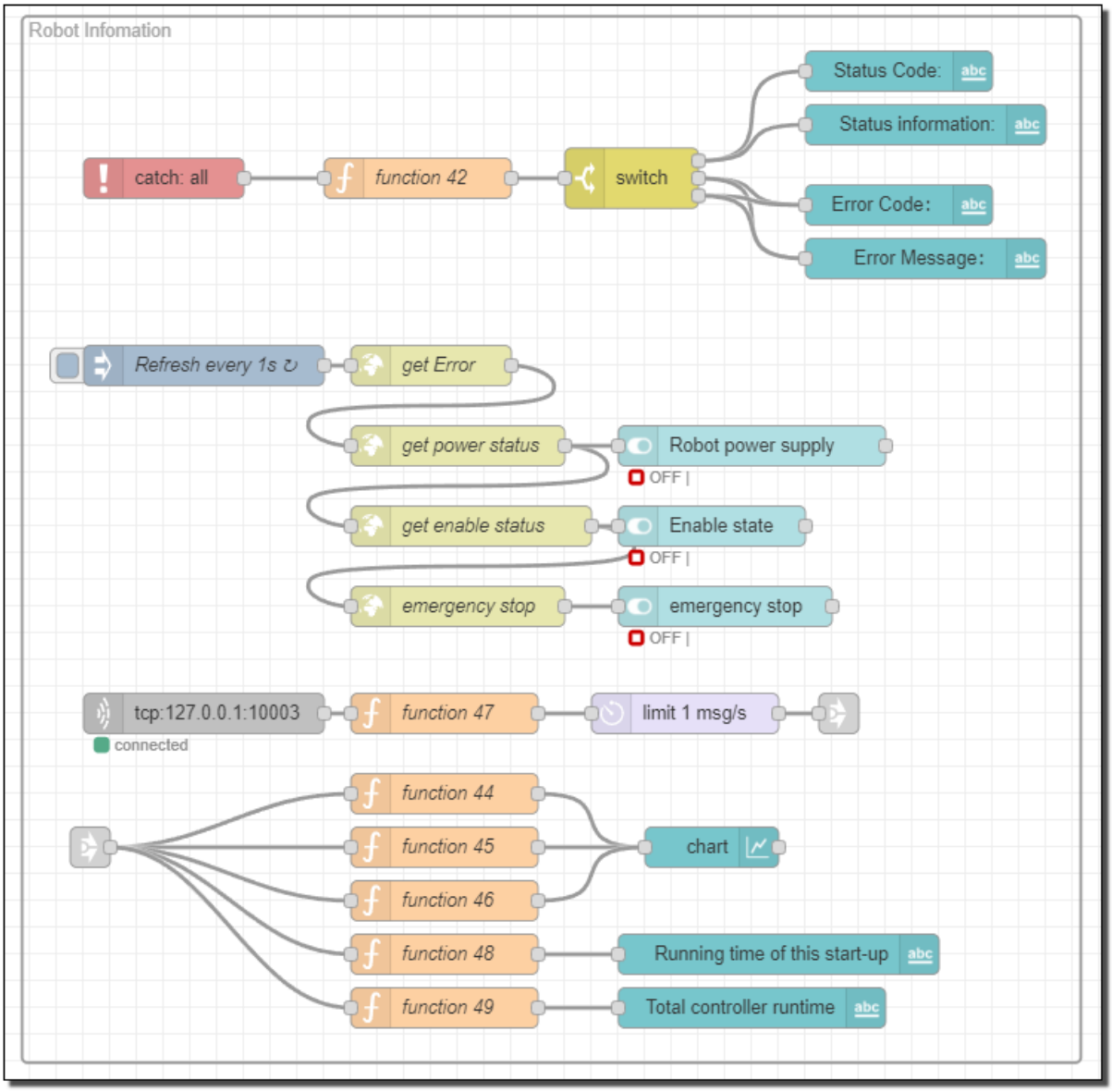
**To monitor the robot’s various statuses**

This template contains one widget that can catch every error code and status code of the robot since it is started up. And one line chart widget that can show the status of the robot, such as the controller’s temperature and the robot’s speed, etc.

**Tip:** Note

This demo video is recorded using a virtual machine, therefore, the data on the controller’s temperature and power are empty.

Video: 机器人信息演示.mp4

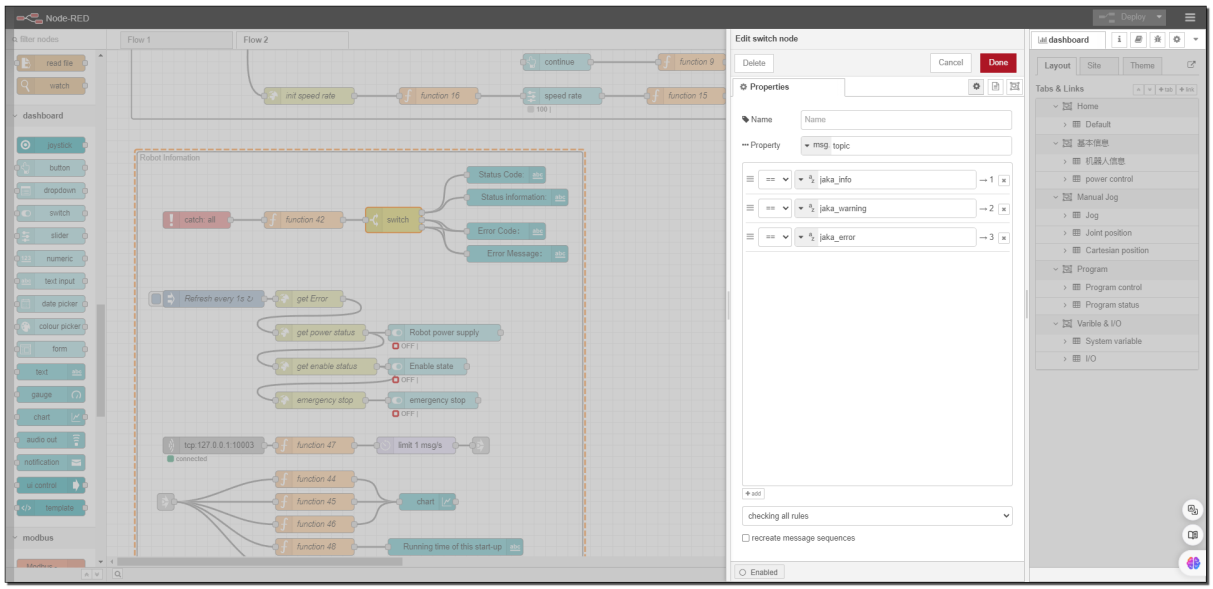



**To catch the error codes and status codes of the robot**

In order to thoroughly catch the required information, users are suggested to use the `catch` to catch all error codes and status codes while using the "Get robot information" node to refresh these error codes and status codes. Then use the `switch` node to group them for display on the page.

The codes caught by "Get robot information", "Set robot information" and "Robot motion control" have three **topic**:

- jaka_info: information
- jaka_warning: waring
- jaka_error: error




### Status Icon

Use the **Switch** widget in Dashboard to select different icons to present different status.

### Information on the Chart

The information gained from port *10003* can be monitored by `tcp in`. To prevent the page from crashing due to frequent refresh, users can use `delay` node to regulate the amount of passed information.

Then use the `link out` node to send the monitored information. Afterwards, users can use `link in` node to receive information send by port 10003 in Node-RED.

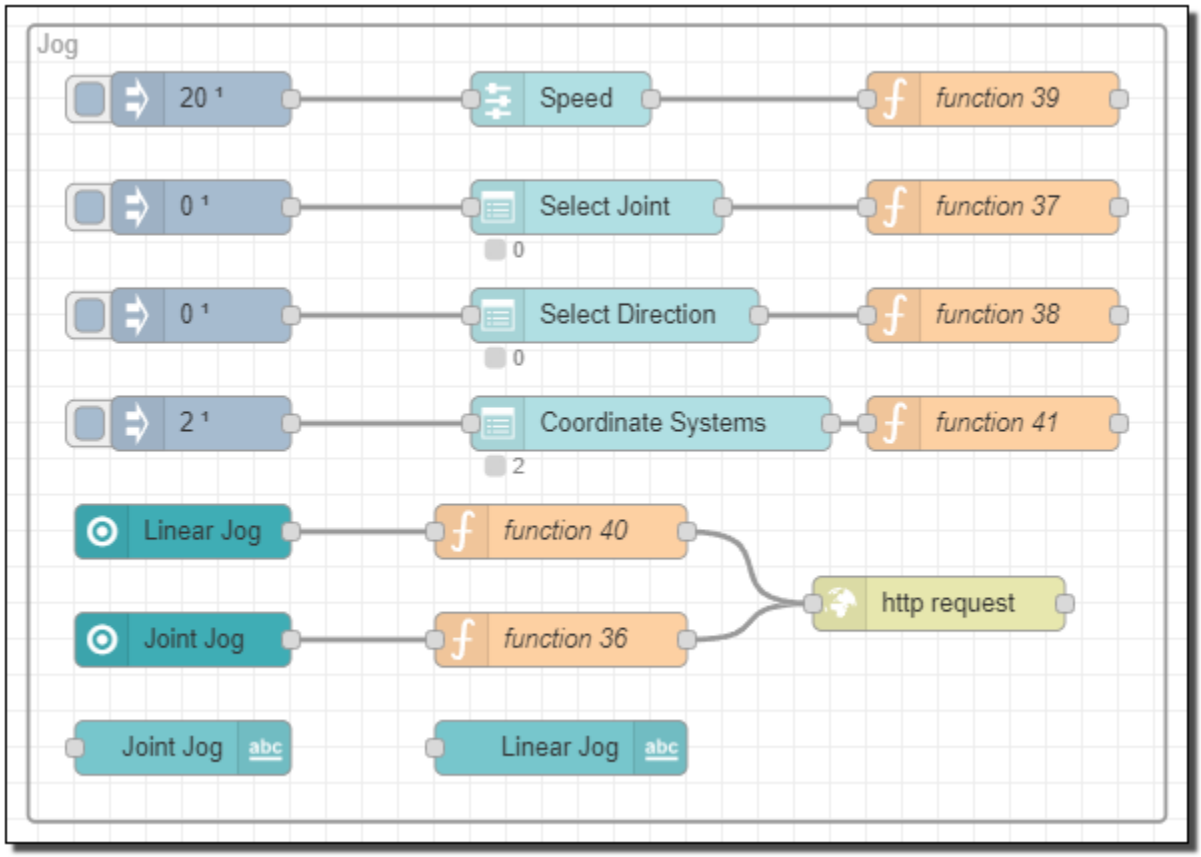
Port 10003 pushes data with timestamp every 80ms, making it very suitable to get data that requires high frequency of refresh and in-time updates, such as the current location, power limit, torque, etc.

### 10.3.5 Jog the Robot

**Using the Joystick to Control the Robot Movement**

This template contains joystick widget and dropdown widget.

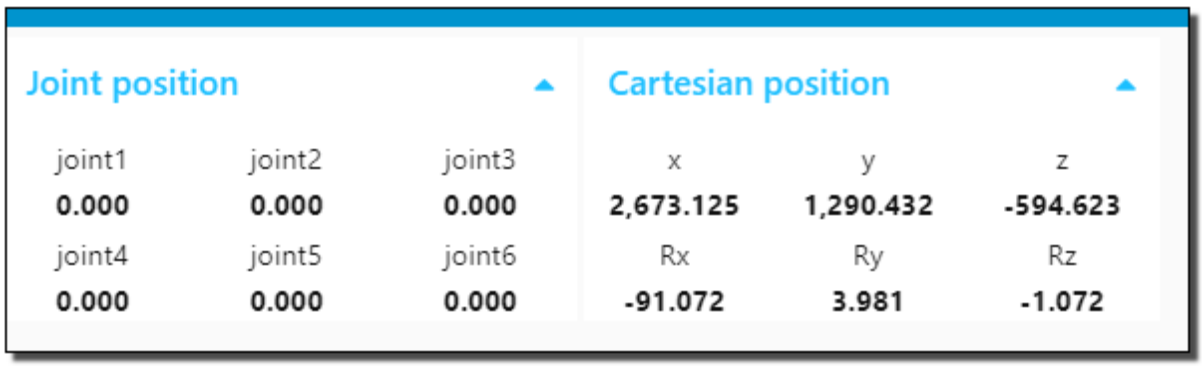
Video: 机器人 Jog 演示.mp4

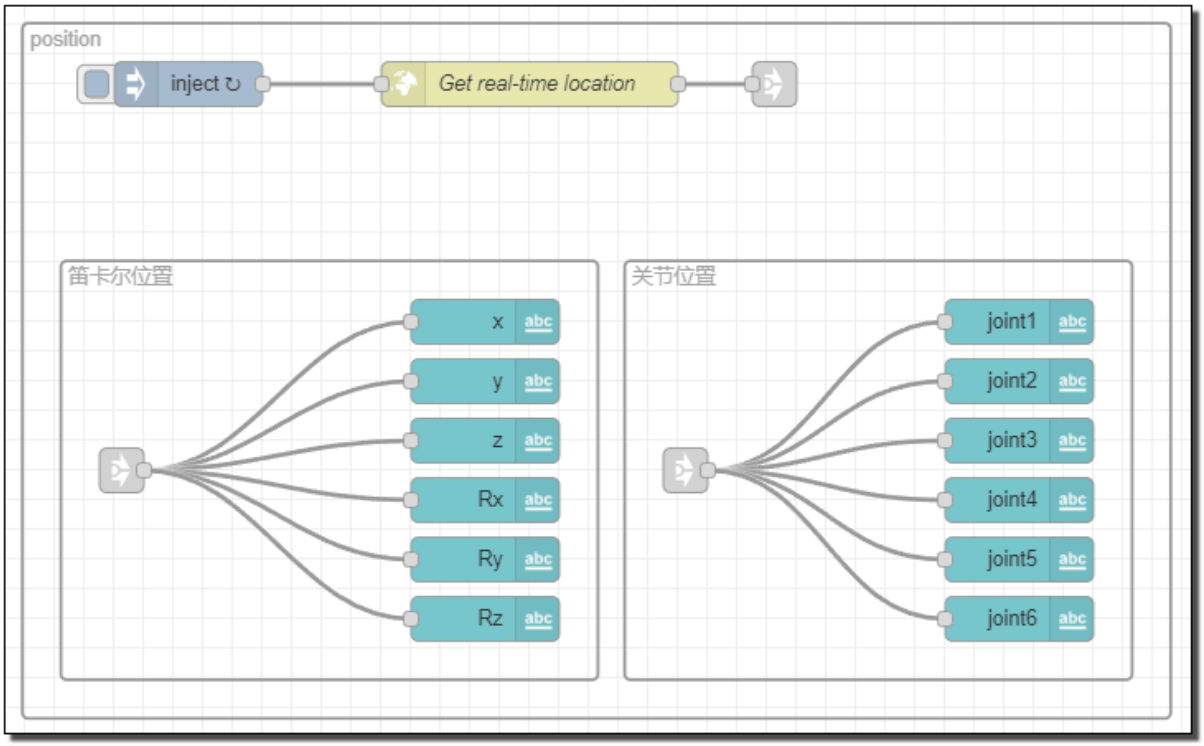



This template supports Linearity Jog and Joint Jog. Only one direction or joint can be jogged at one time. In linearity jog, the world coordinate system and TCP coordinate system can be switched at anytime.

### 10.3.6 Get the Robot Position

**To Get the In-time Position of the Robot**



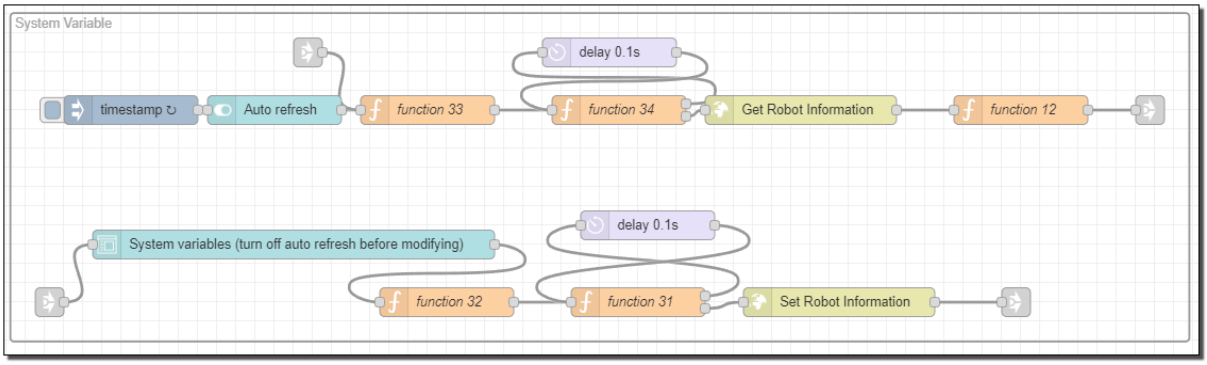



This template uses JAKA customized `Get robot information` node to get the robot's position. But the time lag between the request and the receiving of position data cannot be assured, given that the http is used to request. If the user needs to get the position with high frequency, or needs to get positions with an accurate timestamp, data from port 10003 is suggested.

### 10.3.7 System Variables

**To monitor and set the value of system variables. The monitored data is accurate when the program is running.**

Video: 系统变量演示.mp4




This template can monitor and set two variables by default. If more variables are needed, please modify the system variable form.

Video: 新增系统变量.mp4

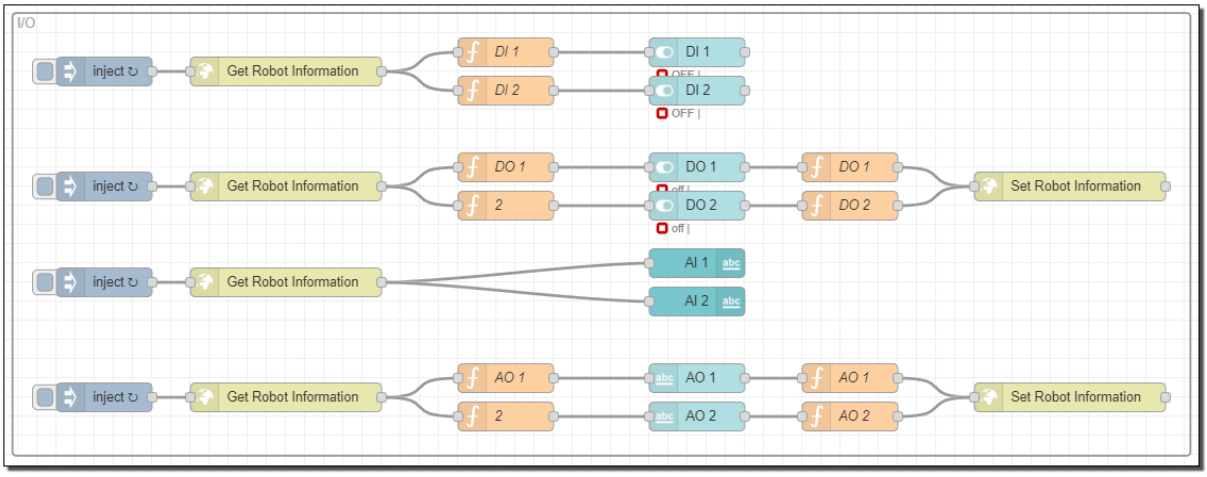
**Warning:** Note

The name of new added variables needs to start with `varName varValue` and a number. The number shows the sequence of these new added variables.

### 10.3.8 I/O Panel

**To monitor and set the I/O value**

Video: IO 面板演示.mp4




**Add I/O numbers**

The video below illustrates how to add more I/Os.

Video: 新增 IO.mp4

**Switch the I/O source**

The video below illustrates how to switch the source of DI from *Modbus slave* to *control cabinet*.

Video: 切换 IO 来源.mp4

# 11 Release

**Coming Soon**

# 12 Case Study - DH_Grapper

**Coming Soon**

# 13 Case Study - Ewellix LiftKit

**Coming Soon**

# 14 AddOn Configuration File Specification

## 14.1 1. File Encoding Requirement

UTF-8 encoding is required.

## 14.2 2. File Naming Requirement

“AddOnName”+”_config.ini”, for example, JAKA_Command_config.ini.

**Tip:** Note:

The “AddOnName” here must be identical to the name of the AddOn folder. Otherwise, the AddOn will not start properly.

## 14.3 3. File Content

A typical AddOn configuration file is shown below:

```
[AddOnInfo]
convention = 3.0
name = JAKA_Command
description = "Output information in logs"
version = 1.0
type = 1
portal = 10006
url = http://localhost/myAddOnUi
languagetype = node-red
service = AddOn.json
serviceenabled = 1
```

## 14.4 4. Field Descriptions:

```
``convention``
AddOn package parser version. If not filled or the parser version is incorrect, an
↪"Operation Invalid, AddOn Failed" prompt will appear when the App attempts to␣
↪activate this AddOn.
``name``
The name of the AddOn package, which will automatically synchronize with the name␣
↪of the AddOn folder when the AddOn is launched. It will be displayed as "Program␣
```

(continues on next page)

(continued from previous page)

```
↪Name" in the App's add-ons interface.
``description``
Functional description of the AddOn, displayed as "Description" in the App's add-
↪ons interface. It is limited to 35 characters. If it exceeds this limit, the
↪"Version" field in the App's add-ons interface will display "INVALID", and the
↪"Description" will be empty. An "Operation Invalid, AddOn Failed" prompt will␣
↪appear if the App attempts to activate this AddOn.
``version``
The version number of the AddOn itself, which can be customized. It corresponds to␣
↪the "Version" in the App's add-ons interface.
```

```
``type``
Type of AddOn: 1 for custom command, 2 for custom service, 3 for custom page. If 1␣
↪or 2 is filled, there will be no globe button in the App's add-ons interface. If␣
↪3 is filled, a globe button will appear. If left blank or filled with other␣
↪numbers or characters, the "Version" in the App's add-ons interface will display
↪"INVALID", and the "Description" will be empty. An "Operation Invalid, AddOn␣
↪Failed" prompt will appear if the App attempts to activate this AddOn.
``portal``
The port assigned automatically within the controller to the current AddOn. No user␣
↪input is required and it can be left blank.
``url``
The URL to which the globe button in the AddOn operation options on the additional␣
↪program page will redirect. This is the path to the entry file ``index.html`` in␣
↪the frontend project of the AddOn. For example, if the ``index.html`` of the␣
↪AddOn frontend project is located in the folder ``JAKA_Command/client``, you can␣
↪fill in ``url = http://localhost/JAKA_Command/client``. The ``localhost`` field␣
↪will be automatically replaced with the current robot's IP address during␣
↪interpretation. If left blank, an "Operation Invalid, AddOn Failed" prompt will␣
↪appear when the App attempts to activate this AddOn. If the filled path is␣
↪incorrect, a "Load failed, please check the path" prompt will be displayed when␣
↪clicking the globe button.
``languagetype``
The backend language used, currently only supports using node-red as the backend.
``service``
Specify the filename in JSON format. This file comes from the json file in the␣
↪AddOn file directory, which contains the flow and node information defined in␣
↪node-red. The file name can be customized, but the two must match. If the names␣
↪do not match, an "AddOn Service does not exist" prompt will appear when the App␣
↪attempts to activate this AddOn.
``serviceenabled``
Addon startup at boot, 0 means this AddOn package is in the closed state at startup,
```

(continues on next page)

(continued from previous page)

```
↪ 1 means this AddOn package is in the open state at startup. If left blank or␣
↪filled with other numbers or characters, an "Operation Invalid, AddOn Failed"␣
↪prompt will appear when the App attempts to activate this AddOn.
```

# 15 Interaction between AddOn Web Pages and the App

When a custom web page from AddOn is opened in the JAKA App, the JAKA App provides the `JAKAController.jakaCall(result)` method for the web page to call. Additionally, developers can implement specific methods on the web page for the App to callback.

## 15.1 JAKAController.jakaCall()

`jakaCall()` is implemented by the App and registered when the browser is opened within the App. When the AddOn web page calls `jakaCall()`, it passes different objects to achieve different functionalities.

This object will always contain the property `type` and may contain `data`.

```
{
    "type":"",
    "data":"{\"parameter_name\":\"parameter_value\"}",
}
```

**Warning:** Note!

```
The value of `data` must be a serialized object.
```

### Passing Parameters to an Instruction Block

```
{
    "type":"saveJakaEditorItem",
    "data":"{\"property_name\":\"property_value\"}",
}
```

### Close Browser Window

```
{
    "type":"close",
}
```

### Open Browser Developer Tools Window

```
{
    "type":"showDevTools",
```

(continues on next page)

(continued from previous page)

```
}
```

**Open Robot Teaching Page**

```
{
    "type":"jakaMoveRobot",
    "data":"{
        \"pose\":\"cartesian_coordinates\",
        \"jointpose\":\"joint_angles\"
    }",
}
```

When the teaching page is saved, the App will call *window.jakaCallBack(data)* and pass the current position information.

**Get the Current App Language**

```
{
    "type":"getLanguage",
}
```

After calling this, the App immediately callbacks *window.changeLanguageHandler(language)* with the current language.

## 15.2 window.jakaCallBack()

This function is called by the App when the save button is clicked on the teaching page, and it passes the current position information.

```
/**
 * Callback function for when the teaching page is closed
 * @param {*} data Contains the current teaching information
 */
window.jakaCallBack = function(data){
    console.log("jakaCallBack Info",data)

    if(data){
      console.log("jointPose",data.data.jointPose)
    }
  }
```

## 15.3 window.changeLanguageHandler()

After calling the method to get the current language, the App immediately calls this function and passes the language being used by the App.

```
/**
 * Trigger getAppLanguage callback function in the app.
 * @param {*} language The current language passed by the app
 */
window.changeLanguageHandler = (language)=>{
    console.log("changeLanguageHandler",language)
}
```



JAKA Robotics Co., Ltd.
Address: Building 6, No. 646 Jianchuan Road, Minhang District, Shanghai, China
Email: docs-support@jaka.com
Website: www.jaka.com