

---

# **Mech-Center Manual**

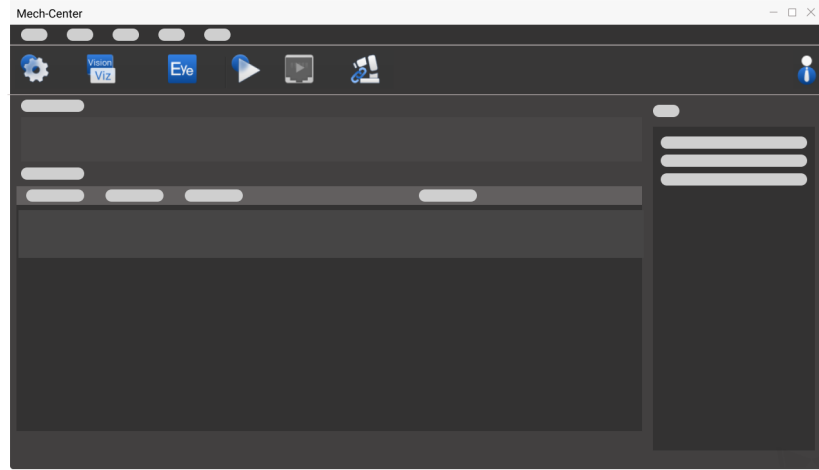
**Mech-Mind**

**Dec 23, 2022**

# CONTENTS

<b>1</b>	<b>Quick Facts of Mech-Center</b>	<b>2</b>
1.1	Menu Bar . . . . .	3
1.1.1	File . . . . .	3
1.1.2	Tool . . . . .	3
1.1.3	User . . . . .	4
1.1.4	View . . . . .	4
1.1.5	Help . . . . .	4
1.2	Toolbar . . . . .	4
1.2.1	Deployment Settings . . . . .	4
1.2.2	Start Viz/Vision . . . . .	9
1.2.3	Start Mech-Eye Viewer . . . . .	9
1.2.4	Run . . . . .	9
1.2.5	Start Interface . . . . .	9
1.2.6	Connect Robot . . . . .	9
1.2.7	Administrator . . . . .	9
1.3	Service Status Bar . . . . .	11
1.4	Project Status Bar . . . . .	12
1.5	Log Panel . . . . .	13
<b>2</b>	<b>Getting Started with Mech-Center</b>	<b>14</b>
2.1	Deployment Settings . . . . .	14
2.2	Open Projects . . . . .	16
2.2.1	Open Mech-Viz Project . . . . .	16
2.2.2	Open Mech-Vision Project . . . . .	17
2.2.3	Configure Camera Settings . . . . .	17
2.3	Connect the Robot . . . . .	18
2.4	Run the Project . . . . .	18
2.5	Example Mech-Viz Projects for Standard Interface . . . . .	19
<b>3</b>	<b>Mech-Interface</b>	<b>20</b>
3.1	Mech-Interface Overview . . . . .	20
3.1.1	Communication Mechanism . . . . .	21
3.1.2	Differences between Standard Interface and Adapter . . . . .	21
3.1.3	Application Scenarios . . . . .	22
3.2	Standard Interface . . . . .	23
3.2.1	Instructions . . . . .	24
3.3	Standard Interface Development Manual . . . . .	25
3.3.1	Overview . . . . .	25

3.3.2	TCP/IP	30
3.3.3	Siemens PLC	56
3.3.4	PROFINET	69
3.3.5	EtherNet/IP	92
3.3.6	Modbus TCP	92
3.3.7	Status Codes and Trouble Shooting	109
3.3.8	Appendix	139
3.4	Adapter	142
3.4.1	Quick Facts of Adapter	143
3.4.2	Adapter Generator Guide	146
3.4.3	Adapter Programming Guide	154
3.4.4	Adapter Programming Examples	193



Mech-Center is the **Control Center** of the Mech-Mind Software Suite independently developed by our company. With an intuitive interface, Mech-Center enables you to implement the global settings of the Mech-Mind Software Suite, backup and restore the whole project, and check the status of Mech-Viz, Mech-Vision, Mech-Eye Viewer, the robot, standard interface, and Adapter. It can also be used to activate and manage Mech-Interface.

---

Please refer to the section below to learn about the **User Interface and Functions** of Mech-Center.

*Quick Facts of Mech-Center*

---

Please refer to the section below to learn about the **Basic Instructions on Using Mech-Center**.

*Getting Started with Mech-Center*

---

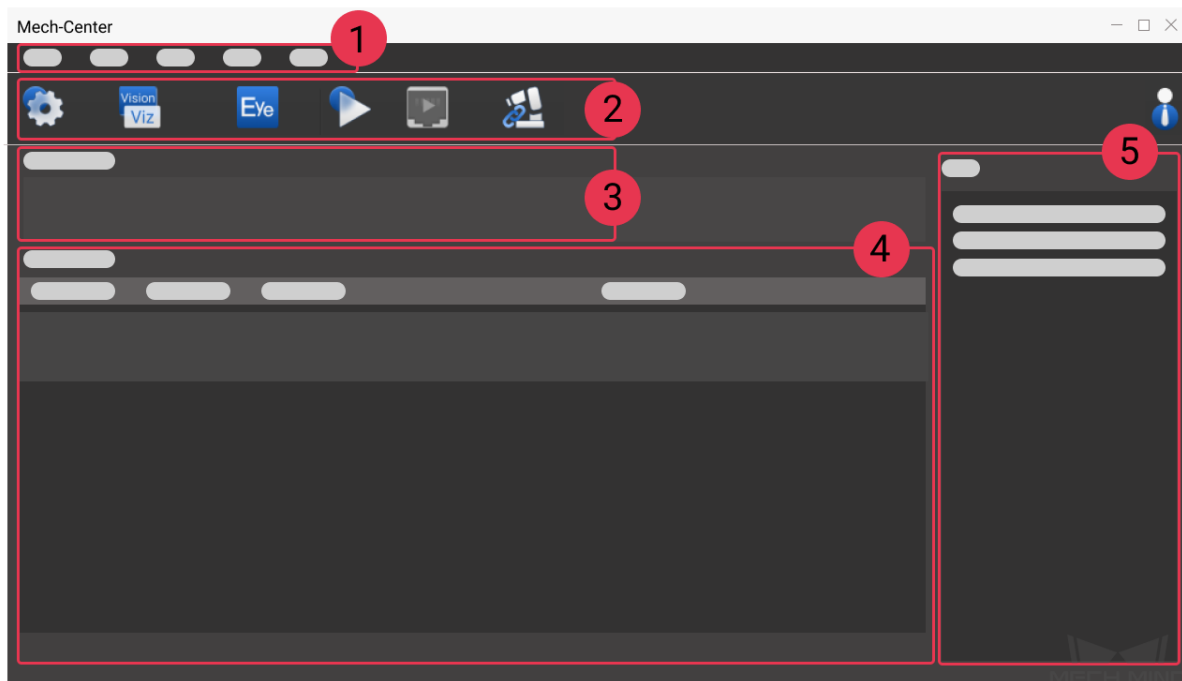
Please refer to the section below to learn about **Mech-Interface**.

*Mech-Interface*

## QUICK FACTS OF MECH-CENTER

Mech-Center is the **Control Center** of the Mech-Mind Software Suite independently developed by our company. With an intuitive interface, Mech-Center enables you to implement the global settings of the Mech-Mind Software Suite, backup and restore the whole project, and check the status of Mech-Viz, Mech-Vision, Mech-Eye Viewer, the robot, standard interface, and Adapter. It can also be used to activate and manage Mech-Interface.

The main interface of Mech-Center consists of 5 parts:



1. *Menu Bar* : Provides functions for managing projects, modifying user interface and view, generating Adapter, checking software version, etc.
2. *Toolbar* : Provides functions for deployment settings, starting Mech-Viz and Mech-Vision, connecting robot, and running the project.
3. *Service Status Bar* : Displays registered software, camera, robot, etc.
4. *Project Status Bar* : Displays the status, execution time, and details of Mech-Viz/Mech-Vision projects.

5. *Log Panel* : Displays the log of current projects and services in real time.

## 1.1 Menu Bar

Menu Bar consists of File, Tool, User, View, and Help, which provide basic functions.

File Tool User View Help

### 1.1.1 File

Options	Description	Shortcut
Backup Projects	Used to backup the project data of Mech-Mind Software Suite. It will save the Mech-Viz and Mech-Vision projects which are set to autoload, and Adapter projects and deployment settings of Mech-Center.	Ctrl+B
Restore Projects	Used to restore the selected backup project and settings. The autoload project will be replaced after restoring. Please make sure to backup the needed project.	Ctrl+R
Import Projects	Used to import projects from other paths.	Ctrl+I
Exit	Exit Mech-Mind Software Suite system.	Ctrl+Q

**Attention:** In Operator mode, only Backup Projects and Exit are available.

### 1.1.2 Tool

Options	Description
Pack Debug Data	Used to pack debug data of Mech-Mind Software Suite. You can customize the time period, package path, and save options.
Adapter Generator	Used to generate customized Adapter program.
Log Viewer	Used to view the log information of Mech-Viz / Mech-Vision. Please select the log file and then load it to view. You can view the log information you need by selecting a log level, entering a key word to filter, or searching. Click and drag the scrollbar at the bottom to view the log information after the specified point of time only.
Show Service Status	Used to display the statuses of the software, robot, interfaces, and other services.

**Attention:** The Adapter Generator option is not available in Operator mode.

### 1.1.3 User

Modify Password: Only available in Administrator mode. Used to modify password. An operator cannot use this function.

**Attention:** If the Mech-Center software is installed on the IPC for the first time, the default password of the Administrator is "123456". After modifying the password, please keep it safe and private.

### 1.1.4 View

Log: Check to display Log in the main interface.

### 1.1.5 Help

Options	Description	Shortcut
User Manuals	Used to open the user manual of standard interface quickly.	F1
Release Notes	Display the information about new features and bug fix of each versions.	N/A
About	Display the version and copyright information of the software.	N/A

## 1.2 Toolbar

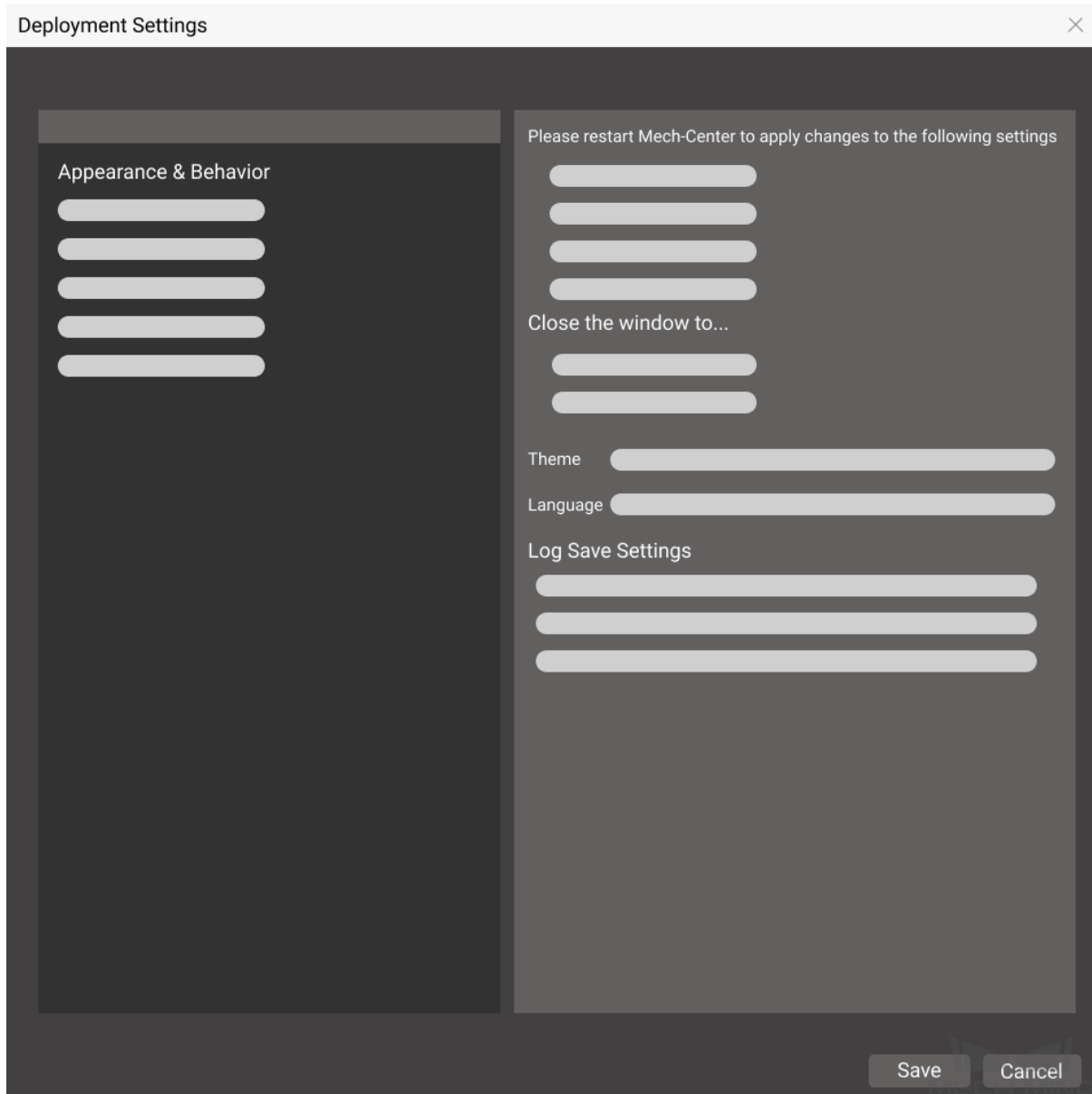
The Toolbar of Mech-Center includes seven buttons, which are Deployment Settings, Start Viz/Vision, Start Mech-Eye Viewer, Run, Start Interface, Connect Robot, and Administrator.

### 1.2.1 Deployment Settings

Deployment Settings is used to implement basic settings of Mech-Center, configure the path of each software, view the path of autoloading projects, select external services, etc.

## Appearance and Behavior

The Appearance and Behavior option is used to customize global settings, as shown below. You can configure relevant settings based on your own behavior, such as changing the theme color, switching interface language (Chinese, English, Japanese, and Korean are now available), changing log save settings, etc.



- If *Hide console when running* is checked, the console will be hid after opening Mech-Center.
- If *Run Mech-Center at PC startup* is checked, Mech-Center will be opened automatically after starting the PC.
- If *Open Mech-Viz/Mech-Vision/Mech-Interface automatically (if any)* is checked, Mech-Viz/Mech-Vision/Mech-Interface will be opened automatically after Mech-Center is opened.

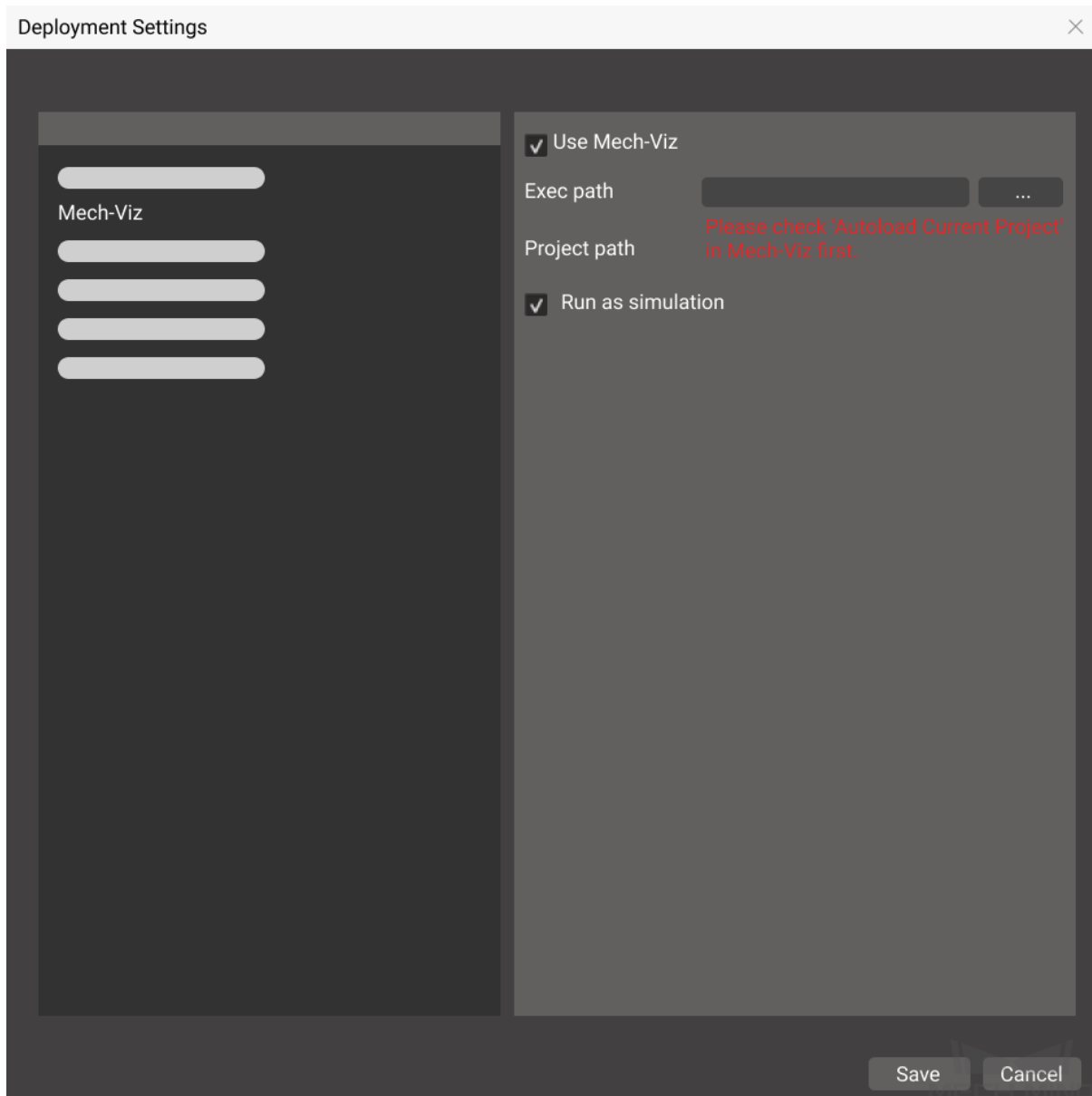


- If *Minimize Mech-Viz/Mech-Vision to System Tray when opening* is checked, the opened Mech-Viz/Mech-Vision will be added in the system tray on the desktop.

**Attention:** Please save the changes and restart Mech-Center for the changes to take effect.

## Mech-Viz

The Mech-Viz option is used to set the open path of Mech-Viz software and display the path of autoloading projects, as shown below.



Click on *Mech-Viz* on the left to start setting. *Use Mech-Viz* is checked by default.

Click on  $\dots$  next to the Exec path, and select the *mmind\_viz.exe* file in the directory where Mech-Viz is installed to complete the path configuration.

Check *Autoload Current Project* in Mech-Viz, and the project path will be added automatically.

---


**Note:** If **Run as simulation** is checked, Mech-Viz will only simulate the project and will not guide the real robot to move.

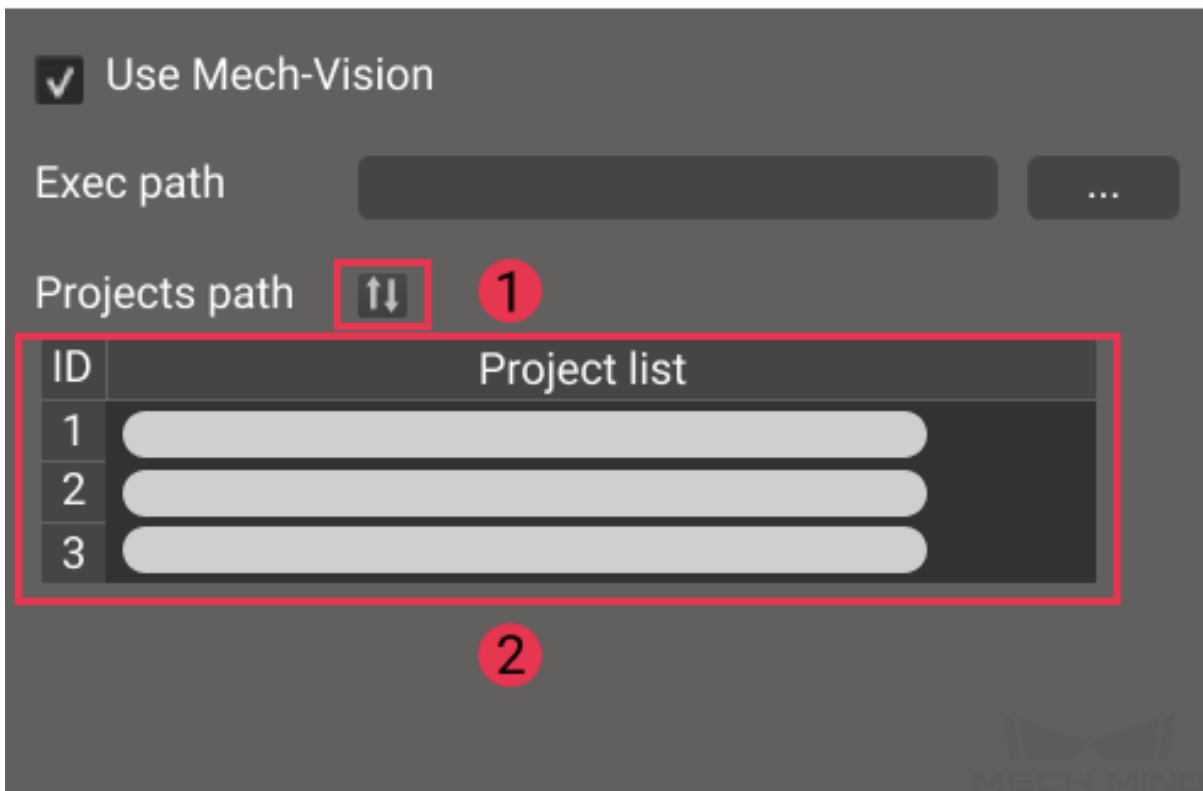
---

### Mech-Vision, Mech-Eye Viewer

The methods to configure Mech-Vision and Mech-Eye Viewer are similar to that of Mech-Viz.

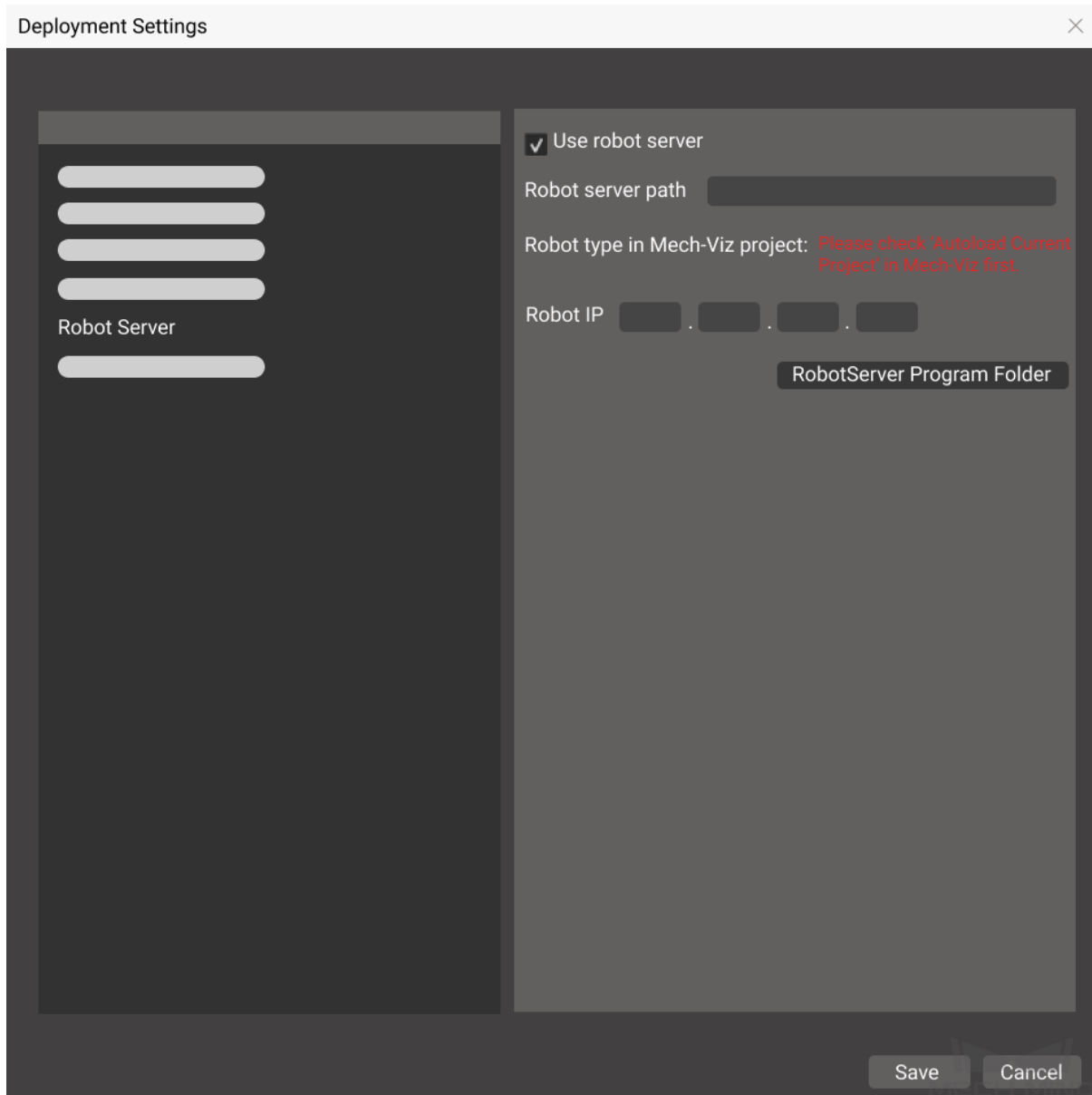
Please follow the instructions below to adjust the sequence of project path in the project list:

1. Open Mech-Vision first, select the project in the **Projects List** and right-click with the mouse, and then check *Autoload Project* in the context menu.
2. Open Deployment Settings in Mech-Center, and click on  to synchronize the Mech-Vision project paths.
3. Press an hold the left mouse button on a project to drag up or down to adjust the sequence.



## Robot Server

The Robot Server option is used to adapt the robot to realize the full control of Mech-Viz software, as shown below.



Click on *Robot Server* to start setting. *Use robot server* is checked by default. The robot server files are in the directory where Mech-Center is installed, and the robot server path will be automatically added.

**Attention:** After loading the Mech-Viz project successfully, the robot type in Mech-Viz project will be filled automatically. Please make sure to enter the correct robot IP which is set in the actual project.

## Mech-Interface

*Mech-Interface* is an unified external interface that realizes communication with third parties.

### 1.2.2 Start Viz/Vision

After opening Mech-Viz/Mech-Vision successfully, their icons will be displayed on the Service Status Bar.

**Attention:**

1. A version compatibility check will be performed when running a project. It is recommended to use Mech-Viz, Mech-Vision, and Mech-Center of versions 1.4.0 and above.
2. When Mech-Center detects that the version of either Mech-Viz or Mech-Vision is lower than 1.4.0, it will prompt that the version must be higher than 1.4.0, and an error message box will pop up after clicking *Run*.

### 1.2.3 Start Mech-Eye Viewer

After opening Mech-Eye Viewer successfully, its icon will be displayed on the Service Status Bar.

### 1.2.4 Run

Run the loaded projects in Mech-Viz and Mech-Vision.

### 1.2.5 Start Interface

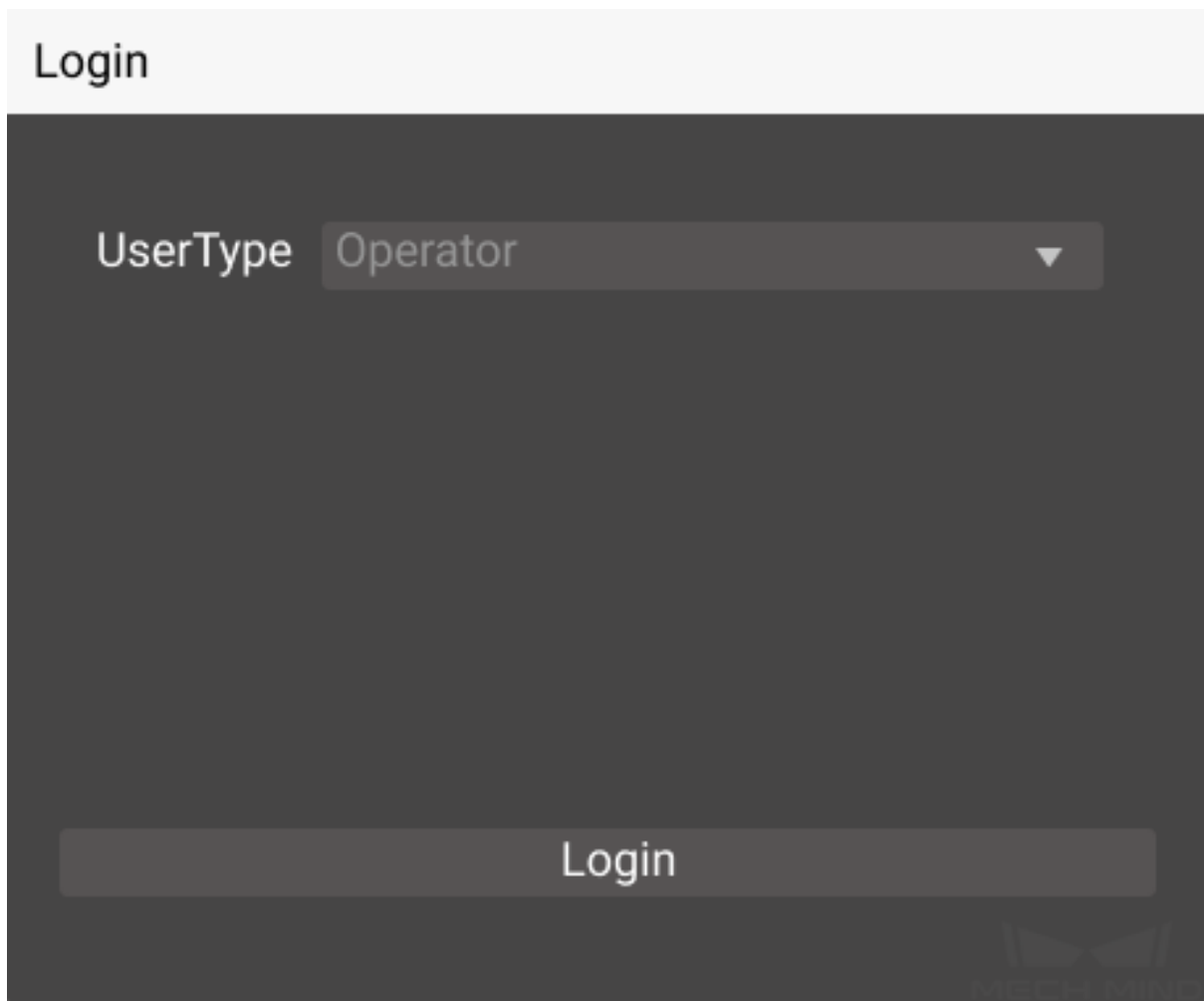
Used to start *Mech-Interface*.

### 1.2.6 Connect Robot

After connecting a real robot successfully, the robot icon will be displayed on the Service Status Bar.

### 1.2.7 Administrator

Mech-Center can be used in either Administrator mode or Operator mode, and the default one is Administrator mode. Operator mode does not enable to edit projects or adjust configurations, and the icon displayed on the toolbar will be the Operator. If you need to switch the mode, please click on the icon, select the user type and click on *Login* to confirm setting.



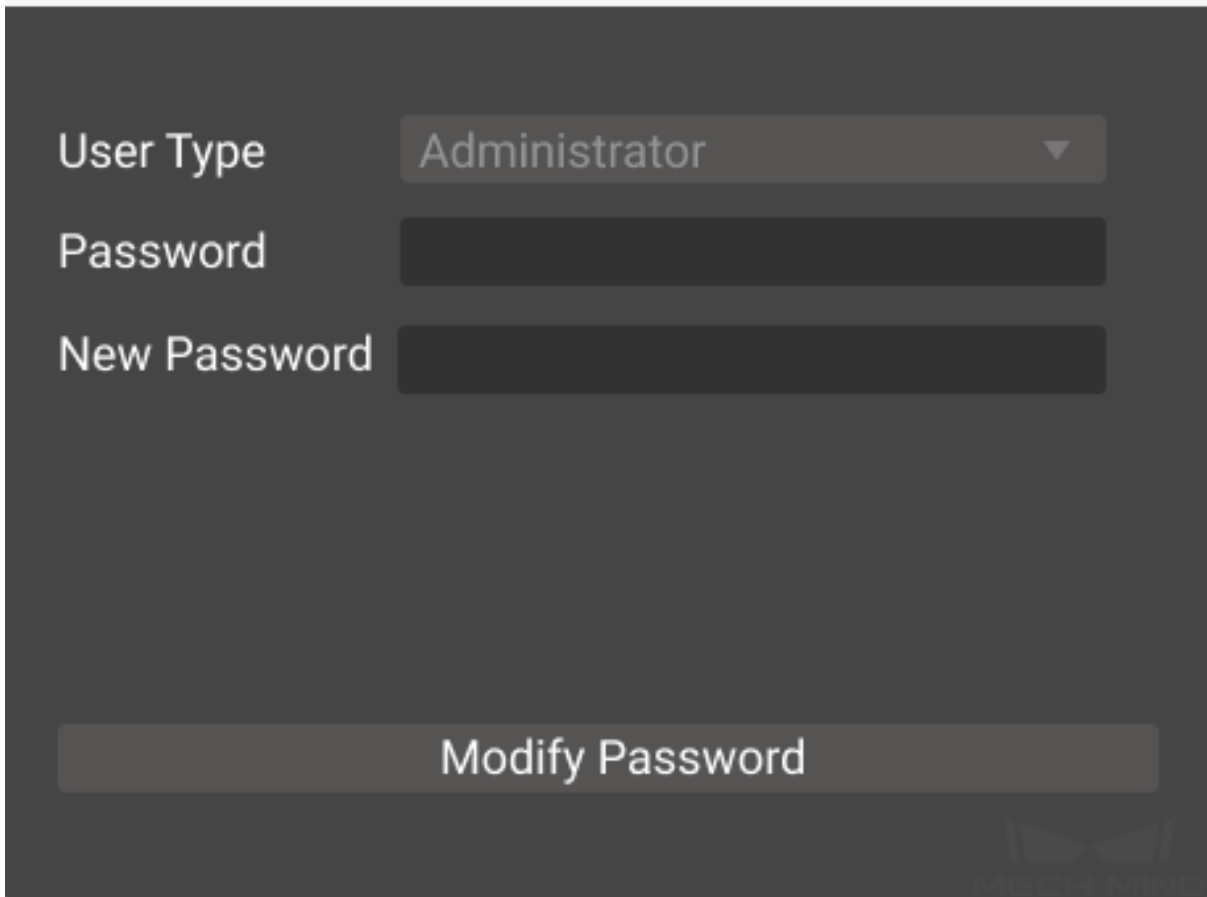
Login

UserType Operator ▼

Login

The Administrator mode requires a password to log in. If you need to modify the password, please go to *User → Modify Password*.

## Modify Password



User Type Administrator

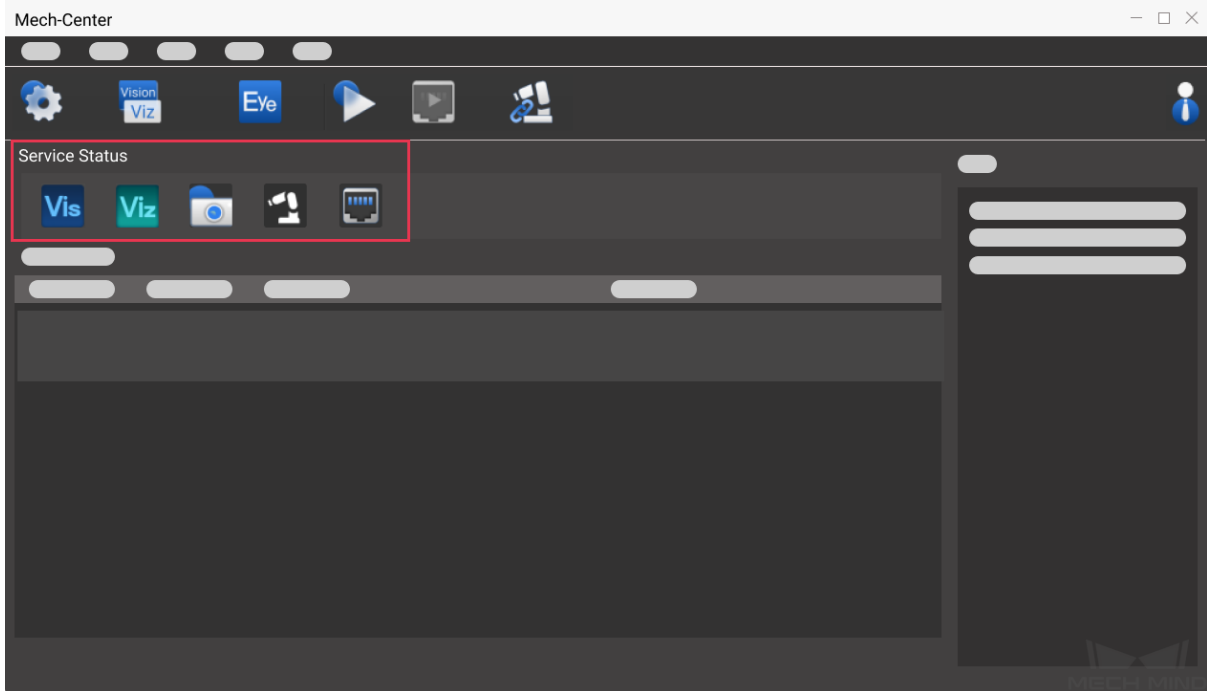
Password

New Password

Modify Password



### 1.3 Service Status Bar

When Mech-Center is running, the Service Status Bar will display the icon of the software, camera, robot, and interface service which are activated. You can click on the icon to open corresponding window.



## 1.4 Project Status Bar

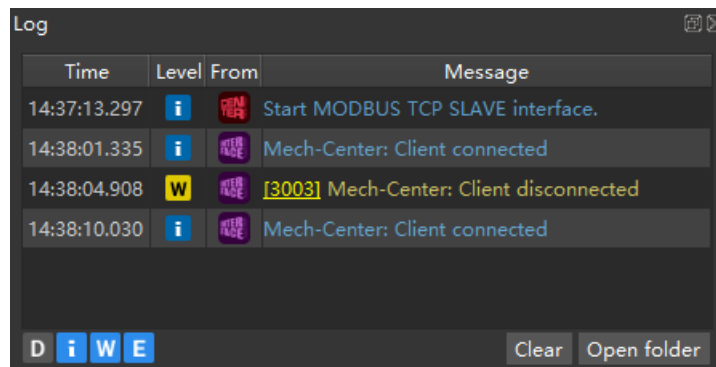
The Project Status Bar displays the project name, status, execution time, and details of Mech-Viz/Mech-Vision projects. You can learn the detailed project running status from the messages in it and the Log Panel on the right.

Project Status				
Project Name	Status	Exec Time	Details	
 test1	IDLE	0.726s	14:24:08 Execution Finished	
 test2	IDLE	1.029s	15:36:08 Execution Finished	

Options	Description
Project Name	The name of Mech-Viz / Mech-Vision projects.
Status	Current statuses of the projects (IDLE or RUNNING).
Execution Time	The amount of time taken by a project to complete its execution.
Details	Record of the execution time and the corresponding status. An error message will be displayed if an error occurs.

## 1.5 Log Panel

Display the log information of the current project and service in real time.




Options	Description
D, i, W, E	The level of logs. D stands for “debugging information”; i stands for “ordinary information”; W stands for “warning”; and E stands for “error”. Click on the tab in the lower left corner to hide or display the log information you need. Tabs in gray indicates that the log information of these levels is hidden.
Clear	Delete all logs.
Open folder	Open the folder where the current log is saved.

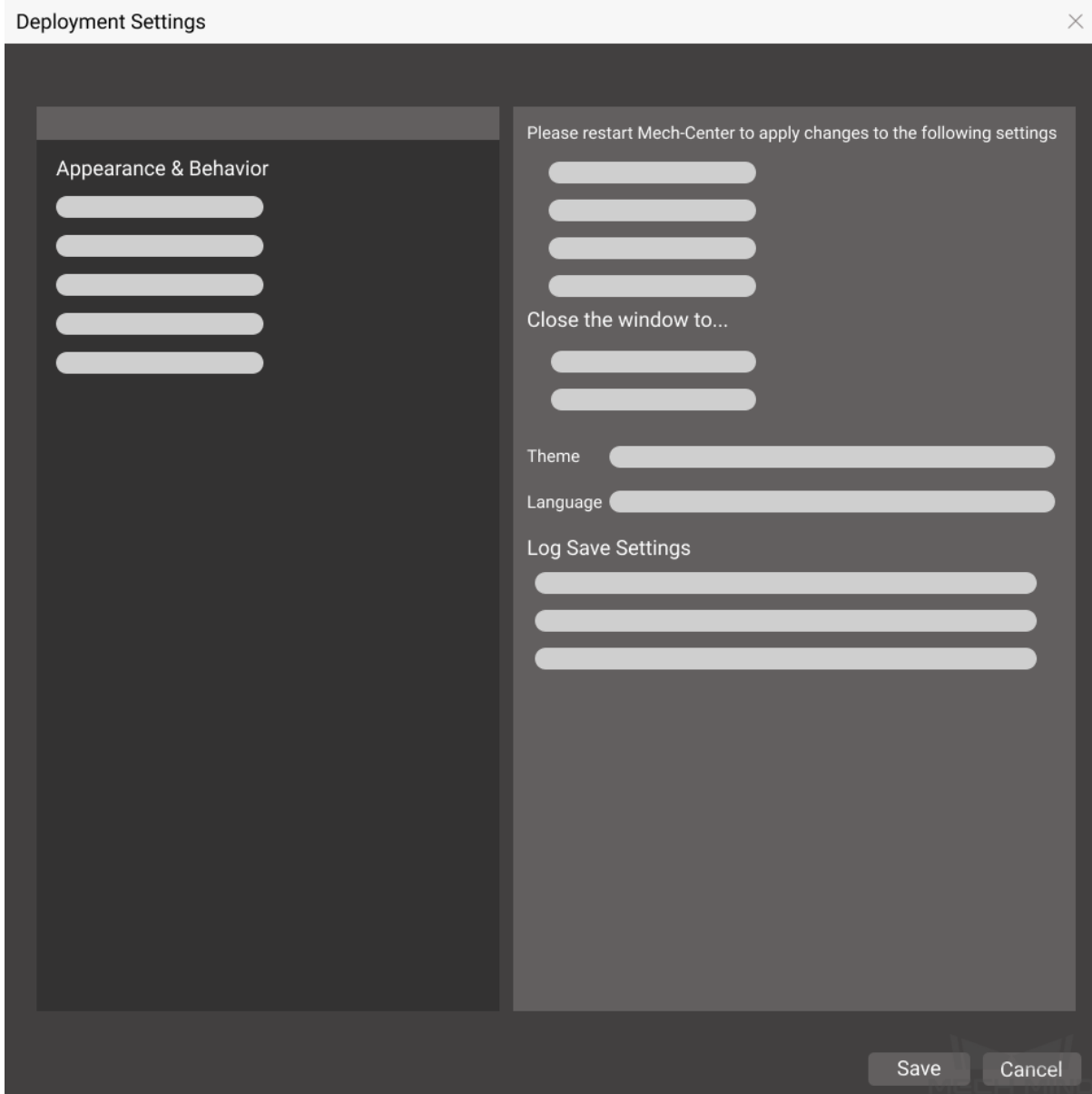
**Tip:** You can be redirected to the online user manual to view the details by clicking the error codes at the beginning of the error messages.



## GETTING STARTED WITH MECH-CENTER

### 2.1 Deployment Settings


Open Mech-Center and click on **Deployment Settings**  on the *Toolbar* to open the Deployment Settings window, as shown below. You can configure the path, parameter, and other settings in it. After configuration, please click on Save. For detailed instructions, please refer to *Deployment Settings*.

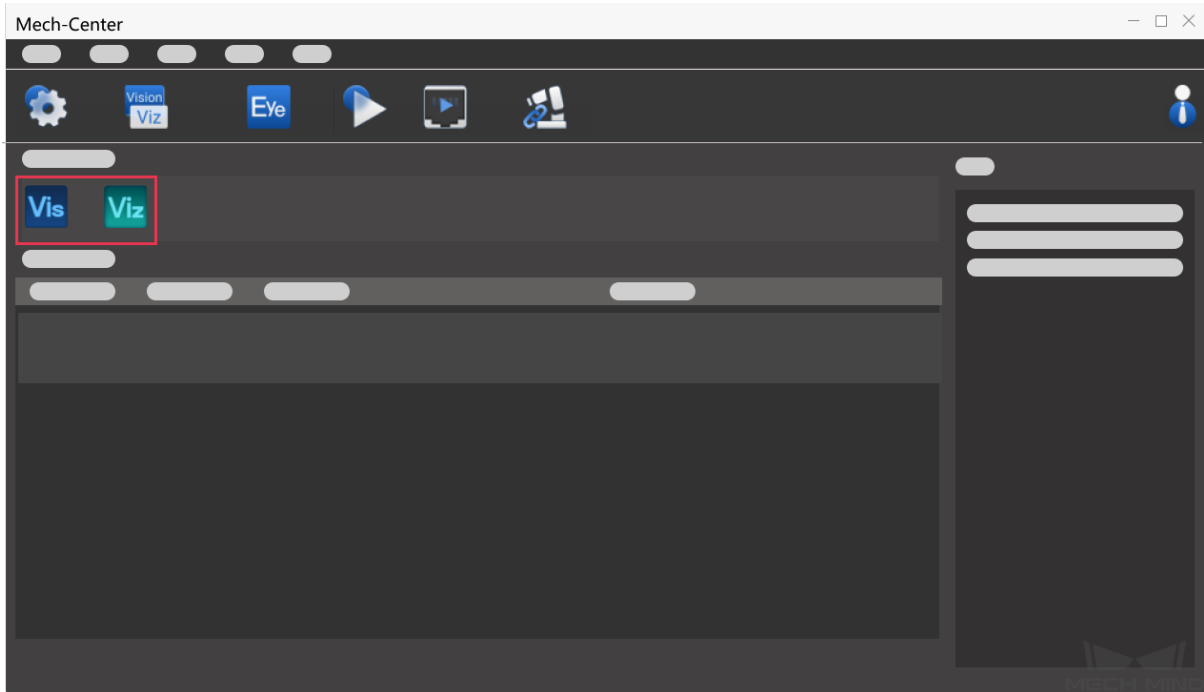


**Hint:**


- Please use Deployment Settings according to actual needs.
- If Mech-Vision, Mech-Viz, and Mech-Eye Viewer are installed successfully, their paths will be automatically added in Deployment Settings, and you do not need to add by yourself.

## 2.2 Open Projects

Click on **Start**  on the *Toolbar* to open Mech-Viz and Mech-Vision. Their icons will be displayed on the *Service Status Bar*.



### 2.2.1 Open Mech-Viz Project

1. Click on  to enter the main interface of Mech-Viz.
2. Create a project: please refer to `getting_started_viz` for detailed instructions. You can also open an existing project.
3. Check *Autoload Current Project* on the toolbar of Mech-Viz.
4. View project status: the project status will be displayed on the *Project Status Bar*.



---

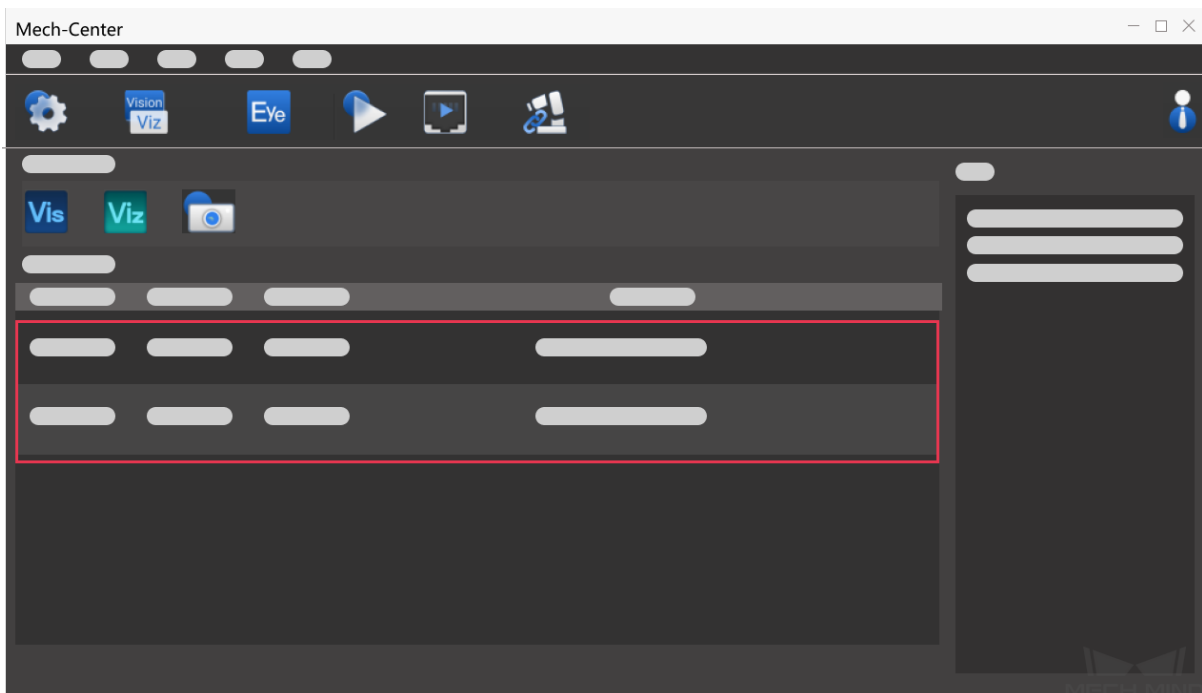
#### Hint:

- After checking *Autoload Current Project* in Mech-Viz, the **Project path** will be automatically added.
-


## 2.2.2 Open Mech-Vision Project




1. Click on  to enter the main interface of Mech-Vision.
2. Create a project: please refer to typical\_applications to create a project. You can also open an existing project.
3. Check *Autoload Project*: select the project in the **Projects List** and right-click with the mouse, and then check *Autoload Project* in the context menu.
4. Add project path: go to *Deployment Settings* → *Mech-Vision*, click on  and then *Save* to add the project path.
5. View project status: the project status will be displayed on the *Project Status Bar*, as shown below.



## 2.2.3 Configure Camera Settings

If you need to check or configure settings of the camera, click on **Start Mech-Eye Viewer**  on the *Toolbar*. The icon will be displayed on the *Service Status Bar*.


## 2.3 Connect the Robot

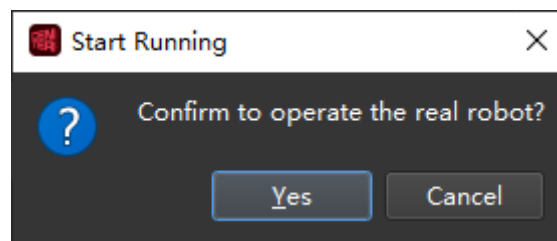
If the on-site robot you are using is not UR, you need to load the program files before connecting to Mech-Center. For detailed instructions, please see [robot\\_integrations](#). Click on **Connect Robot**  on the *Toolbar* to connect a real robot. If the connection is successful, a robot icon will be displayed on *Service Status Bar*.

For projects using Mech-Interface, please click on **Start Interface**  on the *Toolbar*. After the interface service is enabled, an icon will be displayed on *Service Status Bar*.

## 2.4 Run the Project

**Hint:** If you want to control a real robot, please go to *Deployment Settings* → *Mech-Viz* and uncheck *Run as simulation*.

Click on **Run** . If a window as shown below pops up, click on **Yes** to run the loaded project. Then you can view related information on the *Project Status Bar* while running the project.



### Attention:

- Please wait for the project to complete loading before running the project.  
If you do not select any options in the **Version compatibility** pop-up window when you open Mech-Viz, and click *Run* in Meh-Center directly, a message saying that **“Project loading by Mech-Viz in progress. Failed to start the Mech-Viz project: XXX”** will appear in the log panel in Mech-Center. In this case, please select **Yes** in the **Version compatibility** window and then click *Run* in Mech-Center to run the project again.
- When the real robot is working, please ensure the safety of personnel. When an emergency occurs, please press the emergency stop button on the robot teach pendant.

## 2.5 Example Mech-Viz Projects for Standard Interface

In the file location of Mech-Center, in the folder `tool\viz_project`, there are four example Mech-Viz projects for using Standard Interface.

### Check\_collision

For path planning and collision detection in vision-guided picking.

### Outer\_move

For the scenarios in which the robot needs to move to a pose passed in from an external client.

### Suction\_zone

For using multiple suction cup sections (or array grippers) through DO signals.

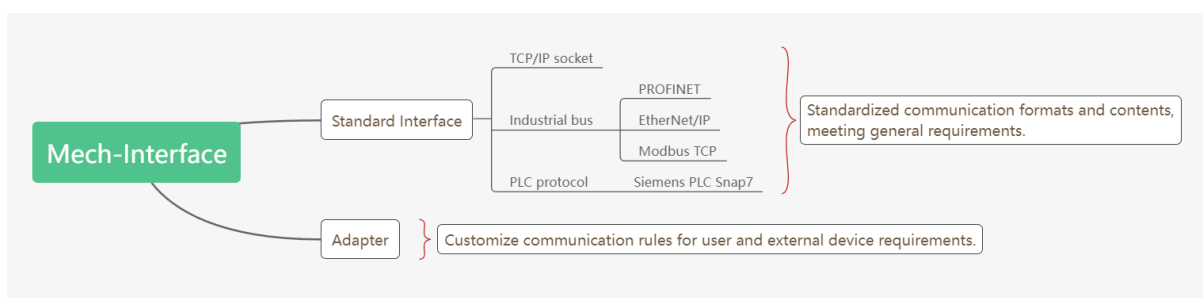
### Vision\_result\_reuse

For the scenarios in which the vision result returned at a time need to be used multiple times for path planning and collision detection in vision-guided picking.

## MECH-INTERFACE

Mech-Interface provides communication service with external devices for the Mech-Mind Software Suite. It receives data from external devices and sends out data from the system.

Mech-Interface consists of two parts: **Standard Interface** and **Adapter**.



---

This chapter contains the following sections.

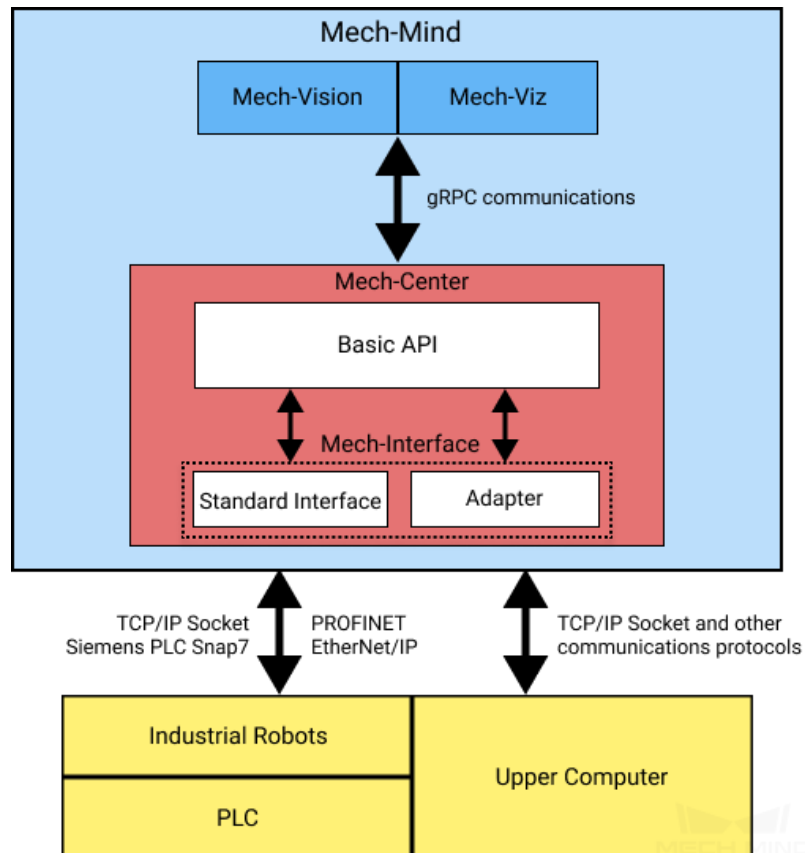
- *Mech-Interface Overview*: the mechanism of **Standard Interface** and **Adapter** and their corresponding application scenarios
- *Standard Interface*: configuration and deployment of **Standard Interface**
- *Standard Interface Development Manual*: communication protocols, commands, error codes, and troubleshooting of **Standard Interface**
- *Adapter*: quick facts, Adapter generator, and programming guide of **Adapter**

### 3.1 Mech-Interface Overview

This section covers the mechanism, differences between the two types of communication, and their application scenarios.

### 3.1.1 Communication Mechanism

The mechanism of the two types of communication are as shown in the figure below.



- **Adapter** is a Python program that is used to establish communication between external communication devices (e.g., industrial robots, upper computers, PLC) and Mech-Vision and Mech-Viz. It cannot only communicate with Mech-Vision and Mech-Viz, but also communicate with external devices over any communication protocol supported by Python.
- **Standard Interface** can be viewed as a complete set of Adapter programs provided by Mech-Mind. It supports a number of communication protocols and provides a powerful control command set and an error monitoring system. With these powerful functions, it can meet the requirements of most users.

### 3.1.2 Differences between Standard Interface and Adapter

Adapter and Standard Interface are both used to establish communication between external devices and Mech-Vision and Mech-Viz. Standard interface is a set of standardized Adapter programs, which does not support customized development.

For internal communication with Mech-Vision and Mech-Viz, both types of communication utilize Basic API, while specific communication protocols are used to communicate with external devices.

The differences between Standard Interface and Adapter are as shown in the table below.



Differences or Similarity	Standard Interface	Adapter
Internal Communication	Utilize Basic API to communicate with Mech-Vision and Mech-Viz	
External Communication	Support the following communication protocols only: TCP/IP Socket Siemens PLC Snap7 PROFINET EtherNet/IP	Support any type of communication protocol supported by Python
Function	Provide vision results only	Provide functions related to vision and other functions supported by Python, such as interface customization, database creation, and order management
Deployment Difficulty	Easy to use, and can be deployed rapidly	Require programming with high time and labor costs
Extensibility	Do not support extension	Can be extended to support more communication protocols and functions

### 3.1.3 Application Scenarios

In real application scenarios, the selection of a specific type of Mech-Interface is based on external communication devices, the communication protocol in use, the required communication functions, etc.

The common communication devices and protocols supported by Standard Interface and Adapter are as shown in the table below.

External Device	Protocol	Type of Mech-Interface	Description
Robot	TCP/IP Socket	Standard Interface	Mech-Interface acts as the server
	PROFINET		Mech-Interface acts as the slave device
	EtherNet/IP		Mech-Interface acts as the adapter (slave device)
	Modbus TCP	Adapter	Currently supported by Adapter only
Upper Computer	HTTP	Adapter	Applicable to integrated project where the robots are full-controlled by Mech-Mind
	WebSocket		
	TCP/IP Socket	Standard Interface	Mech-Interface acts as the server
PLC	TCP/IP Socket	Standard Interface	Mech-Interface acts as the server
	Siemens PLC Snap7		Mech-Interface acts as the client
	PROFINET		Mech-Interface acts as the slave device
	EtherNet/IP		Mech-Interface acts as the adapter (slave device)
	Modbus TCP	Adapter	Currently supported by Adapter only
	Mitsubishi PLC MC	Adapter	Currently supported by Adapter only

**Hint:**

- When Mech-Interface is used to communicate with external devices, it is recommended to use Standard Interface if the project requirements can be met; when Standard Interface cannot meet all the requirement (for example, the communication protocol used by external devices is not supported, or other functions that are not supported by Standard Interface is required), Adapter should be used.
- Please refer to *Standard Interface Development Manual* for functions of Standard Interface.
- Please refer to *Functions* for functions of Adapter.

Please refer to the following sections for more detailed information about Standard Interface and Adapter.

- *Standard Interface*
- *Adapter*

## 3.2 Standard Interface

When only the targets are needed and the Mech-Mind system is not needed to control the robot's movement, the standard interface can be used to transmit the data.

The standard interface only provides the most basic interface functions, such as sending vision points and task data. If more interface functions are required, please see *Adapter* for details.

The standard interface is integrated in the Mech-Center software and no additional generation is required. The service can be started directly in the software.

### 3.2.1 Instructions

#### Start Mech-Interface:

Under *Deployment Settings* → *Mech-Interface*, check *Enable Mech-Interface*, and check the interface service type as *Standard Interface*, as shown in *Figure 1*.

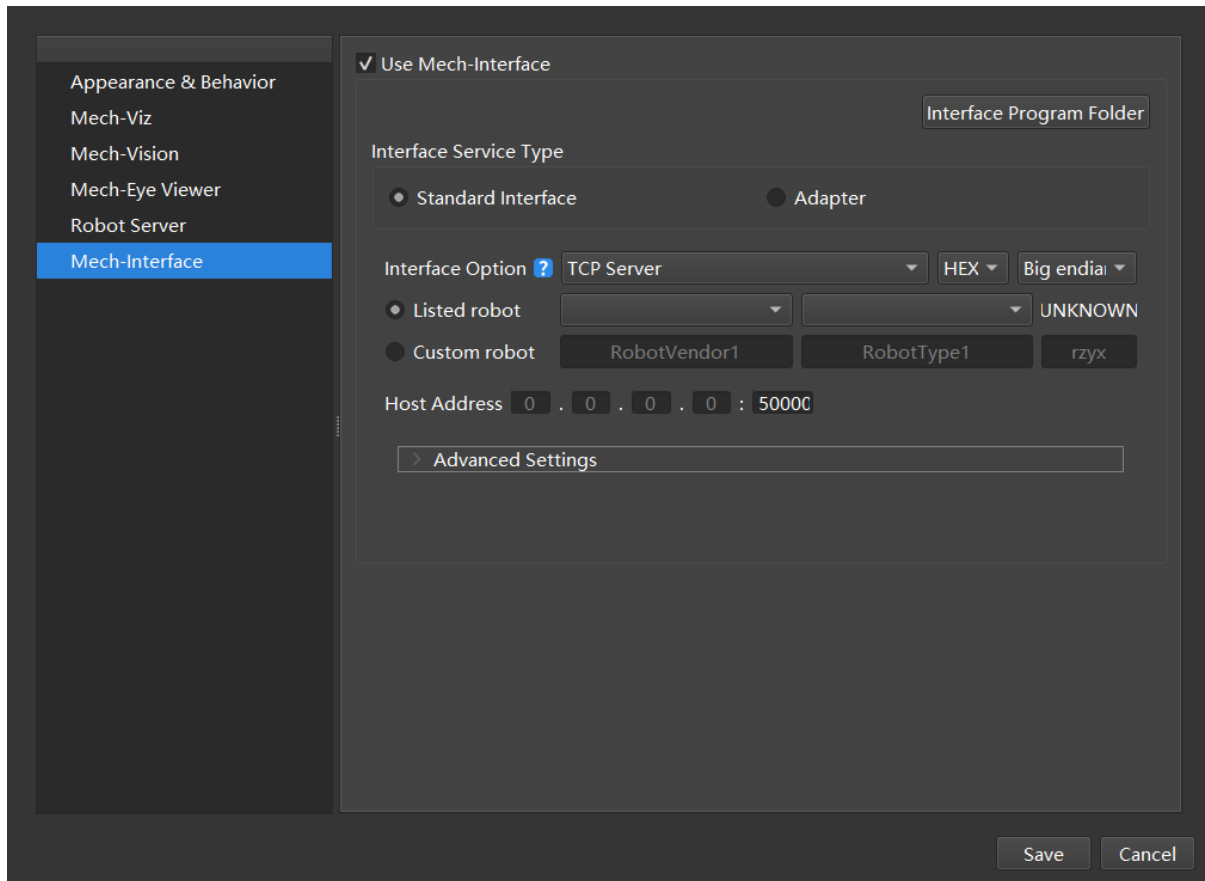


Figure 1. Start Mech-Interface

#### Interface Options, Host Address:

There are two types of external services: TCP Server and Siemens PLC Client.

Siemens PLC Client:

For Siemens PLC Client, the PLC IP and the DB Block Number need to be set. The default DB Block Number is 10.

TCP Server:

For TCP Server, the protocol format needs to be selected as ASCII or HEX. In HEX, please select Big Endian / Little Endian, i.e., ">" (big-endian) or "<" (little-endian). For TCP Server, the port number needs to be set as appropriate, and the default port number is 50000.

Please select the valid communication format to write the interface program based on the setup of the robot.

Robot type	Standard Interface example programs	
Industrial	ABB	HEX
	FANUC	HEX
	KUKA	HEX
	YASKAWA	ASCII
	KAWASAKI	ASCII
Collaborative	FANUC CRX	Plugin (HEX)
	UR	URCap (ASCII)
	TM	Plug and play (ASCII)
	Other	No example program has been provided yet. Please write the interface program based on the robot's support for HEX and ASCII.

---

**Tip:** The robot interface example program package and operating documents are located in Robot\_Interface under the software installation directory.

---

**PROFINET** PROFINET requires setting the robot model.

**EtherNet/IP** EtherNet/IP requires setting the robot model.

**Modbus TCP Slave** Modbus TCP Slave requires setting the slave IP, port number, device address, and byte order (you can do tests to determine the byte order).

#### Select a robot:

Click  to select the corresponding robot brand and model.

After the setting is complete, save and restart Mech-Center. After restarting, click *Start Interface* on the interface to enable the standard interface.

## 3.3 Standard Interface Development Manual

### 3.3.1 Overview

- *Protocols*
- *Introduction to Commands*
- *Mech-Center Service Settings*
- *TCP/IP Socket Communication*
- *Example Programs*

## Protocols

### TCP Server

Mech-Center provides a TCP Server (default port 50000) as an external service interface that supports the transfer of ASCII and HEX data.

### Siemens PLC Client

To communicate with Siemens S7 series PLC, Mech-Center provides a PLC Client (default DB address: 10) based on Snap7 protocol as a communication interface.

## PROFINET

For communication using the PROFINET industrial bus, the conditions that need to be met include:

- The industrial computer or host supports the installation of standard PCI-e cards.
- HMS INpact 40 PIR card and Ixxat VCI driver software have been installed.
- Mech-Center 1.5 or above has been installed.
- Use the PROFINET General Station Description (GSD) file provided by Mech-Center.
- PROFINET communication is in the standard big-endian data format. The data contains 32-bit DINT pose data, and the PROFINET master station (especially the robot controller) needs to support 32-bit integer sending and receiving.

## EtherNet/IP

Mech-Center can be used as an EtherNet/IP slave station to connect to the EtherNet/IP industrial network.

To communicate using the EtherNet/IP industrial bus, the conditions include:

- The industrial computer or host supports the installation of standard PCI-e cards.
- HMS INpact 40 PIR card and Ixxat VCI driver software have been installed.
- Mech-Center 1.5 or above has been installed.
- Use the EtherNet/IP Station Description (GSD) file provided by Mech-Center.
- EtherNet/IP communication is in the standard big-endian data format. The data contain 32-bit DINT pose data, and the EtherNet/IP master station (especially the robot controller) needs to support 32-bit integer sending and receiving.

## Modbus TCP Slave

Mech-Center can be used as a slave device software, providing the standard interface option Modbus TCP Slave for data communication with the master device. This function requires installing Mech-Center 1.6.1 or above.

## Introduction to Commands

### Mech-Vision

- 101: **Start Mech-Vision Project** For scenarios using only Mech-Vision but not Mech-Viz. This command is for starting the running of the corresponding Mech-Vision project for image acquisition and vision data processing.
- 102: **Get Vision Target(s)** For scenarios using only Mech-Vision but not Mech-Viz. This command is for reading the vision recognition results, i.e., target object pick points.
- 103: **Switch Mech-Vision Recipe** This command switches between the saved parameter recipes in Mech-Vision. Multiple parameter recipes are for recognizing different target objects. Parameters involved include recognition models, deep learning model files, etc.
- 110: **Get Custom Output Data from Mech-Vision** This command is for receiving data from Step "Procedure Out" in Mech-Vision when the data types to output are customized (the parameter "Port Type" is set to "Dynamic"). Executing this command once only fetches one pose and its corresponding labels, scores, etc. (if any) from the vision result. If expecting to receive multiple poses, please execute this command multiple times.

### Mech-Viz

- 201: **Start Mech-Viz Project** For scenarios using both Mech-Vision and Mech-Viz. This command starts the Mech-Viz project with the corresponding Mech-Vision project to plan the robot motion path.
- 202: **Stop Mech-Viz Project** This command is for manually terminating the running of Mech-Viz.
- 203: **Select Mech-Viz Branch** This command is for controlling the `branch_by_msg` Task (if there is one) in the Mech-Viz project to let the project run along the specified exit port.
- 204: **Set Move Index** Set the index parameter of the move-type Tasks in the Mech-Viz project. The move-type Tasks that contain index parameters include `move_list`, `move_grid`.
- 205: **Get Planned Path** This command obtains the robot path planned by the Mech-Viz project.
- 206: **Get DO Signal List** This command gets the DO list, i.e., the list of array gripper (or suction cup section) control signals calculated by Mech-Viz, when using multiple suction cup sections for picking multiple objects at a time.
- 207: **Get Mech-Viz Task Parameter** This command reads the values of specified parameters of specified Tasks. Please specify which parameters of which Tasks to read in *Property Config* under *Deployment Settings* -> *Mech-Interface* -> *Advanced Settings*.
- 208: **Set Mech-Viz Task Parameter** This command sets the values of specified parameters of specified Tasks. Please specify what values and which parameters of which Tasks to set in *Property Config* under *Deployment Settings* -> *Mech-Interface* -> *Advanced Settings*.

**210: Get Move Target with Vision Planning Result** This command is for getting a single planned target from Mech-Viz. The target may be a target for a vision\_move Task, or a target for one of other move-type Tasks. A target may contain pose, velocity, gripper info, object info, etc.

### Others

**501: Input Object Dimensions to Mech-Vision** Input object dimensions, i.e., length, width, and height of boxes, to set the 3D object dimensions in the Mech-Vision project through Step read\_object\_dimensions.

**502: Input TCP to Mech-Viz** Set a dynamically changing target in the Mech-Viz project through the outer\_move Task in the project.

**601: Notify** This command does not need to be initiated by the user. It will be executed when the Mech-Vision/Mech-Viz project raises a notification message by a **Notify** Step/Task.

**701: Calibration** For hand-eye calibration for cameras. This command obtains the calibration points and triggers the camera to take pictures, thus completing the calibration. The calibration points are from Mech-Vision.

**901: Get Software Status** Get the status of Mech-Mind Software Suite for checking the project status.

---

**Note:** Unified data units are required for communication:

- The unit of joint angle and Euler angle is degree (°).
  - The unit of XYZ coordinates in the flange pose (pose) is mm.
- 

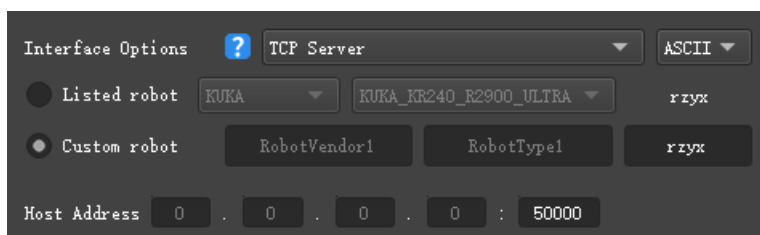
## Mech-Center Service Settings

### TCP Server

The interface service is disabled by default. To enable the interface service, please click on *Deployment Settings* → *Mech-Interface* → *Use Mech-Interface* → *Standard Interface*.

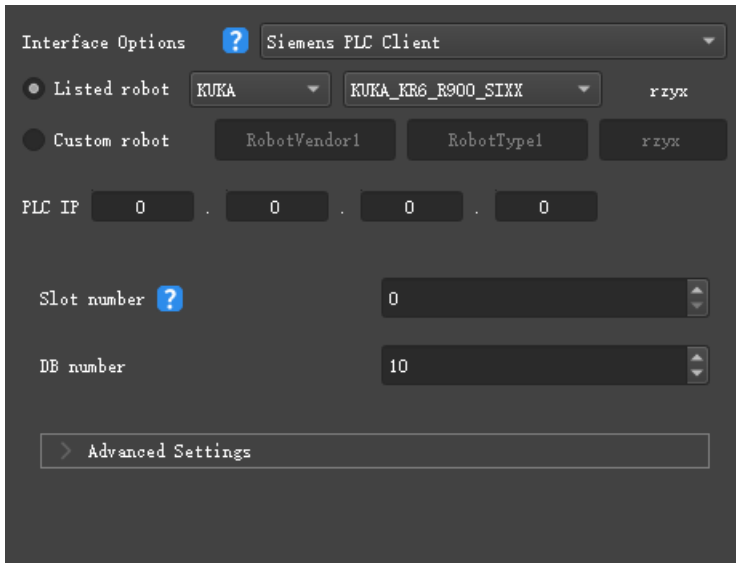
For using external services on the TCP server, please set the IP port according to your actual needs, the default port is 50000.

For TCP Server, please select the data type as ASCII or HEX. For HEX, please select between big-endian and little-endian.



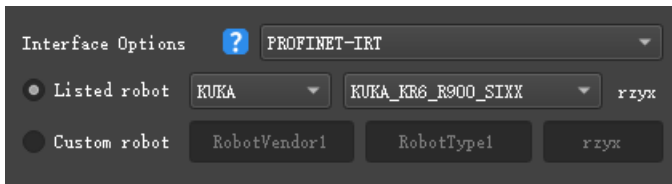
### Siemens PLC Client

For Siemens PLC Client, please set the PLC IP, slot number, and DB number. The default slot number is 0, and the default DB number is 10.



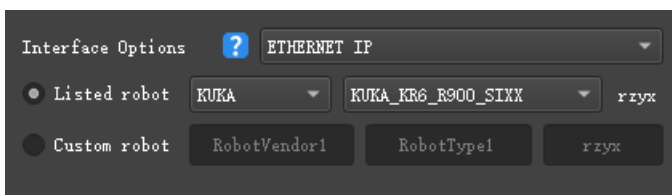
### PROFINET

For PROFINET, the default board IP is 0.0.0.0, and the configuration can be made in the Siemens PLC programming software or HMS IPconfig.



### EtherNet/IP

For EtherNet/IP, the default board IP is 0.0.0.0, and the configuration can be made in HMS IPconfig.





## TCP/IP Socket Communication

### Communication Settings

- Set the IP addresses of the robot and IPC to be in the same subnet. Command line `ping xxx.xxx.xxx.xxx` (the x's are the fields for the IP address) can be used to test whether the network is connected.
- Open the **Deployment Settings** window in Mech-Center, select **Mech-Interface** in the left panel, and check **Enable Mech-Interface**.
- Select **Standard Interface** for **Interface Service Type**.
- For **Interface Options**, select **TCP Server**, and select **ASCII** or **HEX** according to what format the client program supports.
- Select the model of the robot that needs communication. The supported robot models are listed in the options. If the robot model used is not listed, please select **Custom robot** and set the Euler angle format of the robot.
- Set the port for TCP. Default: 50000.
- Advanced settings:
  - Set the max number of poses that can be transmitted each time. Default: 20. The reason is that the length of data that can be transmitted each time is limited.
  - Set the timeout period of receiving data from Mech-Viz. The default timeout period is 10 seconds. Mech-Viz needs some time for computing before outputting the data.
  - Set the timeout period of receiving data from Mech-Vision. The default timeout period is 10 seconds. Mech-Vision needs some time for computing before outputting the data.
- Click on **Save** to save the settings.

### Example Programs

For instructions on using the example programs of the Standard Interface, please see `standard_interface_robot_integrations`.

### 3.3.2 TCP/IP

Mech-Mind Software Suite can communicate with the following robots through TCP/IP:

- ABB
- YASKAWA
- FANUC
- KUKA
- Kawasaki

For setup instructions and other information specific to each robot, please refer to `Robot Integrations - Standard Interface`.

The commands are as follows:

- *Command 101: Start Mech-Vision Project*
- *Command 102: Get Vision Target(s)*
- *Command 103: Switch Mech-Vision Recipe*
- *Command 110: Get Custom Output Data from Mech-Vision*
- *Command 201: Start Mech-Viz Project*
- *Command 202: Stop Mech-Viz Project*
- *Command 203: Select Mech-Viz Branch*
- *Command 204: Set Move Index*
- *Command 205: Get Planned Path*
- *Command 206: Get DO List*
- *Command 207: Read Mech-Viz Task Parameter*
- *Command 208: Set Mech-Viz Task Parameter*
- *Command 210: Get Move Target with Vision Planning Result*
- *Command 501: Input Object Dimensions to Mech-Vision*
- *Command 502: Input TCP to Mech-Viz*
- *Command 601: Notify*
- *Command 701: Calibration*
- *Command 901: Get Software Status*

### **Command 101: Start Mech-Vision Project**

This command starts the running of the Mech-Vision project, which executes image capturing and performs vision recognition.

If the project works in the eye-in-hand mode, the robot pose for image capturing needs to be transmitted by this command into the project.

This command is for scenarios using only Mech-Vision.

#### **Command Sent**

101, project number, expected number of vision points, robot pose type, robot pose

#### **Project number**

The integer ID number of the Mech-Vision project in Mech-Center, i.e., the number shown on the left of the project path in *Deployment Settings* → *Mech-Vision* in Mech-Center. The number can be adjusted by dragging the projects.

#### **Expected number of vision points**

The number of vision points (i.e., vision poses and their corresponding point clouds, labels, indices, etc.) to expect Mech-Vision to output.

- 0: Get all the vision points from the Mech-Vision project's recognition results.
- `integers > 0`: Get the specified number of vision points.
  - If the total number of vision points is smaller than the parameter value, all the available vision points will be returned.
  - If the total number of vision points is greater than or equal to the parameter value, vision points in the quantity of the parameter value will be returned.

---

**Note:** The command to obtain the vision points is command 102. In TCP/IP, due to the limit that a maximum of 20 vision points can be obtained by executing command 102 at a time, after executing command 102 for the first time, one of the parameters returned will indicate whether all the vision points requested have been returned; if not, please repeat executing command 102.

---

### Robot pose type

This parameter indicates the type of the image-capturing pose of the real robot to input to Mech-Vision.

- 0: No robot pose needs to be transmitted by this command. If the project works in the eye-to-hand mode, then image capturing has nothing to do with the robot's pose, so no robot image-capturing pose is needed by Mech-Vision.
- 1: The robot pose transmitted by this command is JPs.
- 2: The robot pose transmitted by this command is a flange pose.

### Robot pose

This parameter is the robot pose needed when the project works in the eye-in-hand mode.

The robot pose is either JPs or flange pose, according to the setting of the parameter **robot pose type**.

### Data Returned

101, `status code`

#### Status code

If there is no error, status code 1102 will be returned. Otherwise, the corresponding error code will be returned.

## Test Samples

Command 101 is executed without error.

```
TCP send string = 101, 1, 10, 1, 0, -20.63239, -107.81205, 0, -92.81818, 0.00307
TCP received string = 101, 1102
```

Error: project ID number does not exist.

```
TCP send string = 101, 2, 10, 1, 0, -20.63239, -107.81205, 0, -92.81818, 0.00307
TCP received string = 101, 1011, 1
```

## Command 102: Get Vision Target(s)

This command gets the vision targets, i.e., robot TCPs to reach the object poses contained in the vision points, after executing command 101.

Mech-Center automatically transforms the object poses in the vision points to their corresponding robot TCPs. The process is as follows:

- Rotate the poses around their Y axes by 180 degrees.
- Determine whether the definition of the reference frame used by the robot model involves robot base height, and add a vertical offset accordingly.

---

**Note:** In TCP/IP, by default, command 102 can only fetch at most 20 targets at a time. Therefore, command 102 may need to be repeatedly executed until all the targets required are obtained.

---

## Command Sent

102, project number

### Project number

The integer ID number of the Mech-Vision project in Mech-Center, i.e., the number shown on the left of the project path in *Deployment Settings* → *Mech-Vision* in Mech-Center. The number can be adjusted by dragging the projects.

## Data Returned

102, status code, transmission completion status, number of targets, reserved field, target, target, ...

---

**Note:** The targets (up to 20 targets by default) are located at the tail of the data returned.

---

### Status code

If there is no error, status code 1100 will be returned. Otherwise, the corresponding error code will be returned.

After executing this command, if the results from Mech-Vision have not been returned, Mech-Center will wait before sending the results to the robot. The default wait time is 10 seconds. If a timeout occurs, the timeout error status code will be returned.

### Transmission completion status

This parameter indicates whether all the targets requested have been obtained.

- 0: Not all the targets requested have been obtained. Please repeat executing command 102 until this parameter turns 1.
- 1: All the targets requested have been obtained.

---

**Note:** If not all the targets requested have been obtained and command 101 is executed at this time, the rest of the targets that are not obtained will be cleared.

---

### Number of targets

The number of targets obtained by executing this command this time.

### Reserved field

This field is not used.

The value defaults to 0.

### Target

`pose, label, velocity`

- **Pose:** For this command, the robot pose is in the form of TCP, not JPs. A TCP includes the Cartesian coordinates (XYZ) and Euler angles (ABC).
- **Label:** The integer label assigned to the pose. If in the Mech-Vision project, the labels are strings, they need to be mapped to integers before outputting from the Mech-Vision project using Step label\_mapping. If there are no labels in the Mech-Vision project, the label defaults to 0.
- **Velocity:** The parameter defaults to 0 for command 102 because Mech-Vision does not plan robot target velocity.

### Test Samples

#### Test samples of normal execution and execution error

Command 102 is executed without error.

```
TCP send string = 102, 1
TCP received string = 102, 1100, 1, 1, 0, 95.7806085592122, 644.5677779910724, 401.
↔1013614123109, 91.12068316085427, -171.13014981284968, 180.0, 0, 0
```

Error: no vision result.

```
TCP send string = 102, 1
TCP received string = 102, 1002, 1
```

### Test Sample of requesting targets

The test sample below is obtaining 22 targets by sending commands 101, 102, and 102 sequentially. Details are as follows:

- TCP/IP sends command 101, with content 101, 1, 0, 1, ..., expecting to obtain all the available targets.
- TCP/IP sends command 102 to obtain part of the targets.
- TCP/IP receives the data returned by executing command 102. The content is 102, 1100, 0, 20, ..., indicating 20 targets have been obtained and not all targets have been obtained.
- TCP/IP sends command 102 again to fetch the remaining targets.
- TCP/IP receives the data returned by executing command 102 again. The content is 102, 1100, 1, 2, ..., which includes 2 targets and indicates all the targets have been obtained.

```

TCP send string = 101, 1, 0, 1, -0, -20.63239, -107.81205, -0, -92.81818, 0.0016
TCP received string = 101, 1102
TCP send string = 102, 1

TCP received string = 102, 1100, 0, 20, 0, 95.7806085592122, 644.5677779910724, 401.
↪1013614123108, 31.12068316085427, ...
TCP received string = 78549940546, -179.99999999999991.0.0, 329.228345202334.712.7061697180302.
↪400.9702665047771, ...
TCP received string =39546, -83.62567351596952, -170.87955974536686, -179.99999999999937, 0, 0,
↪ 223.37118373658322, ...
TCP received string = 005627, 710.1004355953408, 400.82227273918835, -43.89328326393665, -171.
↪30845207792612, ...
TCP received string = 20.86318821742358, 838.7634193547805, 400.79807564314797, -102.
↪03947940869523, -171.149261231 ...
TCP received string = 390299920645, -179.99999999999994, 0, 0, 303.0722145720921, 785.
↪3254917220695, 400.75827437080, ...
TCP received string = 99668287.77.78291612041707, -171.53941633937786, 179.99999998999997, 0.0,
↪ 171.47819668864432, ...
TCP received string = 332193785, 400.6472716208158, -94.3418019038759, -171.10001228964776, -
↪179.39999999999994, ...
TCP received string = 92388542936, 807.5641001485708, 400.6021999602664, - 167.9834797197932.-
↪171.39671274951826, ...
TCP received string = 278.3198007132188, 780.5325992145735, 400.4924381003066, -174.
↪72728396633053, -171.422604771 ...
TCP received string = 3.999999999999994, 0, 0, 183.82195326381233, 862.5171519967056.400.
↪422966515846.-154. 17801945 ...
TCP received string = 173.34301974982765, -180.0, 0, 0

TCP send string = 102, 1

TCP received string = 102, 1100, 1, 2, 0, 315.2017788478321, 592.1261793743445, 399.
↪60526335590957, 126.19602189220371, ...
TCP received string = 686127, -171.44430002882129, -1.3381805753922965e-15, 0, 0

```

### Command 103: Switch Mech-Vision Recipe

This command switches the parameter recipe used by the Mech-Vision project.

In Mech-Vision, you can change the settings of a group of parameters by switching the parameter recipe.

Parameters involved in recipe switching usually include point cloud matching model, image matching template, ROI, confidence threshold, etc.

This command needs to be used before executing command 101 which starts the Mech-Vision project.

#### Command Sent

103, project number, recipe number

##### Project number

The integer ID number of the Mech-Vision project in Mech-Center, i.e., the number shown on the left of the project path in *Deployment Settings* → *Mech-Vision* in Mech-Center. The number can be adjusted by dragging the projects.

##### Recipe number

The ID number of the parameter recipe to switch to, i.e., the number on the left of the parameter recipe name in *Project Assistance* → *Parameter Recipe* → *Parameter Recipe Editor* in Mech-Vision.

Recipe number range: 1—99.

#### Data Returned

103, status code

##### Status code

If there is no error, status code 1107 will be returned. Otherwise, the corresponding error code will be returned.

#### Test Samples

Command 103 is executed normally without errors.

```
TCP send string = 103, 1, 2
TCP received string = 103, 1107
```

Error: invalid recipe number.

```
TCP send string = 103, 1, 2
TCP received string = 103, 1102
```

### Command 110: Get Custom Output Data from Mech-Vision

This command is for receiving custom data, i.e., data other than poses and labels, from Step “Procedure Out” in Mech-Vision (when the parameter “Port Type” is set to “Dynamic”).

Executing this command once only fetches one pose and its corresponding labels, scores, etc. (if any) from the vision result. If expecting to receive multiple poses, please execute this command multiple times.

#### Command Sent

110, project number

##### Project number

The integer ID number of the Mech-Vision project in Mech-Center, i.e., the number shown on the left of the project path in *Deployment Settings* → *Mech-Vision* in Mech-Center. The number can be adjusted by dragging the projects.

#### Data Returned

110, status code, pose transmission completion status, number of custom data items, robot TCP transformed from object pose, label, custom data items, ..., custom data items

##### Status code

If there is no error, status code 1100 will be returned. Otherwise, the corresponding error code will be returned.

After executing this command, if the results from Mech-Vision have not been returned, Mech-Center will wait before sending the results to the robot. The default wait time is 10 seconds. If a timeout occurs, the timeout error status code will be returned.

##### pose transmission completion status

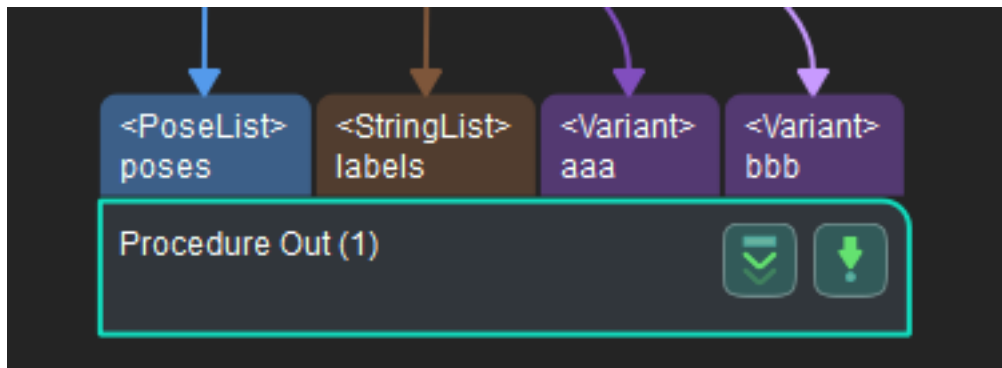
- 0: Not all poses in the vision result have been transmitted.
- 1: All poses in the vision result have been transmitted.

##### Number of custom data items

The total number of custom data items of data types other than poses and labels.

For example, if aaa is a list with 3 items and bbb is a list with 3 items, when executing this command to fetch one item from each list, the parameter value is 1+1=2; if aaa and bbb are both single variables, the parameter value is still 1+1=2.





### Pose

The TCP under the robot frame. Mech-Vision outputs the object pose, and the pose is transformed into robot TCP by Mech-Center. Step "Procedure Out" must have a port for poses.

A TCP for an object pose is usually generated by flipping the z-axis of the object pose.

### Label

Object info label corresponding to the pose. The label should be a positive integer. If not, it should be mapped to a positive integer in the Mech-Vision project by Step `label_mapping`. If there is no port for labels on Step "Procedure Out", the field is filled with 0.

### Custom data

The custom data for the corresponding port of custom data type on Step "Procedure Out".

The custom data parameters are arranged in alphabetical order A–Z by port name.

## Command 201: Start Mech-Viz Project

This command is for scenarios using both Mech-Vision and Mech-Viz.

This command starts the running of the Mech-Viz project, calls the corresponding Mech-Vision project, and lets the Mech-Viz project plan the robot path based on the vision points from Mech-Vision.

For the Mech-Viz project that needs to be started, the option *Autoload* needs to be checked in Mech-Viz's interface.

Please see [Example Mech-Viz Projects for Standard Interface](#) for the description of example Mech-Viz projects.

## Command Sent

201, pose type, robot pose

### Pose type

0:

- The current pose of the robot is not needed by Mech-Viz and no pose will be sent.
- If the project works in the eye-to-hand mode, no robot image-capturing pose will be needed by the project.

- In Mech-Viz, the simulated robot will move from the initial pose JPs = [0, 0, 0, 0, 0, 0] to the first target in the planned path.

1:

- The robot pose will be sent to Mech-Viz and the pose sent is in the form of JPs and flange pose.
- In Mech-Viz, the simulated robot will move from the input initial pose (i.e., the pose sent by this command) to the first target in the planned path.

---

**Note:** If in the scene, there are barriers that stand in the way from the initial pose JPs = [0, 0, 0, 0, 0, 0] to the first target in the planned path, the pose type must be set to 1.

---

### Robot pose

The current JPs and flange pose of the real robot (if pose type is set to 1).

### Data Returned

201, status code

#### Status code

If there is no error, status code 2103 will be returned. Otherwise, the corresponding error code will be returned.

### Test Samples

Command 201 is executed without error.

```
TCP send string = 201, 1, 0, -20.63239, -107.81205, 0, -92.81818, 0.00307
TCP received string = 201, 2103
```

Error: Mech-Viz does not support robot pose in TCP.

```
TCP send string = 201, 2, -0, 682.70355, 665.22266, 90, 179.99785, -89.99693
TCP received string = 201, 2015, 1
```

### Command 202: Stop Mech-Viz Project

This command stops the running of the Mech-Viz project. This command is needed only when the Mech-Viz project fall into an infinite loop or cannot be stopped normally.

## Command Sent

202

## Data Returned

202, status code

### Status code

If there is no error, status code 2104 will be returned. Otherwise, the corresponding error code will be returned.

## Test Sample

Command 202 is normally executed.

```
TCP send string = 202
TCP received string = 202, 2104
```

## Command 203: Select Mech-Viz Branch

This command specifies which branch the project should run along. For this command, the branching is implemented by a `branch_by_msg` Task, and this command selects the branch by specifying an exit port of the Task.

Before executing this command, the Mech-Viz project needs to be started by executing command 201.

When the Mech-Viz project runs to the `branch_by_msg` Task, it will wait for command 203 to specify which exit port of the Task, i.e., the branch, the project should run along.

## Command Sent

203, branching Task ID, exit port number

### Branching Task ID

The Task ID of the `branch_by_msg` Task.

This parameter is for specifying which `branch_by_msg` Task the branch selection should apply to.

The Task ID can be read in the Task's parameters.

### Exit port number

This parameter is for specifying which exit port of the specified Task, i.e., the branch, the project should run along. The value should be an integer ([1, N]).

---

**Note:** An exit port number is the 1-based index of the specified exit port on the Task. For example, if the specified exit port is the second exit port of the Task from left to right, the exit port number is 2.

---

### Data Returned

203, status code

#### Status code

If there is no error, status code 2105 will be returned. Otherwise, the corresponding error code will be returned.

### Test Samples

Command 203 is executed normally without errors.

```
TOP send string = 203, 1, 1
TCP received string = 203, 2105
```

Error: invalid exit port number.

```
TCP send string = 203, 1, 3
TCP received string = 203, 2018, 1
```

### Command 204: Set Move Index

This command is for setting the index parameter of a Task that involves sequential or separate motions or operations.

Tasks with index parameters include `move_list`, `move_grid`, `custom_pallet_pattern`, `smart_pallet_pattern`, etc.

Before executing this command, command 201 needs to be executed to start the Mech-Viz project.

### Command Sent

204, Task ID, index value

#### Task ID

This parameter specifies which Task the index setting should apply to.

The Task ID can be read in the Task's parameters.

#### Index value

The index value that should be set the next time this Task is executed.

When this command is sent, the current index value in Mech-Viz will become the parameter value minus 1.

When the Mech-Viz project runs to the task specified by this command, the current index value in Mech-Viz will be increased by 1 to become the parameter's value.

### Data Returned

204, status code

#### Status code

If there is no error, status code 2106 will be returned. Otherwise, the corresponding error code will be returned.

### Test Samples

Command 204 is executed normally without errors.

```
TCP send string = 204, 2, 6  
TCP received string = 204, 2106
```

Error: Failed to set the index.

```
TCP send string = 204, 3, 6  
TCP received string = 204, 2028, 1
```

### Command 205: Get Planned Path

This command gets the robot motion path planned by Mech-Viz after command 201 is executed to start the Mech-Viz project.

---

**Note:** In TCP/IP, by default, command 205 can only fetch at most 20 targets of the planned path at a time. So, command 205 may need to be executed repeatedly until all the targets required are obtained.

---

---

**Note:** If one of the targets in the path is not supposed to be sent to the robot, please clear the parameter "Send Target" checkbox of the corresponding move-type Task.

---

## Command Sent

205, target type

### Target type

This parameter specifies the type of path targets to return from Mech-Viz.

- 1: The targets returned should be in JPs.
- 2: The targets returned should be in TCP.

## Data Returned

205, status code, transmission completion status, number of points, position of "visual\_move", target, target, ...

### Status code

If there is no error, status code 2100 will be returned. Otherwise, the corresponding error code will be returned.

---

**Note:** When executing this command, if Mech-Viz has not yet had the planned robot motion path (the project is still running), Mech-Center will wait. The default wait time is 10 seconds. If a timeout occurs, a timeout error code will be returned.

---

### Transmission completion status

- 0: Not all the targets of the planned path have been obtained. Please repeat executing this command until this parameter's value is 1.
- 1: All the targets of the planned path have been obtained.

---

**Note:** If the expected number of targets to transmit is greater than 20 (20 is the default setting), please execute command 205 multiple times until the returned value of this parameter is 1.

---

### Number of points

This parameter indicates the number of path targets ([pose, label, velocity]) sent by executing this command this time.

Default range: 0 to 20.

### Position of "visual\_move"

The position of the vision\_move Task, i.e., the move to the vision pose (usually the pose for picking the object) in the entire robot motion path.

For example, if the path is composed of Tasks **move\_1**, **move\_2**, **visual\_move**, **move\_3** sequentially, the position of **visual\_move** is 3.

If in the path there is no vision\_move Task, the returned value will be 0.

### Target

[pose, label, velocity]

- **Pose:** Cartesian coordinates (XYZ) and Euler angles (ABC), or JPs, according to the pose type set by command 205.
- **Label:** Label is the integer label assigned to the pose. If in the Mech-Vision project, the labels are strings, they need to be mapped to integers before outputting from the Mech-Vision project using Step label\_mapping. If there are no labels in the Mech-Vision project, the label defaults to 0.
- **Velocity:** The non-zero velocity parameter percentage value for the move-type Task set in Mech-Viz.

### Test Samples

Command 205 is executed without error.

```
TCP send string = 205, 1

TCP received string =205, 2100, 1. 2, 2, 8.307755332057372, 15.163476541700463, -142.
↔1778810972881, -2.7756047848536745, -31.44046012182799, -96.94907235126934, 0, 64, 8.2↵
↔42574265592342, 12.130080796661591, -141.75872288706663-2.513533225987894, -34.8905853↵
↔039525, -97.19108378871277, 0, 32
```

Error: Mech-Vision runtime error.

```
TCP send string = 205, 1
TCP received string = 205, 2008, 1
```

### Command 206: Get DO List

This command gets the planned DO signal list when there are multiple grippers, such as suction cup sections, to control.

For using this command:

In the parameters of the “set\_do\_list” Task:

- Check “StandardInterface” under “Receiver”
- Check “Get DO List from VisualMove”
- Select a “visual\_move” Task that needs the DO signal list at the bottom of the parameter panel

Before calling this command, command 205 needs to be executed to obtain the planned motion path by Mech-Viz.

Please deploy the Mech-Viz project based on the template project at `/Mech-Center/tool/viz_project/suction_zone`, and set the suction cup configuration file in the Mech-Viz project.





### Command Sent

207, config ID

### Config File Content Format

read, config ID, Task ID, parameter key name

The lines starting with “#” are comment lines and take no effect when executing this command.

read

The string “read” indicates the command is for reading the parameter value.

config ID

The positive integer number for specifying the ID of the read operation. A config ID can only be used for reading one parameter. If reading multiple parameters, please use different config IDs.

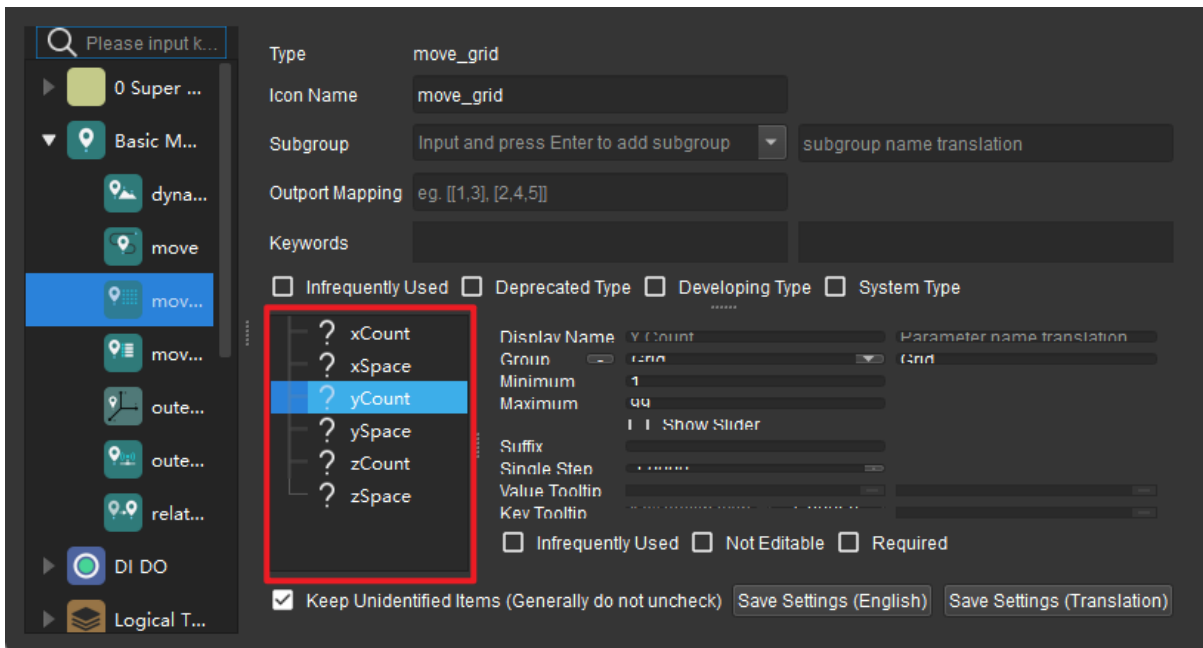
Task ID

The ID of the Task that the parameter to read belongs to.

The Task ID can be read in a Task’s parameters.

Parameter key name

The key name of the parameter to read. Please find the key name in *Settings -> Skill Setting Dialog* in Mech-Viz, as shown in the figure below.



## Config File Content Example

```
read, 1, 10, digitalOutValue
read, 2, 2, xCount
read, 3, 2, yCount
read, 4, 5, allowVisionResultUnused
write, 1, 3, xOffset, 0.000000
write, 1, 3, yOffset, 0.000000
write, 1, 3, zOffset, 0.000030
write, 2, 7, delayTime, 0.1
```

## Data Returned

207, status code, Task parameter value

### Status code

If there is no error, status code 2109 will be returned. Otherwise, the corresponding error code will be returned.

### Task parameter value

The value of the specified parameter of the specified Task in the Mech-Viz project.

Data types supported: INT, FLOAT, STRING.

## Command 208: Set Mech-Viz Task Parameter

This command sets the values of specified parameters of specified Tasks. Please specify what values and which parameters of which Tasks to set in *Property Config* under *Deployment Settings* -> *Mech-Interface* -> *Advanced Settings*.

## Command Sent

208, config ID

## Config File Content Format

```
write, config ID, Task ID, parameter key name, value
```

### write

The string "write" indicates the command is for writing the parameter value.

### config ID

The positive integer number for specifying the ID of the write operation.

A config ID can be used for reading multiple parameters.

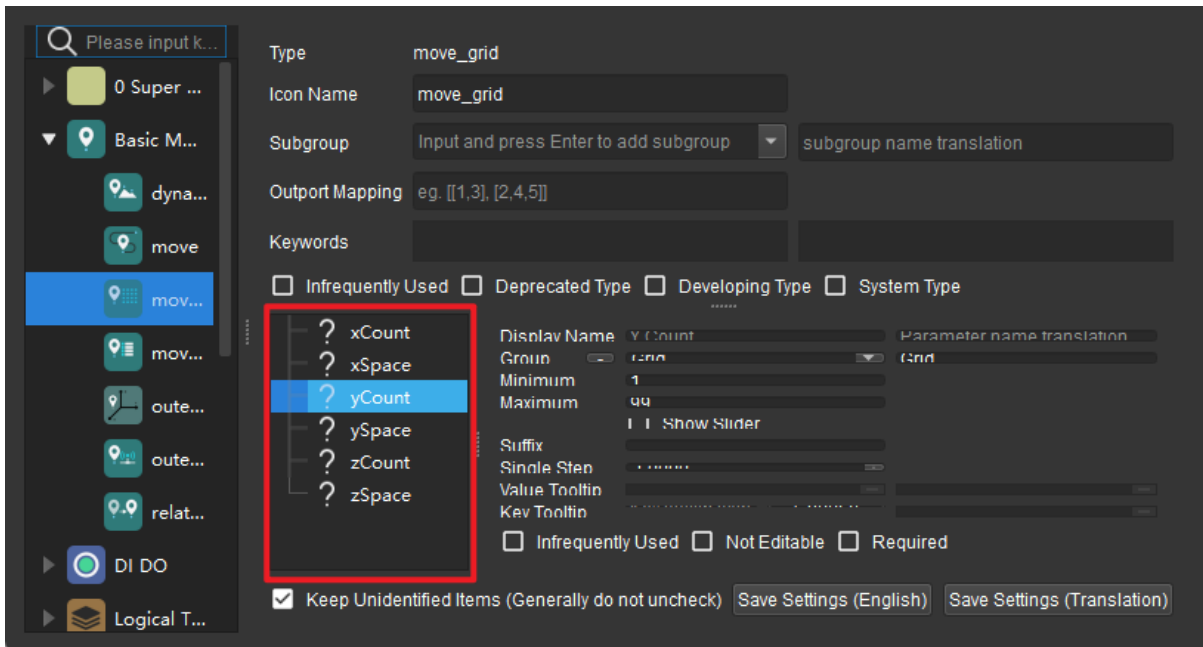
### Task ID

The ID of the Task that the parameter to write belongs to.

The Task ID can be read in a Task's parameters.

**Parameter key name**

The key name of the parameter to read. Please find the key name in *Settings -> Skill Setting Dialog* in Mech-Viz, as shown in the figure below.



**Value**

What value the specific parameter should be set to.

In addition to numbers, string values are also supported.

**Config File Content Example**

```
read, 1, 10, digitalOutValue
read, 2, 2, xCount
read, 3, 2, yCount
read, 4, 5, allowVisionResultUnused
write, 1, 3, xOffset, 0.000000
write, 1, 3, yOffset, 0.000000
write, 1, 3, zOffset, 0.000030
write, 2, 7, delayTime, 0.1
```

## Data Returned

208, status code

If there is no error, status code 2108 will be returned. Otherwise, the corresponding error code will be returned.

## Command 210: Get Move Target with Vision Planning Result

This command is for getting a single planned target from Mech-Viz. The target may be a target for a vision\_move Task, or a target for one of other move-type Tasks. A target may contain pose, velocity, gripper info, object info, etc.

A target obtained by executing this command may be one of the followings.

1. A target of a move-type Task other than vision\_move, with info including the motion type (joint motion or linear motion), gripper ID, and velocity.
2. A target of a vision\_move Task, with info including the label, the total number of workobjects picked, planned number of workobjects for the next multi-pick, suction cup corner ID, TCP offset, workobject orientation, workobject group dimensions.
3. A target of a vision\_move Task, with info output from Step "Procedure Out" in the Mech-Vision project. If the port type of the Step is set to "Dynamic", the target will also include the custom data types.

---

**Note:** Usually, this command is used for projects in which the workobjects are boxes.

---

---

**Note:** Executing this command once only gets one target. If expecting to obtain multiple targets, please execute this command multiple times.

---

## Command Sent

210, expected data format

### Expected data format

Below are four expected data formats for the move target and vision planning result.

1. JPs, motion type, gripper ID, velocity, custom vision output 1, ..., custom vision output N
2. TCP, motion type, gripper ID, velocity, custom vision output 1, ..., custom vision output N
3. JPs, motion type, gripper ID, velocity, vision plan results, custom vision output 1, ..., custom vision output N
4. TCP, motion type, gripper ID, velocity, vision plan results, custom vision output 1, ..., custom vision output N

### Custom vision outputs

The custom vision outputs are data of the specified output port types of Step "Procedure Out" in the Mech-Vision Project.

### Vision plan results

The planning results for the vision\_move Task, including the following info:

- Labels: a field for 10 integers
- Total number of workobjects picked
- Planned number of workobjects for current multi-pick
- Corner-edge ID: the suction cup corner ID can be found in the Vacuum Gripper Configurator in the vision\_move Task
- TCP offset
- Object orientation
- Workobject group dimensions

### Data Returned

210, status code, target transmission completion status, target type, pose, motion type, gripper ID, velocity, vision plan results \*, custom vision outputs \*

\* Whether there are vision plan results and/or custom vision outputs depends on the expected data format sent when executing the command.

#### Status code

If there is no error, status code 2100 will be returned. Otherwise, the corresponding error code will be returned.

#### Target transmission completion status

- 0: Not all targets have been transmitted.
- 1: All targets have been transmitted.

#### Target type

- 0: The target is not a vision\_move Task target.
- 1: The target is a vision\_move Task.

#### Pose

The target pose in JPs or TCP, depending on the parameter sent.

#### Motion type

- 1: joint motion, MOVEJ
- 2: linear motion, MOVEL

#### Gripper ID

The ID of the gripper used on the target. -1 means no gripper is used.

#### Velocity

The percentage value of the velocity at the target, i.e., the velocity set in the move-type Task in the Mech-Viz project.

#### Vision plan results

The planning result info in the target (if the target is a vision\_move target). Usually used in carton multi-pick, depalletizing, etc. The info includes:

- Labels: Object labels. Up to 10 positive integer labels are supported. The default values are zeros.
- Total number of workobjects picked
- Planned number of workobjects for the next multi-pick
- Corner ID. The ID to indicate which corner of the gripper the workobjects are close to.
- TCP offset: The offset from the original TCP on the workobject center to the actual TCP.
- Orientation: The orientation of the workobject frame x-axis relative to the TCP x-axis.
- Workobject dimensions.

#### Custom data from Mech-Vision

The custom data for the corresponding port of custom data type on Step "Procedure Out".

The custom data parameters are arranged in alphabetical order A–Z by port name.

Aside from poses and labels, data from all other ports are considered custom data.

#### Command 501: Input Object Dimensions to Mech-Vision

This command is for dynamically inputting object dimensions into the Mech-Vision project.

Please confirm the actual object dimensions before running the Mech-Vision project.

The Mech-Vision project should have the read\_object\_dimensions Step, and the Step's parameter **Read Object Dimensions from Parameters** should be set to **True**.

#### Command Sent

```
501, project number, length, height, width
```

##### Project number

The integer ID number of the Mech-Vision project in Mech-Center, i.e., the number shown on the left of the project path in *Deployment Settings* → *Mech-Vision* in Mech-Center. The number can be adjusted by dragging the projects.

##### Length, height, width

The object dimensions to input to the Mech-Vision project.

These values will be read by the read\_object\_dimensions Step.

Unit: mm

### Data Returned

501, status code

#### Status code

If there is no error, status code 1108 will be returned. Otherwise, the corresponding error code will be returned.

### Test Samples

Command 501 is executed without error.

```
TCP send string: 501, 1, 100, 200, 300
TCP receive string: 501, 1108
```

Error: Error code 3002, missing height value.

```
TCP send string: 501, 1, 100, 200
TCP receive string: 501, 3002
```

### Command 502: Input TCP to Mech-Viz

This command is for dynamically inputting robot TCP into the Mech-Viz project.

The Task that receives the robot TCP is `outer_move`.

Please deploy the Mech-Viz project based on the template project at `/Mech-Center/tool/viz_project/outer_move`, and put the `outer_move` Task to a proper position in the workflow.

This command needs to be executed before executing command 201.

### Command Sent

502, TCP

#### Robot TCP

The TCP data used to set the target for the Task `outer_move`.

### Data Returned

502, status code

#### Status code

If there is no error, status code 2107 will be returned. Otherwise, the corresponding error code will be returned.

## Test Samples

Command 502 is executed without error.

```
TCP send string: 502, 0, 10, 10, 20, 0, 0
TCP received string: 502, 2107
```

## Command 601: Notify

The user does not need to initiate this command.

When the Mech-Viz/Mech-Vision project runs to Step notify\_vision or Task notify\_viz, Mech-Center will send the custom notification message defined in Step notify\_vision or Task notify\_viz.

In Mech-Vision, Step notify\_vision must be named "Standard Interface Notify".

In Mech-Viz, in the parameters of Task notify\_viz, please check "StandardInterface" under "Receiver".

## Command Sent

None

## Data Returned

601, custom notification message

### Custom notification message

The notification message defined in the **Notify** Task/Step. The message must be an integer.

## Test Sample

When the notification message defined in the **Notify** Task/Step is set to integer 1000, the project will return 1000 when running through the **Notify** Task/Step.

```
TCP receive string = 601, 1000
```

## Command 701: Calibration

This command is for hand-eye calibration (camera extrinsic parameter calibration).

This command syncs the calibration status with Mech-Vision and fetches each calibration point that the robot needs to reach.

This command needs to be executed multiple times to complete the calibration.



## Command Sent

701, calibration status, flange pose, JPs

### Calibration status

- 0: Tell Mech-Vision to initiate the calibration.
- 1: The previous calibration point has been received by the robot.
- 2: The previous calibration point failed to be received by the robot.

### Flange pose

The current flange pose of the robot.

### JPs

The current JPs of the robot.

## Data Returned

701, status code, calibration status, next calibration point's flange pose, next calibration point's JPs

### Status code

This status code is for indicating the status of receiving the calibration point. If the calibration point is transmitted normally, status code 7101 will be returned. Otherwise, the corresponding error code will be returned.

### Calibration status

- 1: Calibration is in progress.
- 0: Calibration finished.

### Next calibration point's flange pose

The flange pose of the next calibration point the robot should move to.

### Next calibration point's JPs

The JPs of the next calibration point the robot should move to.

## Test Samples

Initiate the calibration.

```
TCP send string = 701, 0, 1371.62147, 25.6, 1334.3529, 148.58471, -179.24347, 88.75702, 88.  
↔86102, -7.11107, -28.82309, -0.44014, -67.6509, 31.4764  
TCP received string = 701, 7101, 0, 1271.6969, -743374, 1334.34094, -3128422, 1792412, -91.  
↔11236, 93.28109, -12.0273, -32.8811, -0.37183, -68.41364, 27.02411
```

Obtain the calibration point from Mech-Vision (this process needs to be repeated to obtain multiple calibration points).

```
TCP send string = 701, 1, 1271.6969, -74.3374, 1334.34094, -3128422, 1792412 -91.11236, 93.
↪28109, -12.0273, -32.8811, -0.37183, -68.41364, 27.02411
TCP received string = 701, 7101, 0, 1471.62226, -74.40452, 1334.34235, 148.56924, -179.24432, ↪
↪88.74148, 92.8367, -2.14999, -24.25433, -0.39222, -67.23261, 27.485225
```

Finish the calibration.

```
TCP send string = 701, 1, 1371.60876, 25.53615, 1384.45532, -20.82704, 179.22026, -72.77879, ↪
↪88.88467, -7.42242, -26.68142, -0.2991, -69.95593, 39.26262
TCP received string = 701 7101, 1, 1371.62147, 25.6, 1334.3529, 148, 58471, -179 24347, 88.
↪75702, 88.86102, -7.11107, -28.82309, -0.44014, -67.6509, 31.4764
```

### Command 901: Get Software Status

This command is designed for checking the software running status of Mech-Vision, Mech-Viz, and Mech-Center. At present, this command only supports checking whether Mech-Vision is ready for running the project.

#### Command Sent

901

No parameters.

#### Data Returned

901, status code

Status code

Software status. 1101 means the Mech-Vision project is ready to run. Other codes mean the project is not ready.

#### Test Samples

Mech-Vision is ready for running the project.

```
TCP send string = 901
TCP received string = 901, 1101
```

Mech-Vision is not ready for running the project. Please open the project in Mech-Vision, right-click on the project in **Projects List**, and check *Autoload Project*.

```
TCP send string = 901
TCP received string = 901, 1001, 1
```

### 3.3.3 Siemens PLC

Mech-Mind Software Suite can communicate with Siemens SIMATIC S7 PLCs through the Siemens S7 Standard Interface. For setup instructions, please refer to `standard_interface_siemens_s7_tia_portal` and `standard_interface_siemens_s7_step_7`.

The commands are as follows:

- *Command 101: Start Mech-Vision Project*
- *Command 102: Get Vision Target(s)*
- *Command 103: Switch Mech-Vision Recipe*
- *Command 201: Start Mech-Viz Project*
- *Command 202: Stop Mech-Viz Project*
- *Command 203: Select Mech-Viz Branch*
- *Command 204: Set Move Index*
- *Command 205: Get Planned Path*
- *Command 206: Get DO List*
- *Command 501: Input Object Dimensions to Mech-Vision*
- *Command 502: Input TCP to Mech-Viz*
- *Command 901: Get Software Status*

#### Command 101: Start Mech-Vision Project

This command starts the running of the Mech-Vision project, which executes image capturing, and performs vision recognition.

If the project works in the eye-in-hand mode, the robot pose for image capturing needs to be transmitted by this command into the project.

This command is for scenarios using only Mech-Vision.

#### Command Sent

Parameter	DB offset
Command code 101	2.0
Project number	8.0
Expected number of vision points	6.0
Robot pose type	4.0
Robot pose	12.0 (JPs) or 36.0 (flange pose)

Project number

The integer ID number of the Mech-Vision project in Mech-Center, i.e., the number shown on the left of the project path in *Deployment Settings* → *Mech-Vision* in Mech-Center. The number can be adjusted by dragging the projects.

### Expected number of vision points

The number of vision points (i.e., vision poses and their corresponding point clouds, labels, indices, etc.) to expect Mech-Vision to output.

0

Get all the vision points from the Mech-Vision project's recognition results.

`integers > 0`

Get the specified number of vision points.

If the total number of vision points is smaller than the parameter value, all the available vision points will be returned.

If the total number of vision points is greater than or equal to the parameter value, vision points in the quantity of the parameter value will be returned.

---

**Note:** The command to obtain the vision points is command 102.

---

### Robot pose type

This parameter indicates the type of the image-capturing pose of the real robot to input to Mech-Vision.

0

No robot pose needs to be transmitted by this command.

If the project works in the eye-to-hand mode, then image capturing has nothing to do with the robot's pose, so no robot image-capturing pose is needed by Mech-Vision.

1

The robot pose transmitted by this command is JPs.

2

The robot pose transmitted by this command is a flange pose.

### Robot pose

This parameter is the robot pose needed when the project works in the eye-in-hand mode.

The robot pose is either JPs or flange pose, according to the setting of the parameter **robot pose type**.

**Data Returned**

Parameter	DB offset
Status code	200.0

**Status code**

If there is no error, status code 1102 will be returned. Otherwise, the corresponding error code will be returned.

**Command 102: Get Vision Target(s)**

This command gets the vision targets, i.e., robot TCPs to reach the object poses contained in the vision points, after executing command 101.

Mech-Center automatically transforms the object poses in the vision points to their corresponding robot TCPs. The process is as follows:

- Rotate the poses around their Y axes by 180 degrees.
- Determine whether the definition of the reference frame used by the robot model involves robot base height, and add a vertical offset accordingly.

**Command Sent**

Parameter	DB offset
Command code 102	2.0
Project number	8.0

**Project number**

The integer ID number of the Mech-Vision project in Mech-Center, i.e., the number shown on the left of the project path in *Deployment Settings* → *Mech-Vision* in Mech-Center. The number can be adjusted by dragging the projects.

**Data Returned**

Parameter	DB offset
Status code	200.0
Data transmission status	202.0
Number of targets	204.0
Reserved field	/
Poses	208.0
Labels	1168.0

---

**Note:** The targets (up to 20 targets by default) are located at the tail of the data returned.

---

### Status code

If there is no error, status code 1100 will be returned. Otherwise, the corresponding error code will be returned.

After executing this command, if the results from Mech-Vision have not been returned, Mech-Center will wait before sending the results to the robot. The default wait time is 10 seconds. If a timeout occurs, the timeout error status code will be returned.

### Data transmission status

This parameter indicates whether the data returned includes newly arrived targets.

1: The targets in the data returned are new data. **After PLC reads all the targets (robot TCPs and labels), please reset this field.**

---

**Note:** By default, executing command 102 once can fetch at most 20 targets (robot TCPs and labels) at a time. If the expected number of targets is greater than 20, please execute command 102 multiple times.

---

### Number of targets

The number of targets returned from the Mech-Vision project by executing this command this time.

Default range: 0 to 20.

### Reserved field

This field is not used.

The value defaults to 0.

### Poses

For this command, the robot pose is in the form of TCP, not JPs.

A TCP includes the Cartesian coordinates (XYZ) and Euler angles (ABC).

### Labels

The integer label assigned to the pose. If in the Mech-Vision project, the labels are strings, they need to be mapped to integers before outputting from the Mech-Vision project using Step label\_mapping. If there are no labels in the Mech-Vision project, the label defaults to 0.

## Command 103: Switch Mech-Vision Recipe

This command switches the parameter recipe used by the Mech-Vision project.

In Mech-Vision, you can change the settings of a group of parameters by switching the parameter recipe.

Parameters involved in recipe switching usually include point cloud matching model, image matching template, ROI, confidence threshold, etc.

This command needs to be used before executing command 101 which starts the Mech-Vision project.

**Command Sent**

Parameter	DB offset
Command code 103	2.0
Project number	8.0
Recipe number	10.0

**Project number**

The integer ID number of the Mech-Vision project in Mech-Center, i.e., the number shown on the left of the project path in *Deployment Settings* → *Mech-Vision* in Mech-Center. The number can be adjusted by dragging the projects.

**Recipe number**

The ID number of the parameter recipe to switch to, i.e., the number on the left of the parameter recipe name in *Project Assistance* → *Parameter Recipe* → *Parameter Recipe Editor* in Mech-Vision.

Recipe number range: 1—99.

**Data Returned**

Parameter	DB offset
Status code	200.0

**Status code**

If there is no error, status code 1107 will be returned. Otherwise, the corresponding error code will be returned.

**Command 201: Start Mech-Viz Project**

This command is for scenarios using both Mech-Vision and Mech-Viz.

This command starts the running of the Mech-Viz project, calls the corresponding Mech-Vision project, and lets the Mech-Viz project plan the robot path based on the vision points from Mech-Vision.

For the Mech-Viz project that needs starting, the option *Autoload* should be enabled in Mech-Viz's interface.

Please see [Example Mech-Viz Projects for Standard Interface](#) for the description of example Mech-Viz projects.

**Command Sent**

Parameter	DB offset
Command code	2.0
Pose type	4.0
Robot pose	12.0

**Pose type**

0

The current pose of the robot is not needed by Mech-Viz and no pose will be sent.

If the project works in the eye-to-hand mode, no robot image-capturing pose will be needed by the project.

In Mech-Viz, the simulated robot will move from the initial pose JPs = [0, 0, 0, 0, 0, 0] to the first target in the planned path.

1

The robot pose will be sent to Mech-Viz and the pose sent is in the form of JPs.

In Mech-Viz, the simulated robot will move from the input initial pose (i.e., the pose sent by this command) to the first target in the planned path.

TCP is not supported at present.

---

**Note:** If in the scene, there are barriers that stand in the way from the initial pose JPs = [0, 0, 0, 0, 0, 0] to the first target in the planned path, the pose type must be set to 1.

---

**Robot pose**

The current JPs of the real robot (if pose type is set to 1).

A robot pose consists of six REAL numbers.

**Data Returned**

Parameter	DB offset
Status code	200.0

**Status code**

If there is no error, status code 2103 will be returned. Otherwise, the corresponding error code will be returned.



### Command 202: Stop Mech-Viz Project

This command stops the running of the Mech-Viz project. This command is needed only when the Mech-Viz project fall into an infinite loop or cannot be stopped normally.

#### Command Sent

Parameter	DB offset
Command code 202	2.0

#### Data Returned

Parameter	DB offset
Status code	200.0

#### Status code

If there is no error, status code 2104 will be returned. Otherwise, the corresponding error code will be returned.

### Command 203: Select Mech-Viz Branch

This command specifies which branch the project should run along. For this command, the branching is implemented by a `branch_by_msg` Task, and this command selects the branch by specifying an exit port of the Task.

Before executing this command, the Mech-Viz project needs to be started by executing command 201.

When the Mech-Viz project runs to the `branch_by_msg` Task, it will wait for command 203 to specify which exit port of the Task, i.e., the branch, the project should run along.

#### Command Sent

Parameter	DB offset
Command code 203	2.0
Branching Task ID	60.0
Exit port number	62.0

#### Branching Task ID

This parameter is for specifying which `branch_by_msg` Task the branch selection should apply to.

The Task ID can be read in the Task's parameters.

#### Exit port number

This parameter is for specifying which exit port of the specified Task, i.e., the branch, the project should run along. The value should be an integer ([1, N]).

---

**Note:** An exit port number is the 1-based index of the specified exit port on the Task. For example, if the specified exit port is the second exit port of the Task from left to right, the exit port number is 2.

---

### Data Returned

Parameter	DB offset
Status code	200.0

### Status code

If there is no error, status code 2105 will be returned. Otherwise, the corresponding error code will be returned.

### Command 204: Set Move Index

This command is for setting the index parameter of a Task that involves sequential or separate motions or operations.

Tasks with index parameters include `move_list`, `move_grid`, `custom_pallet_pattern`, `smart_pallet_pattern`, etc.

Before executing this command, command 201 needs to be executed to start the Mech-Viz project.

### Command Sent

Parameter	DB offset
Command code 204	2.0
Task ID	64.0
Index value	66.0

### Task ID

This parameter specifies which Task the index setting should apply to.

The Task ID can be read in the Task's parameters.

### Index value

The index value that should be set the next time this Task is executed.

When this command is sent, the current index value in Mech-Viz will become the parameter value minus 1.

When the Mech-Viz project runs to the task specified by this command, the current index value in Mech-Viz will be increased by 1 to become the parameter's value.

**Data Returned**

Parameter	DB offset
Status code	200.0

**Status code**

If there is no error, status code 2106 will be returned. Otherwise, the corresponding error code will be returned.

**Command 205: Get Planned Path**

This command gets the robot motion path planned by Mech-Viz after command 201 is executed to start the Mech-Viz project.

---

**Note:** If one of the targets in the path is not supposed to be sent to the robot, please clear the parameter "Send Target" checkbox of the corresponding move-type Task.

---

**Command Sent**

Parameter	DB offset
Command code 205	2.0
Target type 4.0	

**Target type**

This parameter specifies the type of path targets to return from Mech-Viz.

1

The targets returned should be in JPs.

2

The targets returned should be in TCP.

**Data Returned**

Parameter	DB offset
Status code	200.0
Data transmission status	202.0
Number of points	204.0
Position of "visual_move"	206.0
Targets' poses	208.0
Targets' labels	1168.0
Targets' velocities	1248.0

### Status code

If there is no error, status code 2100 will be returned. Otherwise, the corresponding error code will be returned.

---

**Note:** When executing this command, if Mech-Viz has not yet had the planned robot motion path (the project is still running), Mech-Center will wait. The default wait time is 10 seconds. If a timeout occurs, a timeout error code will be returned.

---

### Data transmission status

This parameter indicates whether the data returned includes newly arrived targets.

1

The targets in the data returned are new data.

**After PLC reads all the target data (poses, labels, velocities), please reset this field.**

---

**Note:** By default, executing command 205 once can fetch at most 20 targets (poses, labels, velocities) at a time. If the number of targets is greater than 20, please execute command 205 multiple times.

---

### Number of points

This parameter indicates the number of path targets ([pose, label, velocity]) sent by executing this command this time.

Default range: 0 to 20.

### Position of "visual\_move"

The position of the vision\_move Task, i.e., the move to the vision pose (usually the pose for picking the object) in the entire robot motion path.

For example, if the path is composed of Tasks **move\_1**, **move\_2**, **visual\_move**, **move\_3** sequentially, the position of **visual\_move** is 3.

If in the path there is no vision\_move Task, the returned value will be 0.

### Poses

Each pose includes Cartesian coordinates (XYZ) and Euler angles (ABC), or JPs, according to the target type set by this command.

### Labels

A label is the integer label assigned to a pose. If in the Mech-Vision project, the labels are strings, they need to be mapped to integers before outputting from the Mech-Vision project using Step label\_mapping. If there are no labels in the Mech-Vision project, the label defaults to 0.

### Velocities

A velocity is the non-zero velocity parameter percentage value for the corresponding move-type Task in Mech-Viz.

**Command 206: Get DO List**

This command gets the planned DO signal list when there are multiple grippers, such as suction cup sections, to control.

For using this command:

In the parameters of the "set\_do\_list" Task:

- Check "StandardInterface" under "Receiver"
- Check "Get DO List from VisualMove"
- Select a "visual\_move" Task that needs the DO signal list at the bottom of the parameter panel

Before calling this command, command 205 needs to be executed to obtain the planned motion path by Mech-Viz.

Please deploy the Mech-Viz project based on the template project at `/Mech-Center/tool/viz_project/suction_zone`, and set the suction cup configuration file in the Mech-Viz project.

**Command Sent**

Parameter	DB offset
Command code 206	2.0

No parameters.

**Data Returned**

Parameter	DB offset
Status code	200.0
DO signals	1408.0

**Status code**

If there are no errors, status code 2102 will be returned. Otherwise, the corresponding error code will be returned.

**DO signal value**

There are 64 DO signal values, in integers, located at the tail of the data returned.

Range of valid DO values: [0, 999]. Placeholder value: -1.

### Command 501: Input Object Dimensions to Mech-Vision

This command is for dynamically inputting object dimensions into the Mech-Vision project.

Please confirm the actual object dimensions before running the Mech-Vision project.

The Mech-Vision project should have the `read_object_dimensions` Step, and the Step's parameter **Read Object Dimensions from Parameters** should be set to `True`.

#### Command Sent

Parameter	DB offset
Command code 501	2.0
Project number	8.0
[length, height, width]	68.0

#### Project number

The integer ID number of the Mech-Vision project in Mech-Center, i.e., the number shown on the left of the project path in *Deployment Settings* → *Mech-Vision* in Mech-Center. The number can be adjusted by dragging the projects.

#### [length, height, width]

The object dimensions to input to the Mech-Vision project.

Those values will be read by the `read_object_dimensions` Step.

Unit: mm

#### Data Returned

Parameter	DB offset
Status code	200.0

#### Status code

If there is no error, status code 1108 will be returned. Otherwise, the corresponding error code will be returned.

### Command 502: Input TCP to Mech-Viz

This command is for dynamically inputting robot TCP into the Mech-Viz project.

The Task that receives the robot TCP is `outer_move`.

Please deploy the Mech-Viz project based on the template project at `/Mech-Center/tool/viz_project/outer_move`, and put the `outer_move` Task to a proper position in the workflow.

This command needs to be executed before executing command 201.

**Command Sent**

Parameter	DB offset
Command code 502	2.0
TCP	80.0

**Robot TCP**

The TCP data used to set the target for the Task outer\_move.

**Data Returned**

Parameter	DB offset
Status code	200.0

**Status code**

If there is no error, status code 2107 will be returned. Otherwise, the corresponding error code will be returned.

**Command 901: Get Software Status**

This command is designed for checking the software running status of Mech-Vision, Mech-Viz, and Mech-Center. At present, this command only supports checking whether Mech-Vision is ready for running the project.

**Command Sent**

Parameter	DB offset
Command code 901	2.0

No parameters.

**Data Returned**

Parameter	DB offset
Status code	200.0

**Status code**

Software status. 1101 means the Mech-Vision project is ready to run. Other codes mean the project is not ready.

### 3.3.4 PROFINET

Mech-Mind Software Suite can communicate with Siemens SIMATIC S7 PLCs through the PROFINET Standard Interface. For setup instructions, please refer to `standard_interface_profinet_siemens_simatic_s7`.

#### Protocol

*From Mech-Center to PLC*

- *Control\_Output*
  - *Command\_Complete*
  - *Data\_Ready*
  - *Exposure\_Complete*
  - *Trigger\_Acknowledge*
- *Heartbeat*
- *Status\_Code*
- *Calib\_Cam\_Status*
- *Send\_Pose\_Num*
- *Visual\_Point\_Index*
- *DO\_List*
- *Notify\_Message*
- *Send\_Pose\_Type*
- *Target\_Pose*
- *Target\_Label*
- *Target\_Speed*
- *Ext\_Output\_Data*

*From PLC to Mech-Center*

- *Control\_Input*
  - *Reset\_Notify*
  - *Data\_Acknowledge*
  - *Reset\_Exposure*
  - *Trigger*
  - *Comm Enable*
- *Command*
- *Calib\_Rob\_Status*
- *Vision\_Proj\_Num*
- *Vision\_Recipe\_Num*



- *Viz\_Task\_Name*
- *Viz\_Task\_Value*
- *Req\_Pose\_Num*
- *Req\_Pose\_Type*
- *Robot\_Pose\_JPS*
- *Robot\_Pose\_TCP* (will be renamed to *Robot\_Pose\_Flange*)
- *Ext\_Input\_Data*

## From Mech-Center to PLC

### Control\_Output

Bit	Data
7	/
6	/
5	/
4	Command execution complete (Bool)
3	Data ready (Bool)
2	Camera exposure complete (Bool)
1	Trigger Acknowledge (Bool)
0	Heartbeat (Bool)

### Command\_Complete

This module is for indicating that the execution of a command has been completed, and the data returned by the command can be read.

For commands 102 and 205, only when the last byte of data has been returned will this module's signal be set to 1.

### Data\_Ready

This module is for indicating that new data has been sent from Mech-Center to the PLC, and the PLC can read the data.

This module's signal is for command 102 or command 205.

### **Exposure\_Complete**

When the camera completes the exposure, Exposure Complete will be set to 1, indicating that the object can be moved or the robot working eye-in-hand can move.

### **Trigger\_Acknowledge**

Trigger Acknowledge = 1 means Mech-Mind Software Suite has been triggered successfully by the Trigger signal.

Trigger Acknowledge will stay at 1 until the Trigger signal is reset to 0.

### **Heartbeat**

System heartbeat that flips every 1 second.

### **Status\_Code**

The command execution status code returned from Mech-Center.

It may be a normal status code or an error code.

Status code. INT32

### **Calib\_Cam\_Status**

Calibration status. INT

For command 701: calibration.

- 1 means the calibration is in progress.
- 0 means the calibration has been completed.

### **Send\_Pose\_Num**

The number of poses sent by executing the command at the time.

INT8

### Visual\_Point\_Index

Position of vision\_move in the planned path. INT8

For example, if the path is composed of Tasks **move\_1**, **move\_2**, **visual\_move**, **move\_3** sequentially, the position of **visual\_move** is 3.

If in the path there is no vision\_move Task, the returned value will be 0.

### DO\_List

The 64 INT8 DO signals for controlling multiple suction cup sections or array grippers.

Byte	Data
0	DO list 1: signal 0 to 7
1	DO list 2: signal 8 to 15
2	DO list 3: signal 16 to 23
3	DO list 4: signal 24 to 31
4	DO list 5: signal 32 to 39
5	DO list 6: signal 40 to 47
6	DO list 7: signal 48 to 55
7	DO list 8: signal 56 to 63

### Notify\_Message

The customized integer message sent by a **Notify** Task/Step from Mech-Viz/Mech-Vision.

Integer message. INT32

### Send\_Pose\_Type

1 means JPs. 2 means TCP.

Type of pose sent. INT8

### Target\_Pose

Robot pose of the target in either robot TCP or JPs.

---

**Note:** The data from this module should be divided by 10000 before use.

---

A pose in Cartesian coordinates and Euler angles can be represented by:

[X, Y, Z, A, B, C]

JPs consist of up to 6 joint angles:

[J1, J2, J3, J4, J5, J6]

Byte	Data
0 to 3	X or J1, INT32
4 to 7	Y or J2, INT32
8 to 11	Z or J3, INT32
12 to 15	A or J4, INT32
16 to 19	B or J5, INT32
20 to 23	C or J6, INT32

### Target\_Label

Non-negative integer labels of targets. INT32

### Target\_Speed

Velocity parameter percentage values of the move-type Tasks of targets. INT32

### Ext\_Output\_Data

Reserved module for other data for transmission.

This module takes up 40 bytes (INT32[1: 10], 10 INT32 integers in total).

### From PLC to Mech-Center

#### Control\_Input

Bit	Data
7	/
6	/
5	/
4	Reset Notify (Bool)
3	Data Acknowledge (Bool)
2	Reset Exposure_Complete (Bool)
1	Trigger (Bool)
0	Comm Enable (Bool)

### Reset\_Notify

If Reset\_Notify = 1, the content of Notify Message will be cleared.

### Data\_Acknowledge

Data\_Acknowledge is for acknowledging having read the data returned by executing command 102 or command 205.

If Data\_Acknowledge = 0, the PLC has not read the data from Mech-Center and the data are kept at the port.

If Data\_Acknowledge = 1, the PLC has read the data from Mech-Center and Mech-Center can write the data of the next round.

Data\_Acknowledge can be reset at heartbeat flip or when Data\_Ready = 0.

### Reset\_Exposure

If Reset\_Exposure = 1, Exposure Complete will be set to 0.

### Trigger

If Trigger = 1, Mech-Center will read the command sent and the command will be executed.

Trigger Acknowledge can be reset once Mech-Center receives the Trigger signal.

The upward segment of the signal is considered as 1.

### Comm Enable

- 0: Communication disabled. Mech-Center will ignore the Trigger signal.
- 1: Communication enabled. The Trigger signal will work and Mech-Center will receive commands.

### Command

Command code.

INT32.

**Calib\_Rob\_Status**

- 0: The calibration starts.
- 1: The robot has normally moved to the last calibration point sent.
- 2: The robot failed to move to the last calibration point sent.

INT8.

**Vision\_Proj\_Num**

Mech-Vision project ID number.

The integer ID number of the Mech-Vision project in Mech-Center, i.e., the number shown on the left of the project path in *Deployment Settings* → *Mech-Vision* in Mech-Center. The number can be adjusted by dragging the projects.

INT8.

**Vision\_Recipe\_Num**

Mech-Vision parameter recipe number.

The ID number of the parameter recipe to switch to, i.e., the number on the left of the parameter recipe name in *Project Assistance* → *Parameter Recipe* → *Parameter Recipe Editor* in Mech-Vision.

Recipe number range: 1—99.

INT8.

**Viz\_Task\_Name**

The Task ID of the Mech-Viz Task involved. It can be read and set in the Task's parameters.

INT8.

**Viz\_Task\_Value**

The value to set in the Mech-Viz Task's index parameter, or Mech-Viz branching Task exit port number.

INT8.

**Req\_Pose\_Num**

Number of targets to request from Mech-Vision.

0: Request all the available vision points from the vision results in Mech-Vision.

INT8.

**Req\_Pose\_Type**

Type of targets requested. INT8.

- 0: No image-capturing robot pose is needed (eye-to-hand mode).
- 1: The image capturing robot pose sent is in JPs.
- 2: The image capturing robot pose sent is a flange pose.

INT8.

**Robot\_Pose\_JPS**

Robot JPs for image capturing.

Please multiply the JPs data by 10000 before setting it to the module.

JPs include up to 6 joint position data (6 INT32 integers):

[J1, J2, J3, J4, J5, J6]

Byte	Data
0 to 3	J1
4 to 7	J2
8 to 11	J3
12 to 15	J4
16 to 19	J5
20 to 23	J6

**Robot\_Pose\_TCP (will be renamed to Robot\_Pose\_Flange)**

Robot flang pose for image capturing.

Please multiply the pose data by 10000 before setting it to the module.

A flange pose includes Cartesian coordinates (X, Y, Z) and Euler angles (A, B, C), 6 INT32 integers in total.

[X, Y, Z, A, B, C]

Byte	Data
0 to 3	X
4 to 7	Y
8 to 11	Z
12 to 15	A
16 to 19	B
20 to 23	C

### Ext\_Input\_Data

Reserved module for other data for transmission.

This module takes up 40 bytes (INT32[1: 10], 10 INT32 integers in total).

### Profinet Commands

- *Command 101: Start Mech-Vision Project*
- *Command 102: Get Vision Target(s)*
- *Command 103: Switch Mech-Vision Recipe*
- *Command 201: Start Mech-Viz Project*
- *Command 202: Stop Mech-Viz Project*
- *Command 203: Select Mech-Viz Branch*
- *Command 204: Set Move Index*
- *Command 205: Get Planned Path*
- *Command 206: Get DO List*
- *Command 501: Input Object Dimensions to Mech-Vision*
- *Command 502: Input TCP to Mech-Viz*
- *Command 901: Get Software Status*

### Command 101: Start Mech-Vision Project

This command starts the running of the Mech-Vision project, which executes image capturing, and performs vision recognition.

If the project works in the eye-in-hand mode, the robot pose for image capturing needs to be transmitted by this command into the project.

This command is for scenarios using only Mech-Vision.



**Command Sent**

Module	Description
Command	101
Vision_Proj_Num	Project number
Req_Pose_Num	Expected number of vision points
Robot_Pose_Type	Robot pose type
Robot_Pose_JPS / Robot_Pose_TCP *	Robot pose

\* Robot\_Pose\_TCP will be renamed to Robot\_Pose\_Flange as the image capturing robot pose cannot be a TCP.

**Project number**

The integer ID number of the Mech-Vision project in Mech-Center, i.e., the number shown on the left of the project path in *Deployment Settings* → *Mech-Vision* in Mech-Center. The number can be adjusted by dragging the projects.

**Expected number of vision points**

The number of vision points (i.e., vision poses and their corresponding point clouds, labels, indices, etc.) to expect Mech-Vision to output.

0

Get all the vision points from the Mech-Vision project's recognition results.

**integers** > 0

Get the specified number of vision points.

If the total number of vision points is smaller than the parameter value, all the available vision points will be returned.

If the total number of vision points is greater than or equal to the parameter value, vision points in the quantity of the parameter value will be returned.

---

**Note:** The command to obtain the vision points is command 102.

---

**Robot pose type**

This parameter indicates the type of the image-capturing pose of the real robot to input to Mech-Vision.

0

No robot pose needs to be transmitted by this command.

If the project works in the eye-to-hand mode, then image capturing has nothing to do with the robot's pose, so no robot image-capturing pose is needed by Mech-Vision.

1

The robot pose transmitted by this command is in JPs.

2

The robot pose transmitted by this command is a flange pose.

**Robot pose**

This parameter is the robot pose needed when the project works in the eye-in-hand mode.

The robot pose is either in JPs or flange pose, according to the setting of the parameter “robot pose type”.

**Attention:** The floating point data of JPs or flange pose needs to be multiplied by 10000 before setting to the module Robot\_Pose\_JPS or Robot\_Pose\_TCP (will be renamed to Robot\_Pose\_Flange).

### Data Returned

Status code

If there is no error, status code 1102 will be returned. Otherwise, the corresponding error code will be returned.

### Command 102: Get Vision Target(s)

This command gets the vision targets, i.e., robot TCPs to reach the object poses contained in the vision points, after executing command 101.

Mech-Center automatically transforms the object poses in the vision points to their corresponding robot TCPs. The process is as follows:

- Rotate the poses around their Y axes by 180 degrees.
- Determine whether the definition of the reference frame used by the robot model involves robot base height, and add a vertical offset accordingly.

---

**Note:** In PROFINET, by default, command 102 can only fetch at most 20 targets at a time. So, command 102 may need to be repeatedly executed until all the targets required are obtained.

---

### Command Sent

Module	Description
Command	102
Vision_Proj_Num	Project number

### Project number

The integer ID number of the Mech-Vision project in Mech-Center, i.e., the number shown on the left of the project path in *Deployment Settings* → *Mech-Vision* in Mech-Center. The number can be adjusted by dragging the projects.

**Data Returned**

Module	Description
Status code	/
Send_Pose_Num	Number of targets
Send_Pose_Type	Pose type
Target_Pose	Robot TCPs
Target_Label	Labels for targets

---

**Note:** The targets (up to 20 targets) are located at the tail of the data returned.

---

**Status code**

If there is no error, status code 1100 will be returned. Otherwise, the corresponding error code will be returned.

After executing this command, if the results from Mech-Vision have not been returned, Mech-Center will wait before sending the results to the robot. The default wait time is 10 seconds. If a timeout occurs, the timeout error status code will be returned.

**Number of targets**

The number of targets returned from the Mech-Vision project by executing this command this time.

**Pose type**

This module's value defaults to 2, meaning the robot poses are in the form of TCPs, not JPs.

**Robot TCPs**

A TCP includes the Cartesian coordinates (XYZ) and Euler angles (ABC).

**Labels for targets**

The integer labels assigned to the object poses. If in the Mech-Vision project, the labels are strings, they need to be mapped to integers before outputting from the Mech-Vision project using Step label\_mapping. If there are no labels in the Mech-Vision project, the labels default to 0s.

**Attention:** For details about the sending and receiving of poses, please see [Communication Control Flowchart](#).

### Command 103: Switch Mech-Vision Recipe

This command switches the parameter recipe used by the Mech-Vision project.

In Mech-Vision, you can change the settings of a group of parameters by switching the parameter recipe.

Parameters involved in recipe switching usually include point cloud matching model, image matching template, ROI, confidence threshold, etc.

This command needs to be used before executing command 101 which starts the Mech-Vision project.

#### Command Sent

Module	Description
Command	103
Vision_Proj_Num	Project number
Vision_Recipe_Num	Recipe number

#### Project number

The integer ID number of the Mech-Vision project in Mech-Center, i.e., the number shown on the left of the project path in *Deployment Settings* → *Mech-Vision* in Mech-Center. The number can be adjusted by dragging the projects.

#### Recipe number

The ID number of the parameter recipe to switch to, i.e., the number on the left of the parameter recipe name in *Project Assistance* → *Parameter Recipe* → *Parameter Recipe Editor* in Mech-Vision.

Recipe number range: 1—99.

#### Data Returned

##### Status code

If there is no error, status code 1107 will be returned. Otherwise, the corresponding error code will be returned.

### Command 201: Start Mech-Viz Project

This command is for scenarios using both Mech-Vision and Mech-Viz.

This command starts the running of the Mech-Viz project, calls the corresponding Mech-Vision project, and lets the Mech-Viz project plan the robot path based on the vision points from Mech-Vision.

For the Mech-Viz project that needs starting, the option *Autoload* needs to be checked in Mech-Viz's interface.

Please see [Example Mech-Viz Projects for Standard Interface](#) for the description of example Mech-Viz projects.

**Command Sent**

Module	Description
Command	201
Robot_Pose_Type	Pose type
Robot_Pose_JPS	Pose

**Pose type**

0

The current pose of the robot is not needed by Mech-Viz and no pose will be sent.

If the project works in the eye-to-hand mode, no robot image-capturing pose will be needed by the project.

In Mech-Viz, the simulated robot will move from the initial pose JPs = [0, 0, 0, 0, 0, 0] to the first target in the planned path.

1

The robot pose will be sent to Mech-Viz and the pose sent is in the form of JPs.

In Mech-Viz, the simulated robot will move from the input initial pose (i.e., the pose sent by this command) to the first target in the planned path.

TCP is not supported at present.

---

**Note:** If in the scene, there are barriers that stand in the way from the initial pose JPs = [0, 0, 0, 0, 0, 0] to the first target in the planned path, the pose type must be set to 1.

---

**Pose**

The current JPs of the real robot (if pose type is set to 1).

---

**Note:** Before setting as the value of the pose module, the numerical values of the pose need to be multiplied by 10000, to transform floating point numbers into integers.

---

**Data Returned**
**Status code**

If there is no error, status code 2103 will be returned. Otherwise, the corresponding error code will be returned.

### Command 202: Stop Mech-Viz Project

This command stops the running of the Mech-Viz project. This command is needed only when the Mech-Viz project fall into an infinite loop or cannot be stopped normally.

#### Command Sent

Module	Description
Command	202

#### Data Returned

##### Status code

If there is no error, status code 2104 will be returned. Otherwise, the corresponding error code will be returned.

### Command 203: Select Mech-Viz Branch

This command specifies which branch the project should run along. For this command, the branching is implemented by a `branch_by_msg` Task, and this command selects the branch by specifying an exit port of the Task.

Before executing this command, the Mech-Viz project needs to be started by executing command 201.

When the Mech-Viz project runs to the `branch_by_msg` Task, it will wait for command 203 to specify which exit port of the Task, i.e., the branch, the project should run along.

#### Command Sent

Module	Description
Command	203
Viz_Task_ID	branching Task ID
Viz_Task_Value	exit port number

##### Branching Task ID

This parameter is for specifying which `branch_by_msg` Task the branch selection should apply to.

The Task ID can be read in the Task's parameters.

##### Exit port number

This parameter is for specifying which exit port of the specified Task, i.e., the branch, the project should run along. The value should be an integer ( $[1, N]$ ).

**Note:** An exit port number is the 1-based index of the specified exit port on the Task. For example, if the specified exit port is the second exit port of the Task from left to right, the exit port number is 2.

### Data Returned

#### Status code

If there is no error, status code 2105 will be returned. Otherwise, the corresponding error code will be returned.

### Command 204: Set Move Index

This command is for setting the index parameter of a Task that involves sequential or separate motions or operations.

Tasks with index parameters include `move_list`, `move_grid`, `custom_pallet_pattern`, `smart_pallet_pattern`, etc.

Before executing this command, command 201 needs to be executed to start the Mech-Viz project.

### Command Sent

Module	Description
Command	204
Viz_Task_ID	Task ID
Viz_Task_Value	Index value

#### Task ID

This parameter specifies which Task the index setting should apply to.

The Task ID can be read in the Task's parameters.

#### Index value

The index value that should be set the next time this Task is executed.

When this command is sent, the current index value in Mech-Viz will become the parameter value minus 1.

When the Mech-Viz project runs to the task specified by this command, the current index value in Mech-Viz will be increased by 1 to become the parameter's value.

**Data Returned**
**Status code**

If there is no error, status code 2106 will be returned. Otherwise, the corresponding error code will be returned.

**Command 205: Get Planned Path**

This command gets the robot motion path planned by Mech-Viz after command 201 is executed to start the Mech-Viz project.

---

**Note:** Please uncheck the parameter "Send Target" of the corresponding move-type Task.

---



---

**Note:** In PROFINET, by default, command 205 can only fetch at most 20 targets of the planned path at a time. So, command 205 may need to be executed repeatedly until all the targets required are obtained.

---

**Command Sent**

Module	Description
Command	205
Req_Pose_Type	Target type

**Target type**

This parameter specifies the type of poses in the path targets to return from Mech-Viz.

- 1: The targets returned should be in JPs.
- 2: The targets returned should be in TCP.

**Data Returned**

Module	Description
Status code	/
Send_Pose_Num	Number of points
Send_Pose_Type	Pose type in target points
Visual_Point_Index	Position of "visual_move"
Target_Pose	Poses in targets
Target_Label	Labels in targets
Target_Speed	Velocities in targets

**Status code**



If there is no error, status code 2100 will be returned. Otherwise, the corresponding error code will be returned.

---

**Note:** When executing this command, if Mech-Viz has not yet had the planned robot motion path (the project is still running), Mech-Center will wait. The default wait time is 10 seconds. If a timeout occurs, a timeout error code will be returned.

---

### Number of points

This parameter indicates the number of path targets ([pose, label, velocity]) sent by executing this command this time.

---

**Note:** In PROFINET, by default, command 205 can only send at most 20 targets at a time. So, command 205 may need to be repeatedly executed until all the targets required are sent.

---

### Pose type in target points

Same as the sent value in module "Req\_Pose\_Type".

1

JPs

2

TCP

### Position of "visual\_move"

The position of the vision\_move Task, i.e., the move to the vision pose (usually the pose for picking the object) in the entire robot motion path.

For example, if the path is composed of Tasks **move\_1**, **move\_2**, **visual\_move**, **move\_3** sequentially, the position of **visual\_move** is 3.

If in the path there is no vision\_move Task, the returned value will be 0.

### Poses in targets

A pose includes Cartesian coordinates (XYZ) and Euler angles (ABC), or JPs, according to the pose type set by command 205.

### Labels in targets

Label is the integer label assigned to the pose. If in the Mech-Vision project, the labels are strings, they need to be mapped to integers before outputting from the Mech-Vision project using Step label\_mapping. If there are no labels in the Mech-Vision project, the label defaults to 0.

### Velocities in targets

A velocity value is the non-zero velocity parameter percentage value for the move-type Task set in Mech-Viz.

### Command 206: Get DO List

This command gets the planned DO signal list when there are multiple grippers, such as suction cup sections, to control.

For using this command:

In the parameters of the "set\_do\_list" Task:

- Check "StandardInterface" under "Receiver"
- Check "Get DO List from VisualMove"
- Select a "visual\_move" Task that needs the DO signal list at the bottom of the parameter panel

Before calling this command, command 205 needs to be executed to obtain the planned motion path by Mech-Viz.

Please deploy the Mech-Viz project based on the template project at `/Mech-Center/tool/viz_project/suction_zone`, and set the suction cup configuration file in the Mech-Viz project.

### Command Sent

Module	Description
Command	206

### Data Returned

Module	Description
Status code	/
DO List	DO signal values

#### Status code

If there are no errors, status code 2102 will be returned. Otherwise, the corresponding error code will be returned.

#### DO signal values

There are 64 DO signal values, in integers, located at the tail of the data returned.

Range of valid DO values: [0, 999]. Placeholder value: -1.

### Command 501: Input Object Dimensions to Mech-Vision

This command is for dynamically inputting object dimensions into the Mech-Vision project.

Please confirm the actual object dimensions before running the Mech-Vision project.

The Mech-Vision project should have the `read_object_dimensions` Step, and the Step's parameter **Read Object Dimensions from Parameters** should be set to **True**.

#### Command Sent

Module	Description
Command	501
Vision_Proj_Num	Project number
Ext_Input_Data	[length, height, width]

#### Project number

The integer ID number of the Mech-Vision project in Mech-Center, i.e., the number shown on the left of the project path in *Deployment Settings* → *Mech-Vision* in Mech-Center. The number can be adjusted by dragging the projects.

#### Length, height, width

The object dimensions to input to the Mech-Vision project.

Those values will be read by the `read_object_dimensions` Step.

Unit: mm

#### Data Returned

##### Status code

If there is no error, status code 1108 will be returned. Otherwise, the corresponding error code will be returned.

### Command 502: Input TCP to Mech-Viz

This command is for dynamically inputting robot TCP into the Mech-Viz project.

The Task that receives the robot TCP is `outer_move`.

Please deploy the Mech-Viz project based on the template project at `/Mech-Center/tool/viz_project/outer_move`, and put the `outer_move` Task to a proper position in the workflow.

This command needs to be executed before executing command 201.

**Command Sent**

Module	Description
Command	502
Ext_Input_Data	Robot TCP

**Robot TCP**

The TCP data used to set the target for the Task outer\_move.

---

**Note:** Please multiply the original TCP data (unit: mm) by 10,000 before setting the parameter.

---

**Data Returned**
**Status code**

If there is no error, status code 2107 will be returned. Otherwise, the corresponding error code will be returned.

**Command 901: Get Software Status**

This command is designed for checking the software running status of Mech-Vision, Mech-Viz, and Mech-Center. At present, this command only supports checking whether Mech-Vision is ready for running the project.

**Command Sent**

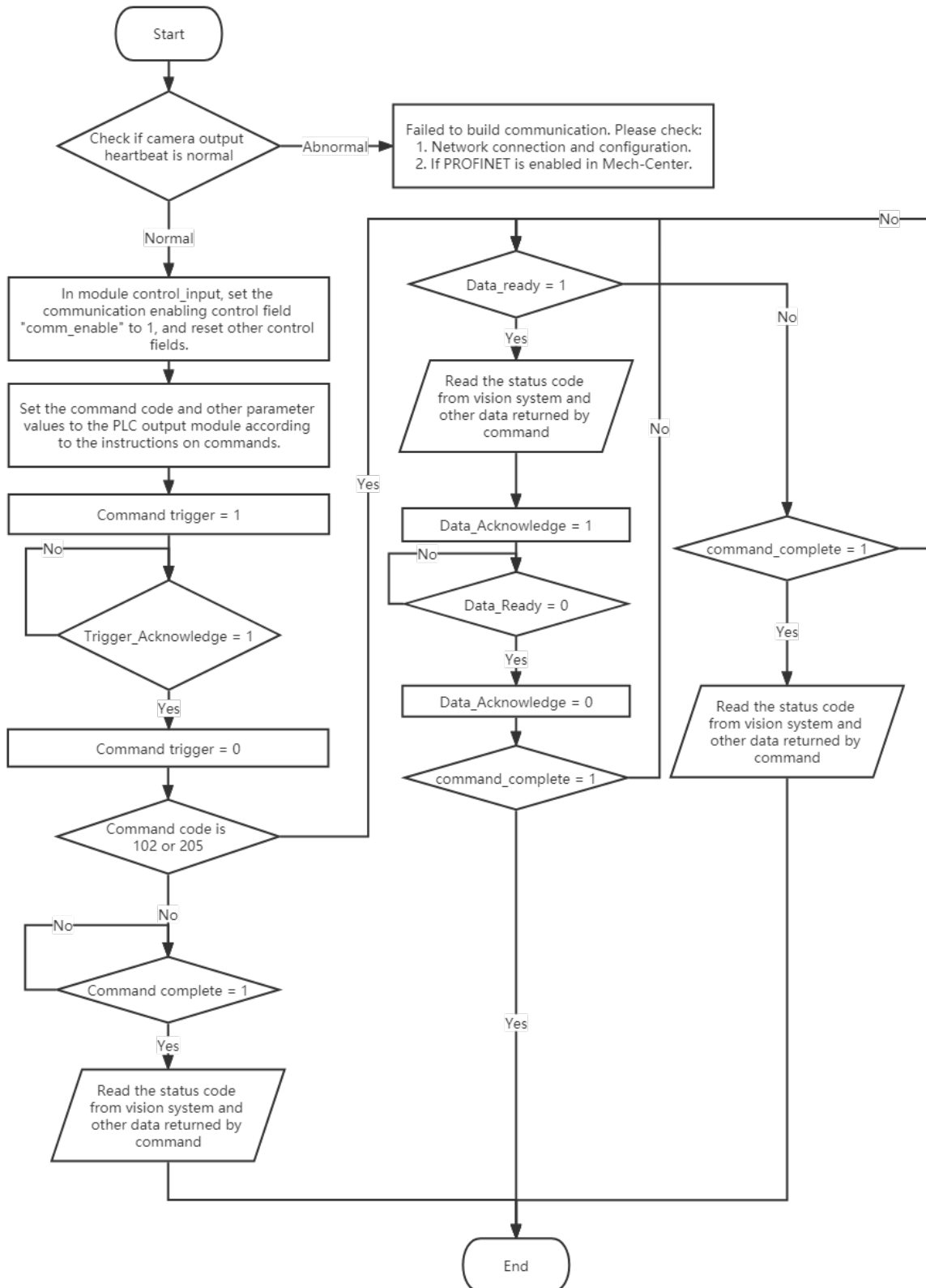
Module	Description
Command	901

**Data Returned**
**Status code**

Software status. 1101 means the Mech-Vision project is ready to run. Other codes mean the project is not ready.



Communication Control Flowchart



### 3.3.5 EtherNet/IP

For development instructions for EtherNet/IP, please see [PROFINET](#).

Mech-Mind Software Suite can communicate with some Keyence and Omron PLCs through the EtherNet/IP Standard Interface. For setup instructions, please refer to `standard_interface_ethernetip_keyence` and `standard_interface_ethernetip_omron`.

### 3.3.6 Modbus TCP

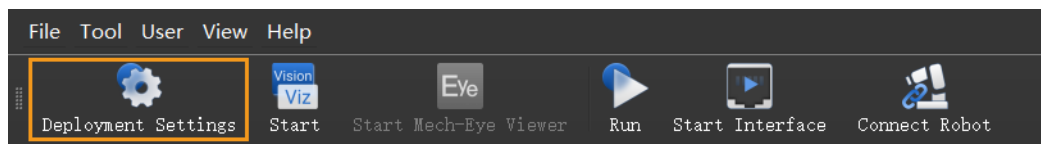
Through Standard Interface based on the Modbus TCP protocol, the Mech-Mind Software Suite can communicate with master equipment (PLC or robot controller).

For specific example programs, please see Modbus TCP–Siemens SIMATIC S7 PLC and Modbus TCP–Mitsubishi Q Series PLC.

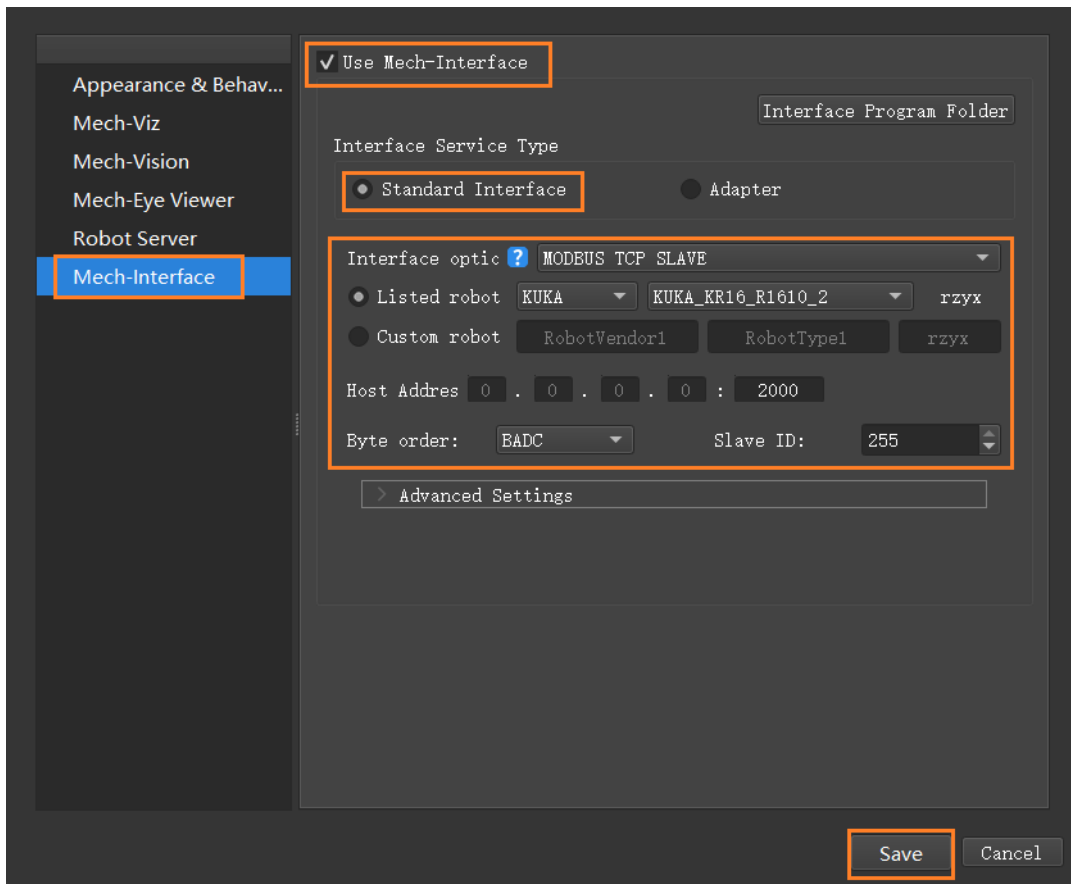
#### Communication Configuration

Mech-Center supports Modbus TCP communication. As a slave device, it provides the standard interface option Modbus TCP SLAVE for data communication with a poll master device. The specific configuration steps are as follows.

1. Configure the IP address of the industrial computer as a value in the same network segment as the master device's IP address. Open the **command prompt** of the industrial computer (you can open it by searching for `cmd`), and enter `ping xxx.xxx.xxx.xxx` (the IP address of the master station), and test whether the connection between the IPC and the master station is normal.
2. Open Mech-Center and open the *Deployment Settings* window.



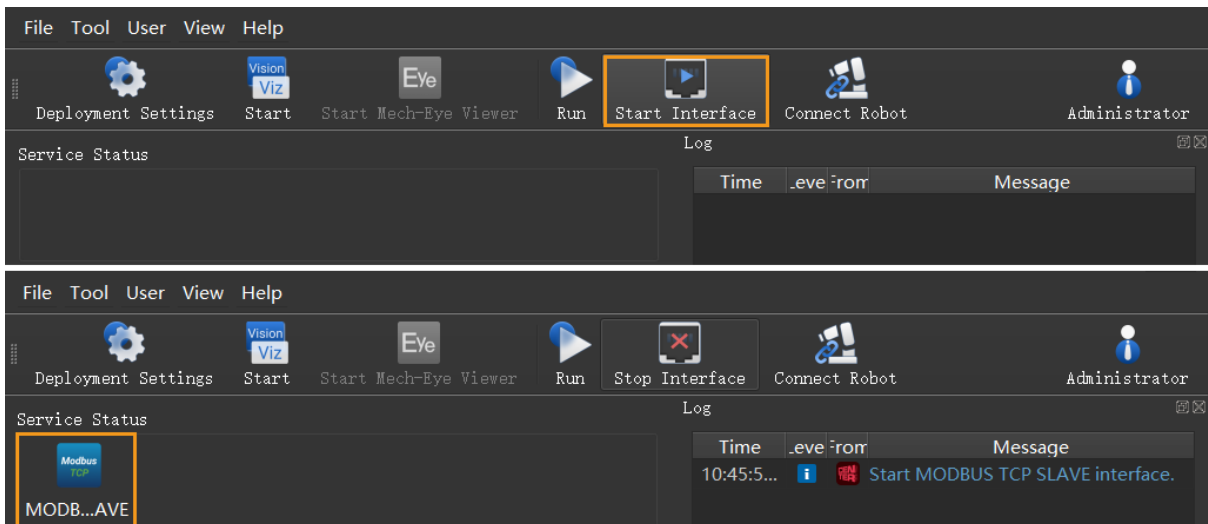
3. Click *Mech-Interface* on the left, and select *Use Mech-Interface* → *Standard Interface* → *Modbus TCP SLAVE*. The slave device IP is 0.0.0.0 (the IPv4 address of the IPC). Enter the slave device address, and select the byte order, and click :guilabel: Save .



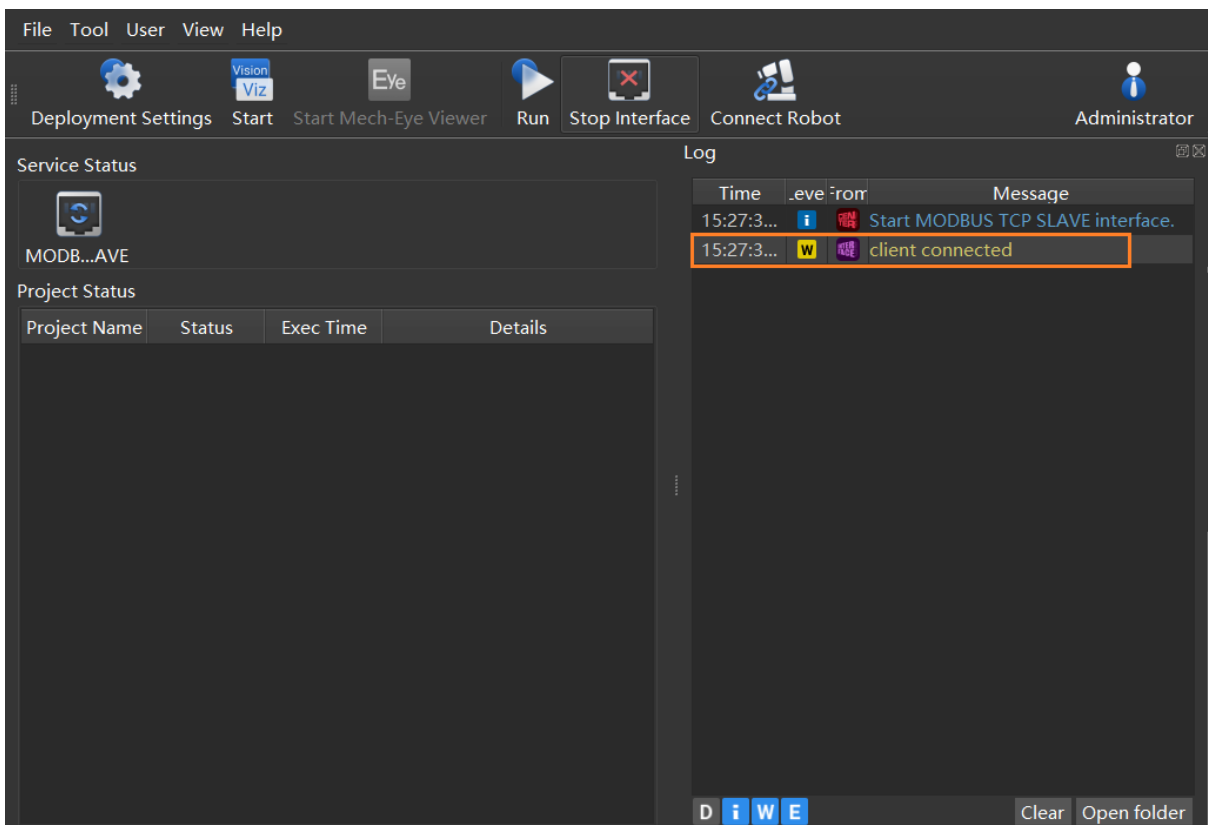
**Tip:** You can set the byte order according to the master's byte order requirements for floating point numbers. For example, for standard big-endian data, select DCBA; for standard little-endian data, select ABCD.

4. In the main window of Mech-Center, click *Start Interface*, and the Modbus TCP SLAVE service icon will appear in the service status bar.





5. If the communication between the Mech-Center and the main station is successfully established, the log section on the right side of the interface will print "Client is connected"; if the communication between the Mech-Center and the main station fails, check whether the physical connection is faulty.



**Register Map**

The register map table for Modbus TCP in Mech-Center is shown below.

Address	Length (in words)	Read/Write	Unit	Holding registers (4x)
0x0000	1	Write	0: Not trigger the command. 1: Trigger the command.	40001–40053: Write into Mech-Center
0x0001	1	Write	Command code	
0x0002	1	Write		
0x0003	1	Write		
0x0004	1	Write		
0x0005	1	Write		
0x0006	12	Write	Unit: degree	
0x0007	12	Write	Units: mm and degree (Euler angles)	
0x0008	1	Write		
0x0009	1	Write		
0x000A	1	Write		40054–40728: Read from Mech-Center
0x000B	1	Write		
0x000C	1	Write		
0x000D	1	Write		
0x000E	6	Write	Unit: mm	
0x000F	12	Write	Units: mm and degree (Euler angles)	
0x0010	1	Write	Only for camera calibration	
0x0011	44	Read		
0x0012	1	Read	0: Not triggered. 1: Triggered.	
0x0013	1	Read		
0x0014	1	Read	1 Hz	
0x0015	1	Read		
0x0016	1	Read	0: Received command 999 (Clear Register Data) 1: The data has newly arrived.	
0x0017	1	Read		
0x0018	1	Read		
0x0019	480	Read		
0x0248	40	Read		
0x0270	40	Read		
0x0298	64	Read		

**Note:** Modbus TCP is a simple bus protocol. It is recommended to read or write about 100 words at a

time. The communication cycle for a single read or write is about 70 ms. When available, choose as much as possible real-time Ethernet protocols such as PROFINET and Ethernet/IP, Siemens Snap7 (Standard Interface Siemens PLC Client in Mech-Center corresponding), or Mitsubishi MELSEC (referred to as MC protocol).

### Holding Register Commands

- *Command 101: Start Mech-Vision Project*
- *Command 102: Get Vision Target(s)*
- *Command 103: Switch Mech-Vision Recipe*
- *Command 201: Start Mech-Viz Project*
- *Command 202: Stop Mech-Viz Project*
- *Command 203: Select Mech-Viz Branch*
- *Command 204: Set Move Index*
- *Command 205: Get Planned Path*
- *Command 206: Get DO List*
- *Command 501: Input Object Dimensions to Mech-Vision*
- *Command 502: Input TCP to Mech-Viz*
- *Command 901: Get Software Status*
- *Command 999: Clear Register Data*

#### Command 101: Start Mech-Vision Project

This command starts the running of the Mech-Vision project, which executes image capturing and performs vision recognition.

If the project works in the eye-in-hand mode, the robot pose for image capturing needs to be transmitted by this command into the project.

This command is for scenarios using only Mech-Vision.

**Command Sent**

Parameter	Address
Command code 101	1
Project number	4
Expected number of vision points	3
Robot pose type	2
Robot pose	6–11 (JPs) or 12–17 (flange pose)

**Project number**

The integer ID number of the Mech-Vision project in Mech-Center, i.e., the number shown on the left of the project path in *Deployment Settings* → *Mech-Vision* in Mech-Center. The number can be adjusted by dragging the projects.

**Expected number of vision points**

The number of vision points (i.e., vision poses and their corresponding point clouds, labels, indices, etc.) to expect Mech-Vision to output.

- 0: Get all the vision points from the Mech-Vision project's recognition results.
- **integers** > 0: Get the specified number of vision points.
  - If the total number of vision points is smaller than the parameter value, all the available vision points will be returned.
  - If the total number of vision points is greater than or equal to the parameter value, vision points in the quantity of the parameter value will be returned.

---

**Note:** The command to obtain the vision points is command 102. Due to the limit that a maximum of 20 vision points can be obtained by executing command 102 at a time, after executing command 102 for the first time, one of the parameters returned will indicate whether all the vision points requested have been returned; if not, please repeat executing command 102.

---

**Robot pose type**

This parameter indicates the type of the image-capturing pose of the real robot to input to Mech-Vision.

- 0: No robot pose needs to be transmitted by this command. If the project works in the eye-to-hand mode, then image capturing has nothing to do with the robot's pose, so no robot image-capturing pose is needed by Mech-Vision.
- 1: The robot pose transmitted by this command is JPs.
- 2: The robot pose transmitted by this command is a flange pose.

**Robot pose**

This parameter is the robot pose needed when the project works in the eye-in-hand mode.

The robot pose is either JPs or flange pose, according to the setting of the parameter **robot pose type**.

Data format: 6 Float type numbers.

**Data Returned**

Parameter	Address
Status code	100

**Status code**

If there is no error, status code 1102 will be returned. Otherwise, the corresponding error code will be returned.

**Command 102: Get Vision Target(s)**

This command gets the vision targets, i.e., robot TCPs to reach the object poses contained in the vision points, after executing command 101.

Mech-Center automatically transforms the object poses in the vision points to their corresponding robot TCPs. The process is as follows:

- Rotate the poses around their Y axes by 180 degrees.
- Determine whether the definition of the reference frame used by the robot model involves robot base height, and add a vertical offset accordingly.

---

**Note:** By default, command 102 can only fetch at most 20 targets at a time. Therefore, command 102 may need to be repeatedly executed until all the targets required are obtained.

---

**Command Sent**

Parameter	Address
Command code 102	1
Project number	4

**Project number**

The integer ID number of the Mech-Vision project in Mech-Center, i.e., the number shown on the left of the project path in *Deployment Settings* → *Mech-Vision* in Mech-Center. The number can be adjusted by dragging the projects.

**Data Returned**

Parameter	Address
Status code	100
Data transmission status	101
Number of targets	102
Reserved field	/
All target poses obtained this time	104
All labels obtained this time	584

**Status code**

If there is no error, status code 1100 will be returned. Otherwise, the corresponding error code will be returned.

After executing this command, if the results from Mech-Vision have not been returned, Mech-Center will wait before sending the results to the robot. The default wait time is 10 seconds. If a timeout occurs, the timeout error status code will be returned.

**Data transmission status**

This parameter indicates whether the data returned is new and can be read.

1: The returned data is new and can be read.

---

**Note:** After the newly returned data is read, reset this parameter to 0.

---

**Number of targets**

The number of targets obtained by executing this command this time.

**Reserved field**

This field is not used.

The value defaults to 0.

**All target poses obtained this time**

The information contained in a single target includes pose coordinates (XYZ) and orientation Euler angles (ABC).

**All labels obtained this time**

The integer label assigned to the pose. If in the Mech-Vision project, the labels are strings, they need to be mapped to integers before outputting from the Mech-Vision project using Step label\_mapping. If there are no labels in the Mech-Vision project, the label defaults to 0.

**Command 103: Switch Mech-Vision Recipe**

This command switches the parameter recipe used by the Mech-Vision project.

In Mech-Vision, you can change the settings of a group of parameters by switching the parameter recipe.

Parameters involved in recipe switching usually include point cloud matching model, image matching template, ROI, confidence threshold, etc.

This command needs to be used before executing command 101 which starts the Mech-Vision project.

**Command Sent**

Parameter	Address
Command code 103	1
Project number	4
Recipe number	5

**Project number**

The integer ID number of the Mech-Vision project in Mech-Center, i.e., the number shown on the left of the project path in *Deployment Settings* → *Mech-Vision* in Mech-Center.

**Recipe number**

The ID number of the parameter recipe to switch to, i.e., the number on the left of the parameter recipe name in *Project Assistance* → *Parameter Recipe* → *Parameter Recipe Editor* in Mech-Vision.

Recipe number range: 1—99.

**Data Returned**

Parameter	Address
Status code	100

**Status code**

If there is no error, status code 1107 will be returned. Otherwise, the corresponding error code will be returned.

**Command 201: Start Mech-Viz Project**

This command is for scenarios using both Mech-Vision and Mech-Viz.

This command starts the running of the Mech-Viz project, calls the corresponding Mech-Vision project, and lets the Mech-Viz project plan the robot path based on the vision points from Mech-Vision.

For the Mech-Viz project that needs to be started, the option *Autoload* needs to be checked in Mech-Viz's interface.

Please see [Example Mech-Viz Projects for Standard Interface](#) for the description of example Mech-Viz projects.

**Command sent**

Parameter	Address
Command code 201	1
Robot pose type	2
Robot pose	6

**Pose type**

0:

- The current pose of the robot is not needed by Mech-Viz and no pose will be sent.
- If the project works in the eye-to-hand mode, no robot image-capturing pose will be needed by the project.
- In Mech-Viz, the simulated robot will move from the initial pose JPs = [0, 0, 0, 0, 0, 0] to the first target in the planned path.

1:

- The robot pose will be sent in the form of JPs.
- In Mech-Viz, the simulated robot will move from the input initial pose (i.e., the pose sent by this command) to the first target in the planned path.

---

**Note:** If in the scene, there are barriers that stand in the way from the initial pose JPs = [0, 0, 0, 0, 0, 0] to the first target in the planned path, the pose type must be set to 1.

---

**Robot pose**

The current JPs of the real robot (if pose type is set to 1).

**Data Returned**

Parameter	Address
Status code	100

**Status code**

If there is no error, status code 2103 will be returned. Otherwise, the corresponding error code will be returned.



### Command 202: Stop Mech-Viz Project

This command stops the running of the Mech-Viz project. This command is needed only when the Mech-Viz project fall into an infinite loop or cannot be stopped normally.

#### Command Sent

Parameter	Address
Command code 202	1

#### Data Returned

Parameter	Address
Status code	100

#### Status code

If there is no error, status code 2104 will be returned. Otherwise, the corresponding error code will be returned.

### Command 203: Select Mech-Viz Branch

This command specifies which branch the project should run along. For this command, the branching is implemented by a branch\_by\_msg Task, and this command selects the branch by specifying an exit port of the Task.

Before executing this command, the Mech-Viz project needs to be started by executing command 201.

When the Mech-Viz project runs to the branch\_by\_msg Task, it will wait for command 203 to specify which exit port of the Task, i.e., the branch, the project should run along.

#### Command Sent

Parameter	Address
Command code 203	1
Branching Task ID	30
Exit port number	31

#### Branching Task ID

The Task ID of the branch\_by\_msg Task.

This parameter is for specifying which branch\_by\_msg Task the branch selection should apply to.

The Task ID can be read in the Task's parameters.

**Exit port number**

This parameter is for specifying which exit port of the specified Task, i.e., the branch, the project should run along. The value should be an integer ([1, N]).

---

**Note:** An exit port number is the 1-based index of the specified exit port on the Task. For example, if the specified exit port is the second exit port of the Task from left to right, the exit port number is 2.

---

**Data Returned**

Parameter	Address
Status code	100

**Status code**

If there is no error, status code 2105 will be returned. Otherwise, the corresponding error code will be returned.

**Command 204: Set Move Index**

This command is for setting the index parameter of a Task that involves sequential or separate motions or operations.

Tasks with index parameters include `move_list`, `move_grid`, `custom_pallet_pattern`, `smart_pallet_pattern`, etc.

Before executing this command, command 201 needs to be executed to start the Mech-Viz project.

**Command Sent**

Parameter	Address
Command code 204	1
Task ID	32
Index value	33

**Task ID**

This parameter specifies which Task the index setting should apply to.

The Task ID can be read in the Task's parameters.

**Index value**

The index value that should be set the next time this Task is executed.

When this command is sent, the current index value in Mech-Viz will become the parameter value minus 1.

When the Mech-Viz project runs to the task specified by this command, the current index value in Mech-Viz will be increased by 1 to become the parameter's value.

### Data Returned

Parameter	Address
Status code	100

### Status code

If there is no error, status code 2106 will be returned. Otherwise, the corresponding error code will be returned.

### Command 205: Get Planned Path

This command gets the robot motion path planned by Mech-Viz after command 201 is executed to start the Mech-Viz project.

---

**Note:** By default, command 205 can only fetch at most 20 targets of the planned path at a time. So, command 205 may need to be executed repeatedly until all the targets required are obtained.

---



---

**Note:** If one of the targets in the path is not supposed to be sent to the robot, please clear the parameter "Send Target" checkbox of the corresponding move-type Task.

---

### Command Sent

Parameter	Address
Command code 205	1
Target type	2

### Target type

This parameter specifies the type of path targets to return from Mech-Viz.

- 1: The targets returned should be in JPs.
- 2: The targets returned should be in TCP.

**Data Returned**

Parameter	Address
Status code	100
Data transmission status	101
Number of points	102
Position of "visual_move"	103
All target poses sent this time	104
All target labels sent this time	584
All target velocities sent this time	624

**Status code**

If there is no error, status code 2100 will be returned. Otherwise, the corresponding error code will be returned.

---

**Note:** When executing this command, if Mech-Viz has not yet had the planned robot motion path (the project is still running), Mech-Center will wait. The default wait time is 10 seconds. If a timeout occurs, a timeout error code will be returned.

---

**Data transmission status**

This parameter indicates whether the data returned is new and can be read.

1: The returned data is new and can be read.

---

**Note:** After the newly returned data is read, reset this parameter to 0.

---

**Number of points**

This parameter indicates the number of path targets ([pose, label, velocity]) sent by executing this command this time.

Default range: 0 to 20.

**Position of "visual\_move"**

The position of the vision\_move Task, i.e., the move to the vision pose (usually the pose for picking the object) in the entire robot motion path.

For example, if the path is composed of Tasks **move\_1**, **move\_2**, **visual\_move**, **move\_3** sequentially, the position of **visual\_move** is 3.

If in the path there is no vision\_move Task, the returned value will be 0.

**Pose**

Cartesian coordinates (XYZ) and Euler angles (ABC), or JPs, according to the pose type set by command 205.

**Label**

Label is the integer label assigned to the pose. If in the Mech-Vision project, the labels are strings, they need to be mapped to integers using Step label\_mapping before outputting from

the Mech-Vision project. If there are no labels in the Mech-Vision project, the label defaults to 0.

### Velocity

The non-zero velocity parameter percentage value for the move-type Task set in Mech-Viz.

### Command 206: Get DO List

This command gets the planned DO signal list when there are multiple grippers, such as suction cup sections, to control.

For using this command:

In the parameters of the "set\_do\_list" Task:

- Check "StandardInterface" under "Receiver"
- Check "Get DO List from VisualMove"
- Select a "visual\_move" Task that needs the DO signal list at the bottom of the parameter panel

Before calling this command, command 205 needs to be executed to obtain the planned motion path by Mech-Viz.

Please deploy the Mech-Viz project based on the template project at `/Mech-Center/tool/viz_project/suction_zone`, and set the suction cup configuration file in the Mech-Viz project.

### Command Sent

Parameter	Address
Command code 206	1

### Data Returned

Parameter	Address
Status code	100
DO signal values	704

### Status code

If there are no errors, status code 2102 will be returned. Otherwise, the corresponding error code will be returned.

### DO signal value

There are 64 DO signal values, in integers.

Range of valid DO values: [0, 999].

Placeholder value: -1.

### Command 501: Input Object Dimensions to Mech-Vision

This command is for dynamically inputting object dimensions into the Mech-Vision project.

Please confirm the actual object dimensions before running the Mech-Vision project.

The Mech-Vision project should have the `read_object_dimensions` Step, and the Step's parameter **Read Object Dimensions from Parameters** should be set to **True**.

#### Command Sent

Parameter	Address
Command code 501	1
Project number	4
[length, width, height]	34

#### Project number

The integer ID number of the Mech-Vision project in Mech-Center, i.e., the number shown on the left of the project path in *Deployment Settings* → *Mech-Vision* in Mech-Center.

#### Length, height, width

The object dimensions to input to the Mech-Vision project.

These values will be read by the `read_object_dimensions` Step.

Unit: mm

#### Data Returned

Parameter	Address
Status code	100

#### Status code

If there is no error, status code 1108 will be returned. Otherwise, the corresponding error code will be returned.

### Command 502: Input TCP to Mech-Viz

This command is for dynamically inputting robot TCP into the Mech-Viz project.

The Task that receives the robot TCP is `outer_move`.

Please deploy the Mech-Viz project based on the template project at `/Mech-Center/tool/viz_project/outer_move`, and put the `outer_move` Task to a proper position in the workflow.

This command needs to be executed before executing command 201.

**Command Sent**

Parameter	Address
Command code 502	1
Robot TCP	40

**Robot TCP**

The TCP data used to set the target for the Task outer\_move.

**Data Returned**

Parameter	Address
Status code	100

**Status code**

If there is no error, status code 2107 will be returned. Otherwise, the corresponding error code will be returned.

**Command 901: Get Software Status**

This command is designed for checking the software running status of Mech-Vision, Mech-Viz, and Mech-Center. At present, this command only supports checking whether Mech-Vision is ready for running the project.

**Command Sent**

Parameter	Address
Command code 901	1

**Data Returned**

Parameter	Address
Status code	100

**Status code**

Software status. 1101 means the Mech-Vision project is ready to run. Other codes mean the project is not ready.

**Command 999: Clear Register Data**

This command is used to clear the data in the registers.

**Command Sent**

Parameter	Address
Command code 999	1

**Data Returned**

Parameter	Address
Status code	100

**Status code**

If there is no error, status code 3103 will be returned. Otherwise, the corresponding error code will be returned.

**3.3.7 Status Codes and Trouble Shooting**
**Overview**

Range	Category
1001—1099	Mech-Vision error codes
1100—1199	Mech-Vision normal status codes
2001—2099	Mech-Viz error codes
2100—2199	Mech-Viz normal status codes
3001—3099	Mech-Center error codes
3100—3199	Mech-Center normal status codes
4000—4099	Robot error codes
4100—4199	Robot normal status codes
7001—7099	Hand-eye calibration error codes
7100—7199	Hand-eye calibration normal status codes



**Mech-Vision**
**Mech-Vision Error Codes**

Code	Meaning
1001	Mech-Vision: Project not registered.
1002	Mech-Vision: No vision result.
1003	Mech-Vision: No point cloud in ROI.
1004	Mech-Vision: Parameter setting failed.
1005	Mech-Vision: Invalid pose type.
1006	Mech-Vision: Invalid pose data.
1007	Mech-Vision: Computing.
1008	Error code not in use.
1009	Mech-Vision: Number of poses and number of motion parameters do not match.
1010	Mech-Vision: Number of poses and number of labels do not match.
1011	Mech-Vision: Project number does not exist.
1012	Mech-Vision: Parameter recipe number out of range.
1013	Mech-Vision: Parameter recipe does not exist.
1014	Mech-Vision: Failed to set parameter recipe.
1015	Mech-Vision: Project runtime error.
1016	Mech-Vision: Failed to start deep learning server.
1017	Mech-Vision: Invalid label mapping.
1018	Mech-Vision: Wrong number of vision points.
1019	Mech-Vision: Execution timed out.
1020	Mech-Vision: Not executed.
1021	Mech-Vision: Failed to set box dimensions; please confirm if Step read_object_dimensions is in the project.
1022	Mech-Vision: Invalid setting values of object dimensions.
1023	Mech-Vision: Failed to connect to camera.
1024	Mech-Vision: Number of poses and number of custom data items do not match.
1025	Mech-Vision: Using virtual camera. Image data has been blocked.

**Mech-Vision Normal Status Codes**

Code	Meaning
1100	Mech-Vision: Successfully obtained vision points.
1101	Mech-Vision: Ready.
1102	Mech-Vision: Successfully triggered project.
1107	Mech-Vision: Successfully switched parameter recipe.
1108	Mech-Vision: Successfully set object dimensions.

**Mech-Viz**
**Mech-Viz Error Codes**

Code	Meaning
2001	Mech-Viz: Project not registered.
2002	Mech-Viz: Project is running.
2003	Mech-Viz: Vision result from Mech-Vision not received.
2004	Mech-Viz: Failed to reach vision point from Mech-Vision.
2005	Mech-Viz: Failed to calculate robot JPs.
2006	Error code not in use.
2007	Mech-Viz: Path planning failed.
2008	Mech-Viz: Project runtime error.
2009	Mech-Viz: TCP not provided.
2010	Mech-Viz: Path not reachable.
2011	Mech-Viz: DO list not provided.
2012	Mech-Viz: Invalid pose type.
2013	Mech-Viz: Invalid pose data.
2014	Mech-Viz: Project not set.
2015	Mech-Viz: Pose of TCP type not supported.
2016	Mech-Viz: Parameter setting failed.
2017	Mech-Viz: Failed to stop execution.
2018	Mech-Viz: Invalid branch_by_msg Task exit number.
2019	Mech-Viz: Failed to set branch_by_msg Task; please confirm whether the Task ID exists.
2020	Mech-Viz: Motion error—singularity.
2021	Mech-Viz: MoveL planning failed.
2022	Mech-Viz: Not executed.
2023	Mech-Viz: Project file error.
2024	Mech-Viz: Invalid branch_by_msg Task ID.
2025	Mech-Viz: Execution timed out.
2026	Mech-Viz: Invalid ID of Task with index parameter.
2027	Mech-Viz: Invalid index value.
2028	Mech-Viz: Index setting failed; please check whether the Task with index parameter exists in the project.
2029	Mech-Viz: Failed to set target of outer_move Task.
2030	Mech-Viz: Invalid vision point.
2031	Mech-Viz: Robot self-collision detected.
2032	Mech-Viz: Collision between robot and scene object detected.
2033	Mech-Viz: Collision detected as point cloud collision point count exceeds threshold.
2034	Mech-Viz: Collision detected as point cloud collision area exceeds threshold.
2035	Mech-Viz: Collision detected as point cloud collision volume exceeds threshold.
2036	Mech-Viz: Vision service did not capture image.
2037	Mech-Viz: No vision result from vision service.
2038	Mech-Viz: No point cloud in ROI in vision result.
2039	Mech-Viz: No vision point for planning.
2040	Mech-Viz: Failed to plan paths for some of vision points from vision result reuse.
2041	Mech-Viz: Failed to get Task parameter.
2042	Mech-Viz: Failed to obtain planning result of outer_move.
2043	Mech-Viz: Failed to get custom data in vision result.

continues on next page

Table 1 – continued from previous page

Code	Meaning
2044	Mech-Viz: Vision service not registered.

**Mech-Viz Normal Status Codes**

Code	Meaning
2100	Mech-Viz: Successfully executed.
2101	Mech-Viz: Successfully stopped execution.
2102	Mech-Viz: Successfully sent DO list.
2103	Mech-Viz: Successfully started.
2104	Mech-Viz: Successfully stopped.
2105	Mech-Viz: Branch successfully set.
2106	Mech-Viz: Index successfully set.
2107	Mech-Viz: Target of outer_move Task successfully set.

**Mech-Center**
**Mech-Center Error Codes**

Code	Meaning
3001	Mech-Center: Illegal command.
3002	Mech-Center: Interface command length or format error.
3003	Mech-Center: Client disconnected.
3004	Mech-Center: Server disconnected.
3005	Mech-Center: Calling Mech-Vision timed out.
3006	Mech-Center: Unknown error.
3007	Mech-Center: Data acknowledge signal timed out.
3008	Mech-Center: Config ID does not exist; failed to read/set Task parameter.

**Mech-Center Normal Status Codes**

Code	Meaning
3100	Mech-Center: Client connection normal.
3101	Mech-Center: Server connection normal.
3102	Mech-Center: Waiting for client to connect.
3103	Mech-Center: Data cache cleared successfully.

**Robot**
**Robot Error Codes**

Code	Meaning
4001	Robot: Invalid robot type.
4002	Robot: Robot Euler angle type not supported.
4003	Robot: Robot service not registered.
4004	Robot: Missing robot parameters.
4005	Robot: Failed to connect to robot. Please check the robot IP address and network configuration.
4006	Robot: The robot burn-in program version is not up to date. Please download the new program.

**Robot Normal Status Codes**

Code	Meaning
4100	Robot: Robot service registered successfully.
4101	Robot: Successfully connected to the robot.
4102	Robot: Robot disconnected.
4103	Robot: User closed robot service.

**Hand-Eye Calibration**
**Hand-Eye Calibration Error Codes**

Code	Meaning
7001	Calibration: Parameter error.
7002	Calibration: Mech-Vision did not output calibration point.
7003	Calibration: Robot failed to reach calibration point.

**Hand-Eye Calibration Normal Status Codes**

Code	Meaning
7100	Calibration: Robot successfully reached calibration point.
7101	Calibration: Mech-Vision normally output calibration point.

---

## Mech-Vision Trouble Shooting

### 1001

Mech-Vision: Project not registered.

Reason:

- The project is not open in Mech-Vision.
- *Autoload Project* has not been set for the Mech-Vision project.

Solution:

- Make sure the project is open in Mech-Vision.
  - Make sure *Autoload Project* has been set for the Mech-Vision project.
- 

### 1002

Mech-Vision: No vision result.

Reason:

- The Mech-Vision project called has been successfully executed but has empty output. Possible reasons for empty output include exceedingly high confidence threshold for instance segmentation, no matched object in the scene, improper ROI settings, low point cloud quality, improper filtering settings, etc.
- Command 102 has been called to obtain all poses in the vision result, and the cache is empty, but the user continues to call command 102.

Solution:

- Check the data flow of the Mech-Vision project by going up from the PoseList port of Step "Procedure Out".
  - Check the client interface program. If the parameter "pose transmission completion status" in the returned data of command 102 is 1, all poses have been transmitted, and continuing calling command 102 will result in this error.
    - Returned data format of command 102: `102, status code, pose transmission completion status, number of poses, reserved field, [pose, label, velocity]`.
-

**1003**

Mech-Vision: No point cloud in ROI.

Reason:

- The Mech-Vision project called has been successfully executed, but there is no point cloud in the 3D ROI.

Solution:

- Please check the ROI settings in the Step that sets the 3D ROI on the point cloud. For some projects, this error code can be used to determine whether the bin has been emptied or whether the bin has not been moved to a proper position. So, this error is not necessarily a project execution error.
- 

**1004**

Mech-Vision: Parameter setting failed.

---

**1005**

Mech-Vision: Invalid pose type.

Reason:

- The parameter that sets the pose type in command 101 has an illegal value.
  - Format of command 101: 101, project number, expected number of vision points, pose type, robot pose

Values of the parameter "pose type":

- ✦ 0: no image-capturing robot pose is needed (such as in the eye-to-hand mode).
- ✦ 1: the image-capturing robot pose is in JPs.
- ✦ 2: the image-capturing robot pose is a flange pose.

Solution:

- Check the client interface program. The value of the parameter "pose type" should be in the range of [0, 2].
-

**1006**

Mech-Vision: Invalid pose data.

Reason:

- The pose data sent by command 101 does not consist of 6 numbers. The robot pose is a 6-axis robot pose by default. If the robot is a 4-axis or a 5-axis robot, please fill the remaining fields with 0.
  - Format of command 101: 101, project number, expected number of vision points, pose type, robot pose

Solution:

- Check the client interface program. The robot pose data sent by command 101 should consist of 6 numbers.
- 

**1007**

Mech-Vision: Computing.

Reason:

- When the Mech-Vision project is still running, the client interface program calls command 101 again to try to trigger the same Mech-Vision project.
- Mech-Vision allows concurrent running of multiple projects, but one Mech-Vision project cannot be started again when it is still running.

Solution:

- Check the client interface program and make sure the project number sent by command 101 is correct.
  - Check the client interface program and see if the program triggers the same Mech-Vision project twice in a short time.
- 

**1008**

Error code not in use.

---

**1009**

Mech-Vision: Number of poses and number of motion parameters do not match.

Reason:

- This error usually occurs in projects in which the vision result forms a robot motion path (such as gluing applications).

Solution:

Not available yet.

---

**1010**

Mech-Vision: Number of poses and number of labels do not match.

Reason:

- The number of poses and number of labels output from the preceding Steps of Step "Procedure Out" do not match.

Solution:

- Check the poses and labels in the data flow of the Mech-Vision project.
- 

**1011**

Mech-Vision: Project number does not exist.

Reason:

- The project number set in the parameter of command 101 does not exist in the deployment settings of Mech-Center.
  - For example, when there is only one project's path in the deployment settings, if command 101 calls for project No. 2, the error will be raised.

Solution:

- Check if the corresponding Mech-Vision projects all have *Autoload Project* checked.
  - Check the client interface program and make sure command 101 has the correct project number in its parameter.
-



**1012**

Mech-Vision: Parameter recipe number out of range.

Reason:

- The parameter recipe number in command 103 does not have the corresponding recipe set in the Mech-Vision project.
  - For example, when there are only two parameter recipes in the Mech-Vision project, if command 103 is to set parameter recipe No. 3, this error will be raised.

Solution:

- Check if the Mech-Vision project has the parameter recipe to be set.
  - Check if the client interface program correctly calls command 103.
- 

**1013**

Mech-Vision: Parameter recipe not set.

Reason:

- The client interface program calls command 103 to set the parameter recipe, but there is no parameter recipe set in the Mech-Vision project.

Solution:

- Check the parameter recipe settings in the Mech-Vision project, and ensure the parameter recipes and their numbers are correct.
  - If the project does not require switching parameter recipes, please do not call command 103.
- 

**1014**

Mech-Vision: Failed to set parameter recipe.

Reason:

- The connection between Mech-Center and Mech-Vision is not established.

Solution:

- Please restart Mech-Center and Mech-Vision.
-

**1015**

Mech-Vision: Project runtime error.

Reason:

- The Mech-Vision project has runtime error with error code CV-Exxxx or has other errors for which Mech-Center does not have analyses. When such an error occurs, the Mech-Vision project is terminated without finishing execution.

Solution:

- Check the error message in Mech-Vision and modify the Mech-Vision project if necessary.
- 

**1016**

Mech-Vision: Failed to start deep learning server.

Reason:

- The called Mech-Vision project has a deep learning Step(s), but the deep learning server has not been started. The corresponding error message in Mech-Vision is DL-E0201 deep learning server not started.

Solution:

- Check if the Mech-Vision project is executed for the first time. Check if the time cost for starting the deep learning server is too long. If model pre-loading is required, please check "Preload Mode When Opening Project" in the parameters of the deep learning Step.
  - Check if the deep learning environment has been installed properly.
- 

**1017**

Mech-Vision: Invalid label mapping.

Reason:

- The label(s) in the vision result from Mech-Vision is not INT data. When using the standard interface, string labels should be mapped to integer labels using Step "Label Mapping", otherwise this error will occur.

Solution:

- Check the data flow in Mech-Vision, if the labels input to Step "Procedure Out" are not integers, please map the labels to positive integers using Step "Label Mapping".
-

**1018**

Mech-Vision: Wrong number of vision points.

Reason:

- The client interface program calls command 101 to trigger the Mech-Vision project. In the command's parameter, the number of vision points to expect Mech-Vision to output exceeds the data length of the interface.
  - Format of command 101: `101, project number, expected number of vision points, pose type, robot pose`
- The data length can be set in the deployment settings of Mech-Center. Please set the max number of poses sent each time under *Mech-Interface* -> *Advanced Settings*. Range: [1, 30].

Solution:

- Check the client interface program and make sure the parameter "number of poses" is smaller than the setting in Mech-Center.
- 

**1019**

Mech-Vision: Execution timed out.

Reason:

- After calling command 102 to obtain the vision result and the timing has started, the Mech-Vision project did not finish execution within the specified time.
- The timeout period can be set under *Mech-Interface* -> *Advanced Settings* in the deployment settings of Mech-Center. The default timeout period is 10 seconds.

Solution:

- Check the client interface program. A delay operation can be added before calling command 102.
  - For a Mech-Vision project that takes a long time to execute, the timeout period can be set longer in the deployment settings.
- 

**1020**

Mech-Vision: Not executed.

Reason:

- The client interface program did not call command 101 to start the Mech-Vision project but went on to call command 103 to try to get the vision result.
  - For example, if two Mech-Vision projects are registered in Mech-Center, if the client interface program starts project 1, but then tries to get the vision result of project 2, this error will occur.

Solution:

- Check the client interface program and make sure the project number specified by command 102 is correct.
- 

### 1021

Mech-Vision: Failed to set object dimensions; please confirm if Step read\_object\_dimensions is in the project.

Reason:

- The client interface program calls command 501 to try to input object dimensions to the Mech-Vision project, but the Mech-Vision project does not have a "Read Object Dimensions" Step.

Solution:

- Check the Mech-Vision project and make sure it has a "Read Object Dimensions" Step.
- 

### 1022

Mech-Vision: Invalid setting values of object dimensions.

Reason:

- The client interface program calls command 501 to try to input object dimensions to the Mech-Vision project, but the dimension values are invalid (zero or negative values).

Solution:

- Check the client interface program and make sure command 501 passes valid dimension values (positive real numbers).
- 

### 1023

Mech-Vision: Failed to connect to camera.

Reason:

- The client interface program makes the Mech-Vision project trigger image-capturing, but the camera is disconnected. The corresponding error message in Mech-Vision is CV-E0201.

Solution:

- Check the network connection.
- Check if the IP addresses of the camera and the IPC are under the same network segment  
*(The above two can be done by typing 'ping' followed by the camera's IP address in the command prompt.)*
- Check the camera power supply, IPC firewall settings, etc.

- If you cannot locate the issue, please contact our support team.
- 

## 1024

Mech-Vision: Number of poses and number of custom data items do not match.

Reason:

- After the client interface program calls command 110, in the returned data, the number of poses and number of items of a custom data type do not match. A custom data type refers to data input to a port other than PoseList and LabelList of Step "Procedure Out" in the Mech-Vision project. Possible cases include:
  - Empty data at the custom data port.
  - List length at the custom data port does not equal the pose list length.
- Suppose the number of poses input to Step "Procedure Out" is  $N$ . It is required that any custom data port (data other than poses and labels) either has **1** item or  **$N$**  item. Otherwise, this error will be raised.
  - For example, if the custom port data is the number of objects obtained by instance segmentation, only **1** data item is required, and each time the command is executed, the same value will be returned; if the custom data is the box dimensions,  **$N$**  data items are required, and each time the command is executed, the "box dimensions" data corresponding to the pose will be returned.

Solution:

- Please check the data flow in the Mech-Vision project.
- 

## 1025

Mech-Vision: Using virtual camera. Image data has been blocked.

---

### Mech-Viz Trouble Shooting

#### 2001

Mech-Viz: Project not registered.

Reason:

- The project is not open in Mech-Viz.
- The project specified by the command 201 called does not have *Autoload* enabled.

Solution:

- Make sure the project is open in Mech-Viz.
  - Make sure *Autoload* has been enabled for the Mech-Viz project.
- 

## 2002

Mech-Viz: Project is running.

Reason:

- When a Mech-Viz project is still running, command 201 is called to trigger the Mech-Viz project.

Solution:

- Make sure the client interface program does not call twice command 201 to trigger the same Mech-Viz project within a short time.
- 

## 2003

Mech-Viz: Vision result from Mech-Vision not received.

Reason:

- Task "check\_look" does not find any vision result and takes the 2nd exit ("no result"), and the project ends.

Solution:

- If necessary, add additional Tasks to the "no result" exit of Task "check\_look".
  - Check if the workobjects have all been picked away.
  - Check the Mech-Vision project to see if settings such as ROI settings, instance segmentation confidence threshold, etc. are proper.
- 

## 2004

Mech-Viz: Failed to reach vision point from Mech-Vision.

Reason:

- The pose in the vision point from vision service is out of the reachable range of the robot.

Solution:

- Check if the point cloud position in Mech-Viz is normal.
  - Check if the camera extrinsic parameters are normal.
  - Check if the robot TCP settings in Mech-Viz are normal and whether any workobject is out of the reachable range of the robot.
-

**2005**

Mech-Viz: Failed to calculate robot JPs.

Reason:

- Mech-Viz failed to calculate robot JPs for a target.

Solution:

Not available yet.

---

**2006**

Error code not in use

---

**2007**

Mech-Viz: Path planning failed.

Reason:

- Mech-Viz failed to plan a path for any of the vision points in the vision result, and not all reasons for plan failures of the vision points are the same.

*(If the planning failed on the vision points for the same reason, the corresponding error code for such a reason will be raised)*

- For example, if there are 40 vision points in the vision result, for 20 of which the planning failed because the robot cannot reach the vision point, and for 20 of which the planning failed because collisions were detected, this error will be raised.

Solution:

- Check the plan history in Mech-Viz.
- 

**2008**

Mech-Viz: Project runtime error.

Reason:

- An error occurred when the Mech-Viz project was running and the project has been terminated. The error may be reported by a Mech-Viz error code MP-Exxxx or may not yet have an analysis.

Solution:

- Check the pop-up window or logs of Mech-Viz.
-

**2009**

Mech-Viz: TCP not provided.

Reason:

- The client interface program calls command 205 to obtain the planned path from the robot, and requires that the targets should be robot TCPs, but the returned data does not contain robot TCPs.

Solution:

- In Mech-Viz, check the "Others" panel and make sure the option "Send TCP" is checked.
- 

**2010**

Mech-Viz: Path not reachable.

Reason:

- An error with Mech-Viz error code MP-E0007 occurred when running the Mech-Viz project.

Solution:

Not available yet.

---

**2011**

Mech-Viz: DO list not provided.

Reason:

- The client program calls command 206 to obtain the DO signal list that controls the tool (sectioned suction cups, array gripper, etc.), but in the Mech-Viz project, no DO signal list has been configured.

Solution:

- Check whether there is a set\_do\_list Task that follows the vision\_move Task. In the parameters of the set\_do\_list Task, under "Receiver", "StandardInterface" should be checked.
- 

**2012**

Mech-Viz: Invalid pose type.

Reason:

- The client interface program calls command 201 to trigger the Mech-Viz project, the value of the parameter "pose type" is illegal.
  - Format of command 201: 201, pose type, robot pose.
  - Values of parameter "pose type":



- ✧ 0: no image-capturing pose is required (such as in the eye-to-hand mode).
- ✧ 1: the robot pose is in JPs.
- The client interface program calls command 205 to obtain the planned path, the value of the parameter “expected pose type” is illegal.
  - Format of command 205: 205, **expected pose type**.
  - Values of parameter “expected pose type”:
    - ✧ 1: the returned pose type is expected to be JPs.
    - ✧ 2: the returned pose type is expected to be TCP.

Solution:

- Check the command sent by the client interface program.
- 

## 2013

Mech-Viz: Invalid pose data.

Reason:

- The client interface program calls command 201 to trigger the Mech-Viz project, the robot pose data sent is not in the format of 6 numbers. The robot pose is a 6-axis robot pose by default. If the robot is a 4-axis or a 5-axis robot, please fill the remaining fields with 0.
  - Format of command 201: 201, **pose type**, **robot pose**.

Solution:

- Check the robot pose data in command 201 sent by the client interface program. If the Mech-Viz project does not require inputting robot pose data, please set the parameter “pose type” to 0.
- 

## 2014

Mech-Viz: Project not set.

Reason:

- The project is not open in Mech-Viz.
- The Mech-Viz project does not have *Autoload* checked.

Solution:

- Make sure the project is open in Mech-Viz and has *Autoload* checked.
-

**2015**

Mech-Viz: Pose of TCP type not supported.

Reason:

- The client interface program calls command 201 to trigger the Mech-Viz project, the pose type set is TCP, but currently Mech-Viz does not support inputting robot pose as TCP.

Solution:

- Make sure the parameter "pose type" in command 201 sent by the client interface program is set to either 0 or 1.
    - Format of command 201: `201, pose type, robot pose.`
    - Values of parameter "pose type":
      - ✦ 0: no image-capturing pose is required (such as in the eye-to-hand mode).
      - ✦ 1: the robot pose is in JPs.
- 

**2016**

Mech-Viz: Parameter setting failed.

Reason:

- An error occurred when the client interface program called command 208 to set a parameter of a Task in the Mech-Viz project.

Solution:

- Check if the Task ID and parameter key name in the configuration file are correct.
  - The configuration file (*Property Config*) can be found under *Deployment Settings* -> *Mech-Interface* -> *Advanced Settings* in Mech-Center.
- 

**2017**

Mech-Viz: Failed to stop execution.

Reason:

- The client interface program calls command 202 to stop the running of the Mech-Viz project, but the project does not stop within 5 seconds.

Solution:

Not available yet.

---

**2018**

Mech-Viz: Invalid branch\_by\_msg Task exit number.

Reason:

- The client interface program calls command 203 to set the exit of the branching Task in the Mech-Viz project. In the command, the exit number is less than or equal to 0 or is greater than the total number of exits of the Task.
  - Format of command 203: 203, **branching Task ID**, **exit number**

Solution:

- Check the exits of the branching Task in the Mech-Viz project.
  - Check the exit number set in command 203.
- 

**2019**

Mech-Viz: Failed to set branch\_by\_msg Task; please confirm whether the Task ID exists.

Reason:

- The client interface program calls command 203 to set a branch in Mech-Viz, but the branching Task ID set in the command either does not exist in the Mech-Viz project or is not a positive integer.
- The ID of a Task in Mech-Viz can be found and set in the Task's parameters.
  - Format of command 203: 203, **branching Task ID**, **exit number**

Solution:

- Make sure the Task ID sent by the command has a corresponding Task in Mech-Viz.
  - Make sure the Task IDs set in the command and in the Mech-Viz projects are positive integers.
- 

**2020**

Mech-Viz: Motion error—singularity.

Reason:

- When Mech-Viz is planning the robot path, a robot singularity is encountered.

Solution:

- Check where the singularity is in the Mech-Viz project and modify the corresponding targets.
-

**2021**

Mech-Viz: MoveL planning failed.

Reason:

- Mech-Viz found that the robot cannot reach a target by linear motion.

Solution:

- Modify the target or add intermediate points.
  - Change the motion type to joint motion.
- 

**2022**

Mech-Viz: Not executed.

Reason:

- The client interface program calls command 203 to set branch, but the Mech-Viz project is not running.
- The client interface program calls command 205 to get the planned path but the Mech-Viz project has not been executed.
- The client interface program calls command 205 to get the planned path but the Mech-Viz project does not have an output.

Solution:

- Check the client interface program to make sure that the Mech-Viz project is running when setting the branch.
  - Check the client interface program to make sure that the Mech-Viz project is executed before getting the planned path.
- 

**2023**

Mech-Viz: Project file error.

---

**2024**

Mech-Viz: Invalid branch\_by\_msg Task ID.

Reason:

- The client interface program calls command 203 and sets a Task ID that is not a positive integer.
  - Format of command 203: `203, branching Task ID, exit number`

Solution:

- Make sure the parameter "branching Task ID" of command 203 is a positive integer.
  - The ID of a Task in Mech-Viz can be found and set in the Task's parameters.
- 

**2025**

Mech-Viz: Execution timed out.

Reason:

- The client interface program calls command 205 to get the planned path from Mech-Viz, but the project execution does not finish within the specified time (10 seconds by default).

Solution:

- Check the normal project execution time cost of the Mech-Viz project. If the time cost is longer than 10 seconds, please adjust the settings under *Mech-Interface -> Advanced Settings* in the deployment settings of Mech-Center.
  - In the client interface program, a delay can be inserted in the client interface program before calling command 205.
- 

**2026**

Mech-Viz: Invalid ID of Task with index parameter.

Reason:

- The client interface program calls command 204 to set the index parameter of a Task with index parameter, and the Task ID is not a positive integer.
  - Format of command 204: `204, Task ID, index value`

Solution:

- Check the client interface program and make sure that the Task ID set in command 204 is a positive integer.
  - The ID of a Task in Mech-Viz can be found and set in the Task's parameters.
-

**2027**

Mech-Viz: Invalid index value.

Reason:

- The client interface program calls command 204 to set the index parameter of a Task with an index parameter, and the index value is not a positive integer.
  - Format of command 204: `204, Task ID, index value`

Solution:

- Check the client interface program and make sure that the index value set in command 204 is a positive integer.
- 

**2028**

Mech-Viz: Index setting failed; please check whether the Task with index parameter exists in the project.

Reason:

- The client interface program calls command 204 to set the index parameter of a Task with index parameter, and the Task ID does not have a corresponding Task in the Mech-Viz project.
  - Format of command 204: `204, Task ID, index value`

Solution:

- Check the client interface program and make sure the Task ID set in command 204 has a corresponding Task in the Mech-Viz project. A Task's ID can be found and set in its parameters in Mech-Viz.
- 

**2029**

Mech-Viz: Failed to set target of outer\_move Task.

---

**2030**

Mech-Viz: Invalid vision point.

Reason:

- Unknown error. Mech-Viz error code: MP-E0010

Solution:

Not available yet.

---

**2031**

Mech-Viz: Robot self-collision detected.

Reason:

- Mech-Viz detected robot self-collision and the path planning failed.

Solution:

- Check the corresponding target in the Mech-Viz project.
- 

**2032**

Mech-Viz: Collision between robot and scene object detected.

Reason:

- Mech-Viz detected collision between the robot and scene object model(s) and the path planning failed.

Solution:

- Check the corresponding target and scene object(s) in the Mech-Viz project.
- 

**2033**

Mech-Viz: Collision detected as point cloud collision point count exceeds threshold.

Reason:

- Mech-Viz detected collision between the robot and workobject. The number of point cloud points involved in the collision is above the threshold. The path planning failed.

Solution:

- Check the corresponding target in the Mech-Viz project.
  - Check the setting of the collision point count threshold and modify it if necessary.
- 

**2034**

Mech-Viz: Collision detected as point cloud collision area exceeds threshold.

Reason:

- Mech-Viz detected collision between the robot and workobject. The surface area involved in the collision is above the threshold. The path planning failed.

Solution:

- Check the corresponding target in the Mech-Viz project.
  - Check the setting of the collision area threshold and modify it if necessary.
- 

## 2035

Mech-Viz: Collision detected as point cloud collision volume exceeds threshold.

Reason:

- Mech-Viz detected collision between the robot and workobject. The volume involved in the collision is above the threshold. The path planning failed.

Solution:

- Check the corresponding target in the Mech-Viz project.
  - Check the setting of the collision volume threshold and modify it if necessary.
- 

## 2036

Mech-Viz: Vision service did not capture image.

Reason:

- In the Mech-Viz project, the "check\_look" Task takes the 4th exit ("(vision service) not called"), and the exit is not connected to any further Tasks. The Mech-Viz project terminates.

Solution:

- Check whether the vision services selected in "visual\_look" and "check\_look" are consistent.
- 

## 2037

Mech-Viz: No vision result from vision service.

Reason:

- In the Mech-Viz project, the "check\_look" Task takes the 2nd exit ("no result"), and the exit is not connected to any further Tasks. The Mech-Viz project terminates. The corresponding Mech-Vision project does not have a vision result output.

Solution:

- Please see the solution of [1002](#).
-



**2038**

Mech-Viz: No point cloud in ROI in vision result.

Reason:

- In the Mech-Viz project, the “check\_look” Task takes the 5th exit (“no point cloud in ROI”), and the exit is not connected to any further Tasks. The Mech-Viz project terminates. The corresponding Mech-Vision project does not have a vision result output.

Solution:

- Please see the solution of [1002](#).
- 

**2039**

Mech-Viz: No vision point for planning.

Reason:

- In the Mech-Viz project, the Task vision\_move does not have any vision point for planning. If in the project there is no “check\_look” Task or the “check\_look” Task takes the 2nd exit (“no result”) and the exit is not connected to any further Tasks, this error will occur.

Solution:

- Please see the solution of [1002](#).
  - Check if the “check\_look” Task is used correctly in the Mech-Viz project.
- 

**2040**

Mech-Viz: Failed to plan paths for some of vision points from vision result reuse.

Reason:

- In the Mech-Viz project, reusing the vision result is enabled (in the parameters of Task “visual\_look”). The same vision result is used for multiple plannings, and some plannings failed.
- When this error occurs, the successful planning will still be output.

Solution:

- Check the plan history of Mech-Viz and locate why some planning failed.
-

**2041**

Mech-Viz: Failed to get Task parameter.

Reason:

- The client interface program calls command 207 to read the parameter value of a Task in Mech-Viz but encounters an error.

Solution:

- Check if the content of the configuration file is correct. The configuration file (*Property Config*) can be found under *Deployment Settings -> Mech-Interface -> Advanced Settings* in Mech-Center.
    - In particular, check if the Task ID and parameter key name are correct.
- 

**2042**

Mech-Viz: Failed to get planning result of vision\_move.

Reason:

- The client interface program calls command 210 to get the planning result of vision\_move from Mech-Viz but encounters an error.

Solution:

Not available yet.

---

**2043**

Mech-Viz: Failed to get custom data in vision result.

Reason:

- The client interface program calls command 210 to get the planning result of vision\_move and custom data in the vision result from Mech-Viz but encounters an error in getting the custom data part.

Solution:

- Please see the solution of [1024](#).
-

**2044**

Mech-Viz: Vision service not registered.

Reason:

- In the Mech-Viz project, the “visual\_look” Task does not have a vision service selected.

Solution:

- Please select a correct vision service in the “visual\_look” Task.
- 

**Mech-Center Trouble Shooting****3001**

Mech-Center: Illegal command.

Reason:

- The format of the command sent by the client interface program is not supported.

Solution:

- Please check the client interface program.
- 

**3002**

Mech-Center: Interface command length or format error.

Reason:

- The length of the command sent by the client interface program is illegal. For example, the robot pose data sent in the command does not consist of six numbers.
- The format of the command sent by the client interface program is illegal. For example, commas are used as separators.

Solution:

- Please check the client interface program.
-

**3003**

Mech-Center: Client disconnected.

---

**3004**

Mech-Center: Server disconnected.

---

**3005**

Mech-Center: Calling Mech-Vision timed out.

---

**3006**

Mech-Center: Unknown error.

---

**3007**

Mech-Center: Data acknowledge signal timed out.

Reason:

When using PROFINET/EthernetIP:

- When sending the pose data to the port, Mech-Center needs to check that the signal "data\_acknowledge" is 0. If in the specified time, the client interface program does not reset the signal "data\_acknowledge" to 0, this error will be raised.
- After sending the pose data to the port, Mech-Center needs to check that the signal "data\_acknowledge" is 1, indicating that the data has been read. If in the specified time, the client interface program does not set the signal "data\_acknowledge" to 1, this error will be raised.

Solution:

- Check the client interface program and make sure that the signal "data\_acknowledge" is 0 before calling command 102 or command 205.
  - Check the client interface program and make sure that the signal "data\_acknowledge" is set to 1 immediately after reading the data from Mech-Center.
-

**3008**

Mech-Center: Config ID does not exist; failed to read/set Task parameter.

Reason:

- The client interface program calls command 207 to read a Task parameter value or command 208 to set a Task parameter value, but the corresponding content is not found in the configuration file.
- The configuration file (*Property Config*) can be found under *Deployment Settings* -> *Mech-Interface* -> *Advanced Settings* in Mech-Center.

Solution:

- Check the configuration file and make sure the config ID is a positive integer, and the content is correct.
- 

**Hand-Eye Calibration Trouble Shooting****7001**

Calibration: Parameter error.

Reason:

- The client interface program sends the robot pose while performing calibration, but the robot pose does not consist of six numbers. The robot pose is a 6-axis robot pose by default. If the robot is a 4-axis or a 5-axis robot, please fill the remaining fields with 0.

Solution:

- Check the robot pose data sent by the client interface program
- 

**7002**

Calibration: Mech-Vision did not output calibration point.

Reason:

- During calibration, Mech-Vision does not send the robot pose to Mech-Center.

Solution:

Not available yet.

---

7003

Calibration: Robot failed to reach calibration point.

### 3.3.8 Appendix

- *Use Mech-Viz for Collision Detection*
- *Use Mech-Viz for Controlling Suction Cup Sections or Array Gripper*
- *Map String Labels to Integer Labels in Mech-Vision*
- *Flowchart of Calibration*
- *Add Signal for Exposure Completion in Mech-Vision*

#### Use Mech-Viz for Collision Detection

If collision detection is required, please build the Mech-Viz project by referring to the sample project at *Mech-Center/tool/viz\_project/check\_collision*.

Please note:

1. **check\_collision** is only a sample project. In the project, except for move Tasks, the Tasks in the workflow are not supposed to be deleted or modified in their positions in the workflow.
2. Please select the actual robot model in use for the project.
3. The move Tasks can be deleted, added, or modified according to the actual needs.

#### Use Mech-Viz for Controlling Suction Cup Sections or Array Gripper

If controlling multiple suction cup sections or an array gripper, please build the Mech-Viz project by referring to the sample project at *Mech-Center/tool/viz\_project/suction\_zone*.

Please note:

1. **suction\_zone** is only a sample project. In the project, except for move Tasks, the Tasks in the workflow are not supposed to be deleted or modified in their positions in the workflow.
2. Please select the actual robot model in use for the project.
3. The move Tasks can be deleted, added, or modified according to the actual needs.
4. Please configure the suction cup file in the project.
5. DO list can only be obtained after image capturing.

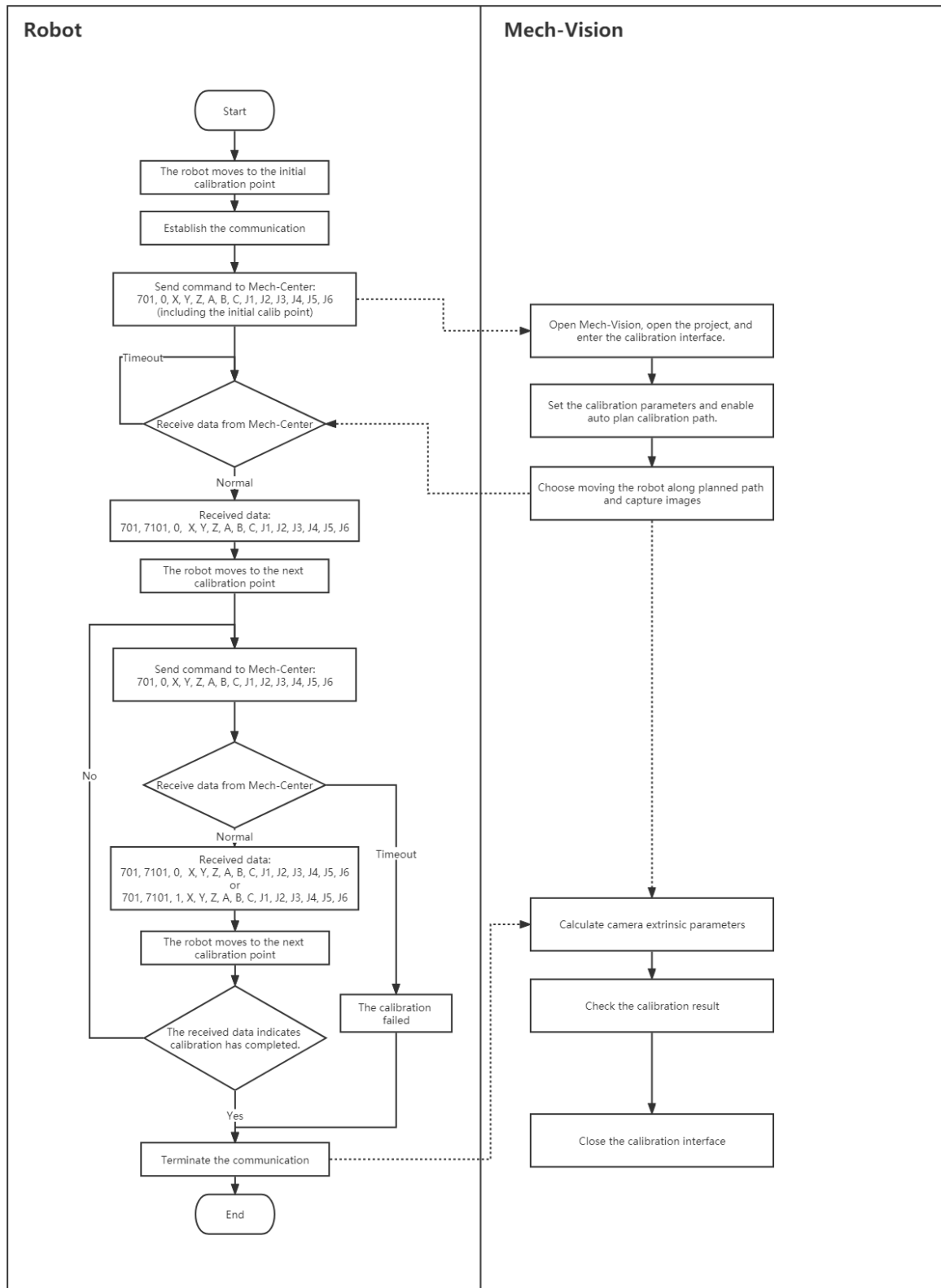
### Map String Labels to Integer Labels in Mech-Vision

The Step to map string labels to integer labels in Mech-Vision is `label_mapping`.

Label mapping can be done by configuring a mapping file in the Step's parameters. A sample label mapping file is as follows:

```
{  
  "Medium", "3",  
  "Large", "2",  
  "Small", "1"  
}
```

Flowchart of Calibration





**Note:**

1. The robot pose sent and received can be either a flange pose ([X, Y, Z, A, B, C]) or JPs ([J1, J2, J3, J4, J5, J6]).
  2. The last point sent from Mech-Vision is the initial calibration point input to Mech-Vision at first, telling the robot to move back to the initial position, and no image capturing will be performed for the last point sent.
  3. In the data sent from Mech-Vision, the third argument (value: 0 or 1) indicates whether the calibration has finished. 0 means not finished and the next calibration point is on the way; 1 means the calibration has finished.
- 

**Add Signal for Exposure Completion in Mech-Vision**

For Profinet and Ethernet/IP, a camera Exposure Complete signal can be used to shorten the system cycle time.

When the Mech-Vision project takes a long time to run, the system cycle time can be shortened by moving the robot immediately after camera exposure.

In the Mech-Vision project, please make the following modification to implement the Camera Exposure Complete signal:

1. Add a notify\_vision Step, and connect it to the control flow port of the capture\_images\_from\_camera Step.
2. For the capture\_images\_from\_camera Step, set the parameter **Trigger Control Flow When Output** to True.
3. Name the notify\_vision Step to "Standard Interface Notify", and set the message content to "1001". Please do not change the content later.

After the settings above, when the camera finishes exposure, an Exposure Complete signal will be sent, and please reset the signal using Exposure Complete Reset after receiving it.

If Mech-Center does not receive the reset signal for over 10 seconds, it will raise an error message: **Mech-Center data confirmation signal timeout**.

## 3.4 Adapter

When the Standard Interface cannot meet complex actual requirements, a customized Adapter program can be created to control the project.

This chapter includes the following sections.

- [Quick Facts of Adapter](#)
- [Adapter Generator Guide](#)
- [Adapter Programming Guide](#)
- [Adapter Programming Examples](#)

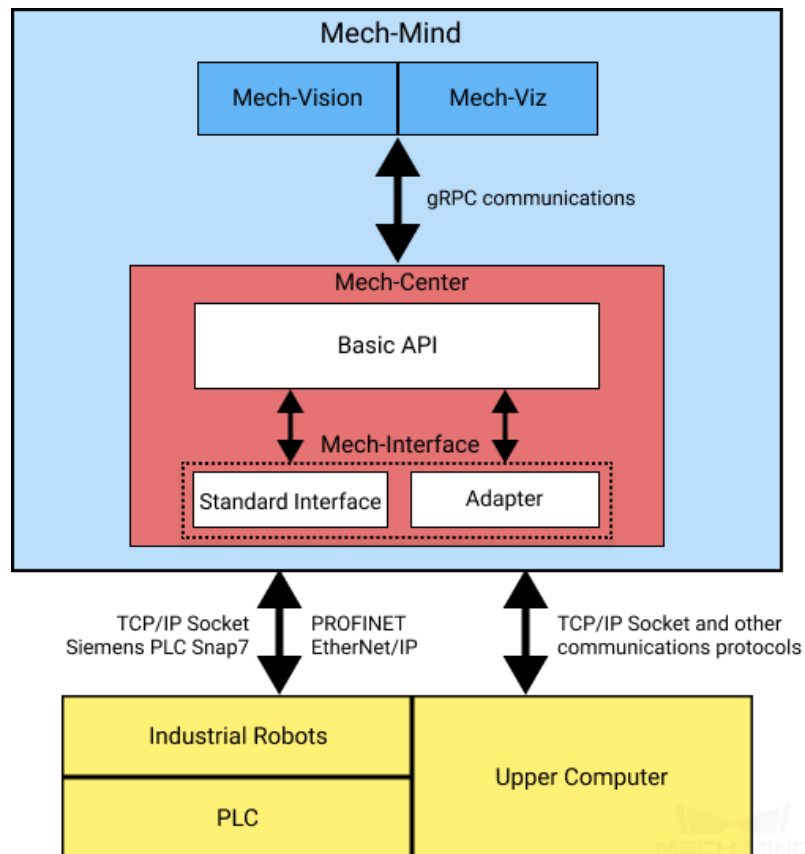
**Tip:** If an Adapter project needs an extra Python library, please install it to the “python” directory of the installation path of Mech-Center. A Python library can be installed as follows:

1. Open the **Command Prompt** or **PowerShell** program.
2. Switch to the “python” directory of the Mech-Center software, such as C:\Mech-Mind\Mech-Center-1.6.x\python.
3. Execute the following command: `./python -m pip install library_name`.

### 3.4.1 Quick Facts of Adapter

#### Introduction

Adapter is an integrated component used for communication in Mech-Center. It employs gRPC communication with Mech-Vision and Mech-Viz via Basic API interfaces, and enables communication with external devices over common industrial protocols, such as TCP/IP Socket, HTTP, and Mitsubishi PLC MC Protocol.



Please refer to [Application Scenarios](#) for application scenarios of Adapter.

## Functions

With Adapter, you can:

- Control Mech-Vision and Mech-Viz

Category	Function
Mech-Vision	Start Mech-Vision project and get vision target(s)
	Configure Step parameters in Mech-Vision
	Read Step parameters in Mech-Vision
	Switch Mech-Vision Recipe
Mech-Viz	Start Mech-Viz
	Stop Mech-Viz
	Configure Task parameters in Mech-Viz
	Read Task parameters in Mech-Viz
	Set serial number of the gripper
	Set the operation speed of the robot
	Set parameters of collision detection
	Obtain the running status of Mech-Viz
Others	Please refer to <a href="#">Adapter Programming Guide</a>

- Enable non-vision functions such as user interface customization, database creation, file reading and writing, and communication with the Web system.

You will need to program in Python to enable these functions on external services.

## Development

For TCP/IP Socket communication, Mech-Center provides an **Adapter Generator**, which helps beginners to generate an Adapter program rapidly and then build an Adapter project. Please refer to [Adapter Generator Guide](#) for detailed information.

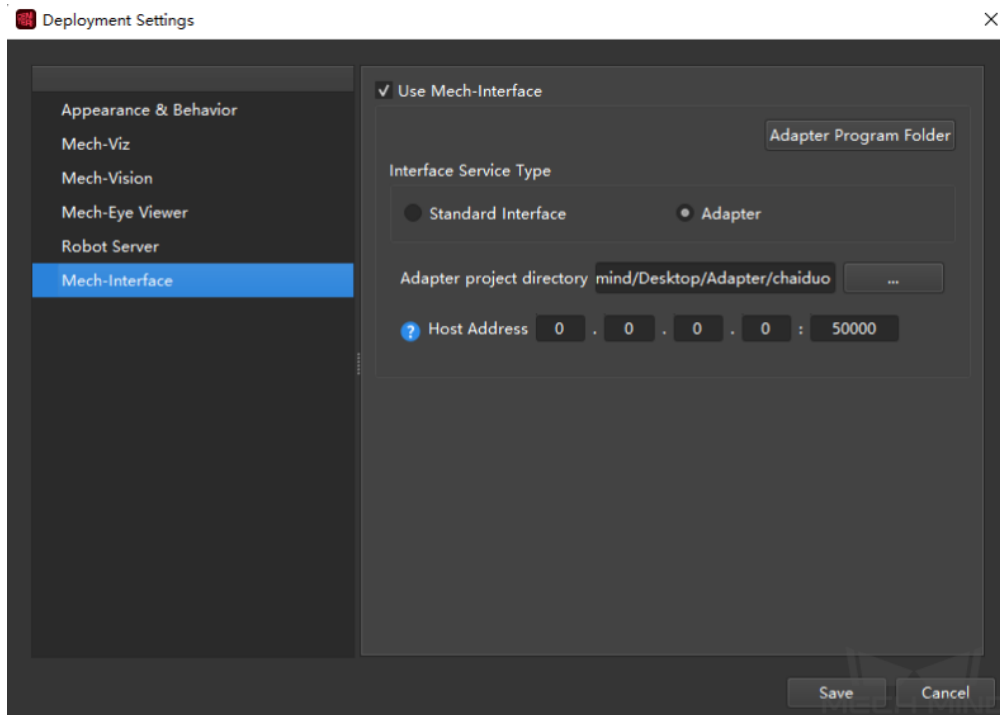
You can program based on the generated Adapter program.

If you would like to program Adapter from scratch, please refer to [Adapter Programming Guide](#) and [Adapter Programming Examples](#) for detailed instructions.

## Deployment

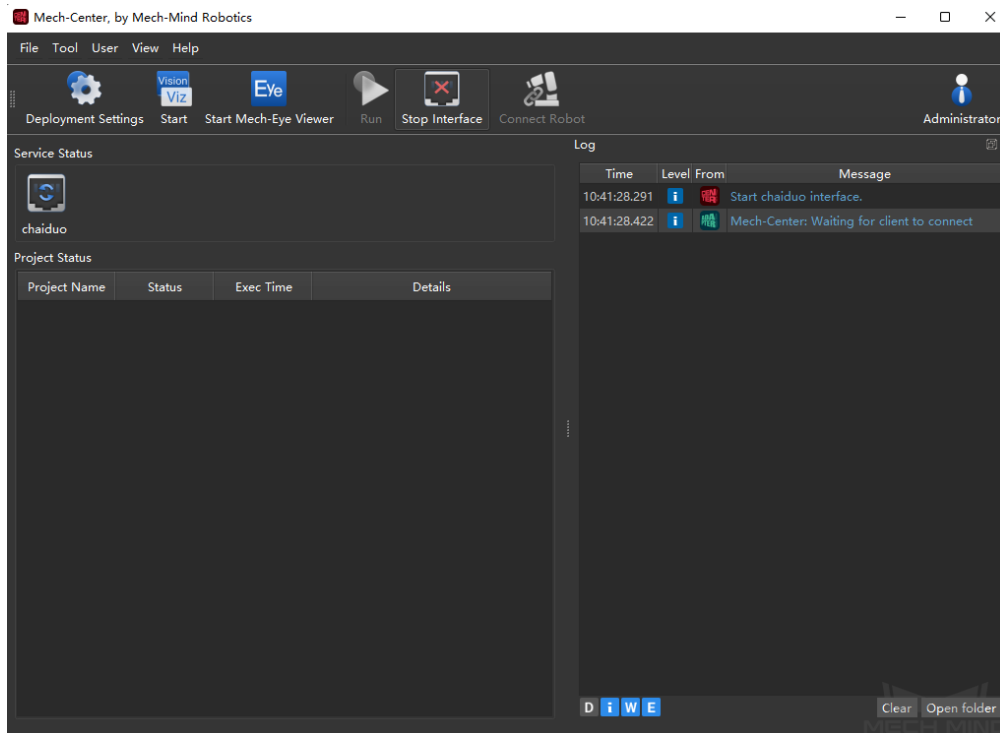
After programming the Adapter, you can deploy the program in the project.

1. Open Mech-Center, and click *Deployment Settings* in the toolbar.
2. In the **Deployment Settings** window, select **Mech-Interface**; select the **Use Mech-Interface** checkbox and select **Adapter** as the **Interface Service Type**.



3. Click ... next to the **Adapter project directory** to select the Adapter program.
4. Set the **Host Address** according to the on-site situations. The port must be consistent with the peer port.
  - If the peer device works as the server, the **Host Address** should be set as the IP address of the peer device.
  - If the peer device works as the client, the **Host Address** should be set to 0.0.0.0.
1. Click **Save** and then restart Mech-Center.
2. Click *Start Interface* in the toolbar to enable Adapter service.

When the *Start Interface* button turns to *Stop Interface* and the Adapter program is displayed in the **Service Status** panel, the Adapter service is enabled successfully, as shown below.



After reading this section, you can generate your first Adapter program according to [Adapter Generator Guide](#).

### 3.4.2 Adapter Generator Guide

This section introduces the steps for using Adapter Generator to generate an Adapter program rapidly.

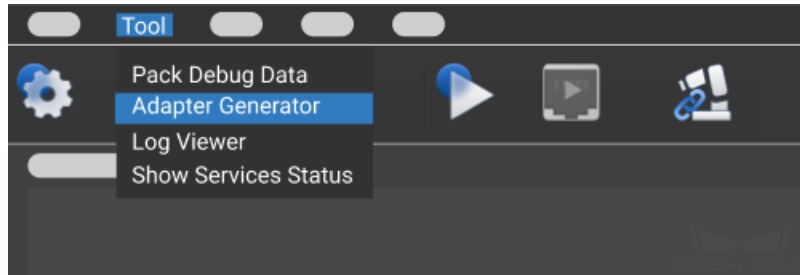
#### Introduction of the Adapter Generator

Adapter Generator is a built-in component of Mech-Center. It applies to scenarios where vision results are provided by Mech-Vision only and a TCP/IP Socket is used for the communication.

With the Adapter Generator, you can:

- Generate an Adapter program rapidly and create an Adapter project.
- Learn to program Adapter to meet complex actual requirements.

Select *Tool* → *Adapter Generator* to open the Adapter Generator and follow the instructions to set up.



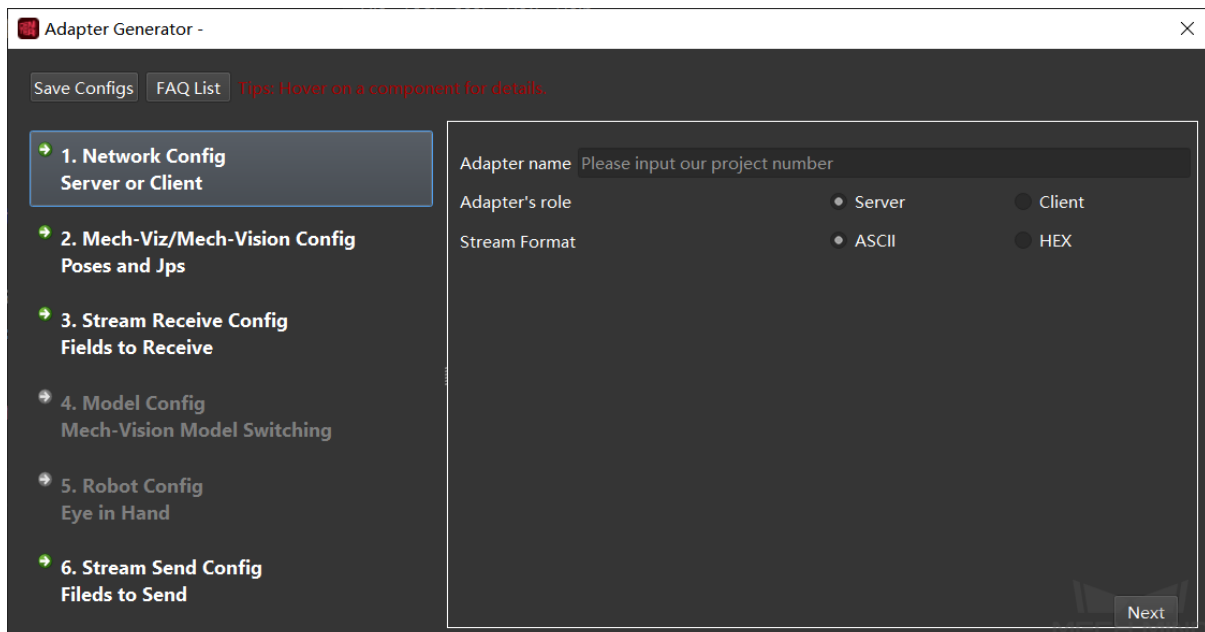
**Attention:** The Adapter Generator is only available in the Administrator mode.

### Generate an Adapter Program

**Hint:** You can hover over the options to view the tooltips.

### Network Config—Server or Client

Please set the parameters **Adapter name**, **Adapter's role**, and **Stream Format** in this step, and then click **Next**.



Parameters:

- **Adapter name:** specifies the name of the Adapter program.
- **Adapter's role:** specifies the Adapter as a server or client. If the Adapter functions as a client and there is a port restriction on the server, please check **Bind Port**.

---

**Hint:** In order to achieve successful communication, please make sure the host address is configured correctly in Deployment Settings. Please refer to [Deployment](#) for detailed instructions.

---

- **Stream Format:** specifies the format for data transmission. ASCII string and HEX are supported. If HEX is selected, you will need to choose between **Big endian** and **Little endian**.

### Mech-Viz/Mech-Vision Config—Poses and JPs

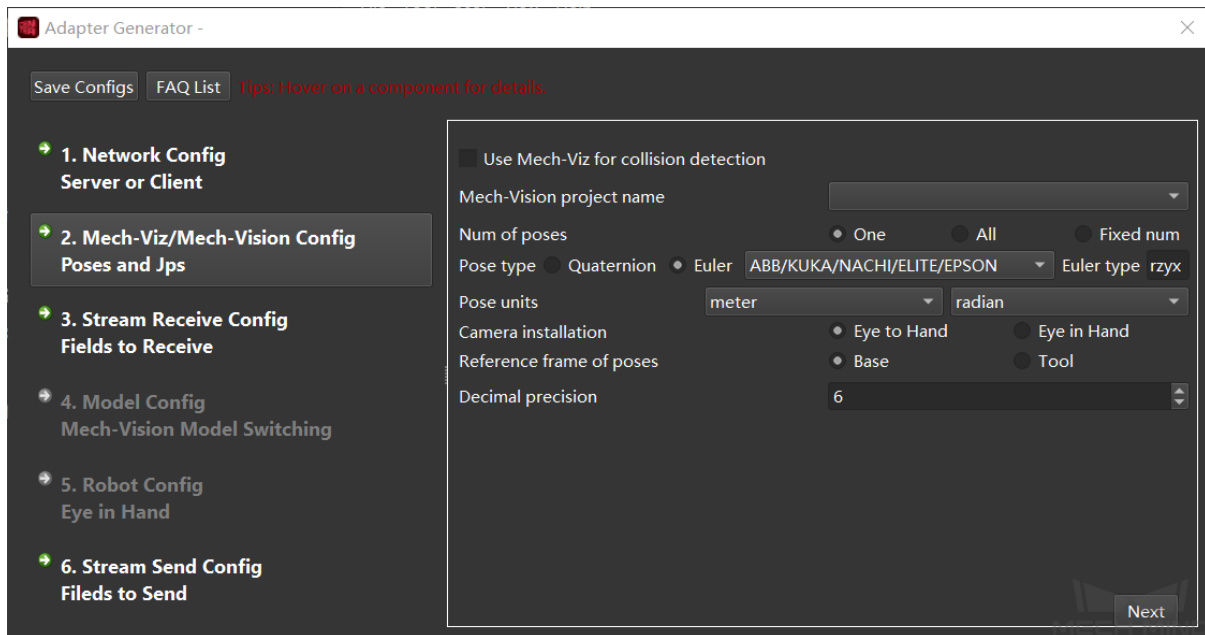
---

**Hint:**

- Before this step, please prepare an executable Mech-Vision project and a Mech-Viz project used for collision detection. Please enable **Autoload** in the toolbar of Mech-Viz, and right-click the project name in the Projects List in Mech-Vision and select **Autoload Project**. Then make sure the Mech-Vision project and Mech-Viz project are loaded in Mech-Center.
  - Mech-Center provides an example project used for collision detection, which is stored in `tool\viz_project\check_collision.in` in the installation directory of Mech-Center.
- 

**Attention:** As the workflow of `check_collision` example project, the Task `vision_look` triggers the camera to capture images. Non-move-type Tasks must be included in the workflow. Please do not modify the names of Tasks, or else an error may occur.

Please configure parameters related to Mech-Vision and Mech-Viz projects, and then click on **Next**.



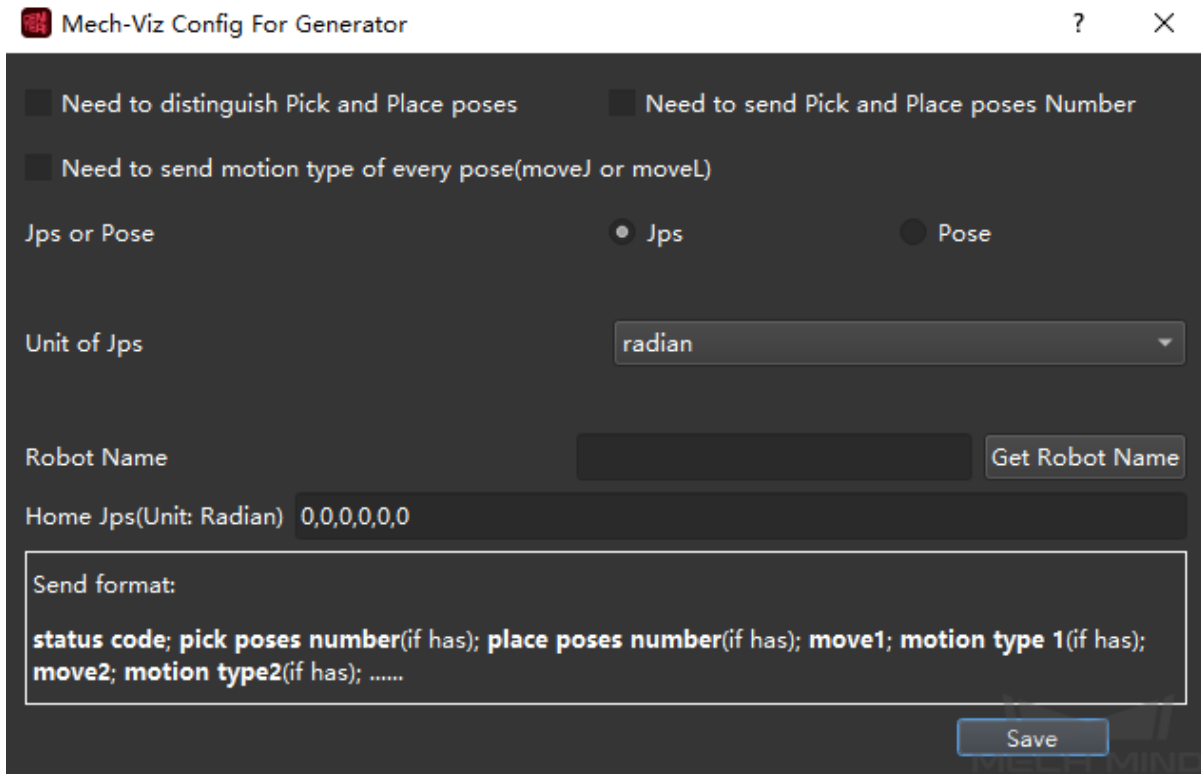
### Parameters related to Mech-Vision

- **Mech-Vision project name:** select the Mech-Vision project that communicates with Adapter in the drop-down list.
- **Num of poses:** select the number of poses to be sent to the peer device.
- **Pose type:** select whether to use quaternions or Euler angles to represent the poses.
- **Pose units:** the unit used for the poses; usually in millimeter and degree.
- **Camera Installation:** chooses the camera installation methods, including ETH and EIH.
- **Reference frame of poses:** the robot base reference frame is usually used as the reference frame of poses. For scenarios using EIH, when the pose of robot flange cannot be provided, only the tool reference frame can be used.
- **Decimal precision:** the precision of the pose data. The maximum number of decimal places is 10.

### Parameters related to Mech-Viz

- **Use Mech-Viz for collision detection:** select this option to detect collisions on the vision points. Pick points from unsuccessful planning will be filtered, and collision-free pick points will be kept.
- **Mech-Viz Config:** this option is only available when *Use Mech-Viz for collision detection* is selected. Click the *Mech-Viz Config* button to open the **Mech-Viz Config For Generator** window. After configuring relevant parameters, click *Save*.

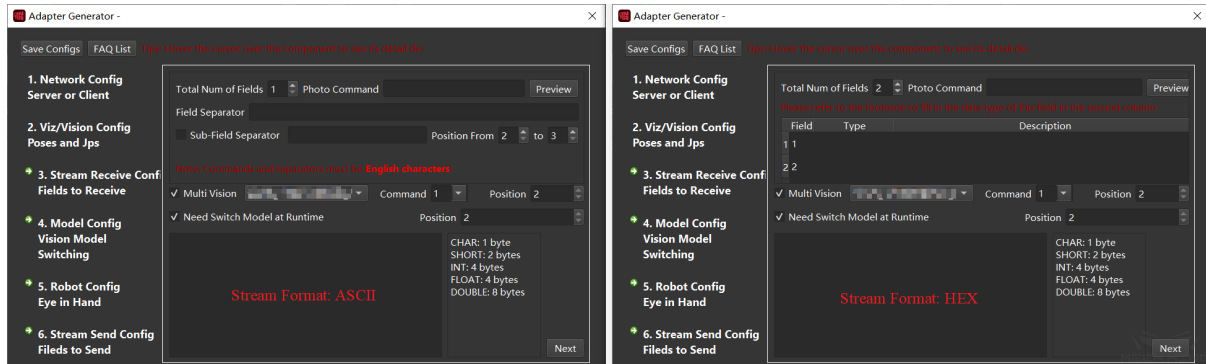




- **Need to distinguish Picking and Placing Poses:** the picking pose refers to all poses before the vision\_move Task (including the pose of vision\_move Task), and the place pose refers to poses after the vision\_move Task.
- **Need to send Pick and Place poses Number:** if there are many targets, the number of poses can be sent as well.
- **Need to send motion type of every pose(moveJ or moveL):** the motion type of the move-type Tasks, including moveJ and moveL.
- **Update Code:** the code for moveJ is 1, and the code for moveL is 2 by default. You can also customize the code. After modification, click *Update Code* to apply the settings.
- **Jps or Pose:** the format of the poses; **Jps** is selected by default. If **Pose** is selected, please go to **Others** panel in Mech-Viz and select **Send TCP**.
- **Unit of Jps/Unit of Pose:** when **Jps** is selected, **degree** is usually used as the unit. When **Pose** is selected, and the pose is represented by **quaternions**, **millimeter** is usually used as the unit; when the pose is represented by **Euler angles**, **millimeter** and **degree** are usually used as the unit.
- **Robot Name:** click *Get Robot Name* to autoload the name of the robot. The simulated robot in Mech-Viz should be based on a real robot. Adapter will simulate the service of the real robot. Please make sure the robot name here is the same as the one in Mech-Viz.
- **Home Jps(Unit: Radian):** set the home position of the robot in Mech-Viz. The unit is radian. Please use commas to connect the numbers. You can add a move Task in the workflow and set its JPs as the home position, and then copy the JPs.

## Stream Receive Config—Fields to Receive

Please set the format of the field, including photo command, multiple projects (instruction code), dynamically switch templates (template instruction code), as well as the total number of fields, field types, field separators and sub-field separators. After the configuration, click on *Next*.



### Functions:

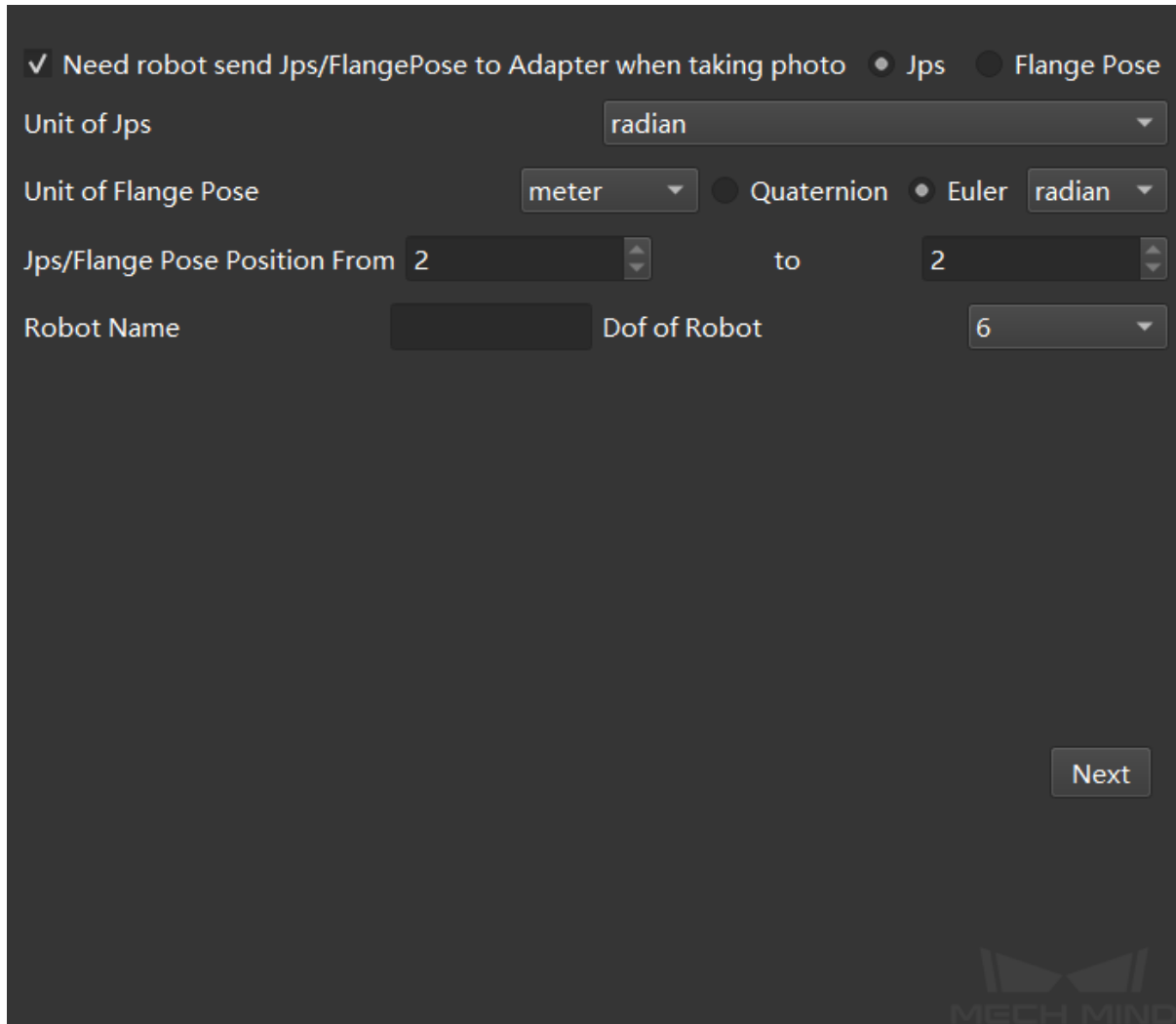
- **Total Num of Fields:** related to the number of parameters that need to be set, and the value range is 1~10. There must be a camera command in the field.
- **Photo Command:** the photo command sent from external devices to Mech-Mind software system, which triggers the camera to capture images. When the stream format is ASCII string, it is recommended to use English letters to represent the command. For example, use letter *p* as the command and the field position is 1 by default. When the communication format is hexadecimal (HEX), an integer in hexadecimal format is required, such as *0xff* or *ff*.
- **Field Separator and Sub-Field Separator:** only need to be set when the communication format is ASCII string. If there are more than two fields, you need to fill in the field separator; if there are additional separators in the additional information, the sub-field separator is also necessary, and you can specify the start and end range of the sub-field.
- **Field Type and Description:** It needs to be set when the communication format is hexadecimal (HEX). The available types are CHAR, SHORT, INT, FLOAT, and DOUBLE. You can describe the function in the **Description**.
- **Multi Vision:** This setting is optional. When there are multiple Vision projects in a project, different Vision projects need to be called according to external commands, and the command codes are configurable.

**Attention:** The command code and the field position of each project should be unique.

## Robot Config—Eye in Hand

**Hint:** This step is only available when **Camera Installation** is set to **Eye In Hand** in the previous *Mech-Viz/Mech-Vision Config—Poses and JPs* step.

Please set the robot pose when capturing images in this step and then click *Next*.



### Parameters:

- **Need robot send Jps/Flange Pose to Adapter when taking photo:** if the object pose in the robot base reference frame needs to be sent to the peer device, JPs or flange pose when the robot captures images are needed. After checked this option, you can select between JPs and flange pose.
- **Unit of Jps:** unit of joint positions; radian and degree are supported.
- **Unit of Flange Pose:** unit of the flange pose. When **Quaternion** is selected, **millimeter** and **meter** are available units; when the pose is represented by **Euler angles**, **millimeter** and **degree** are unavailable.

units.

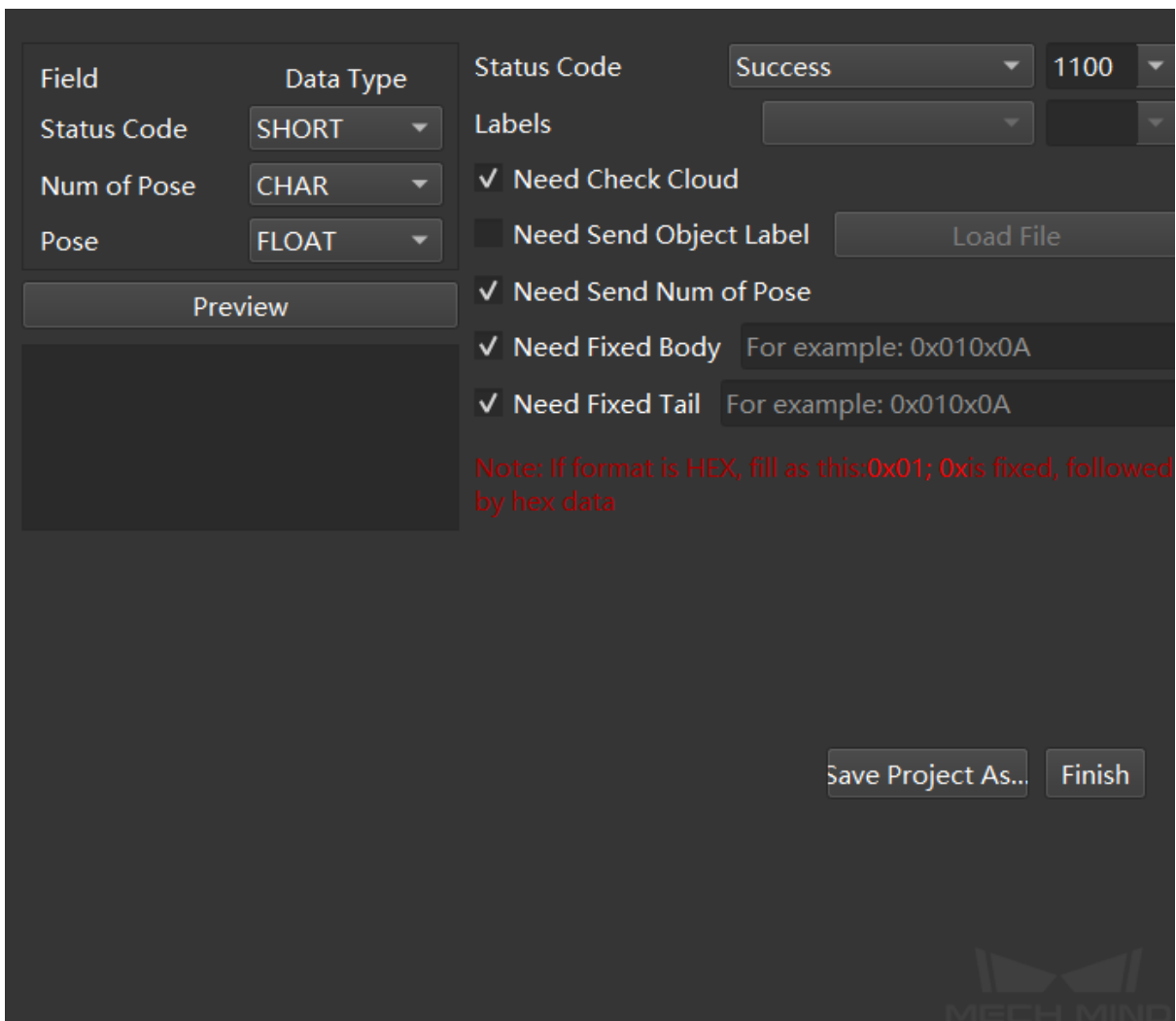
- **Jps/Flange Pose Position From:** It is the position of the start and end fields of the pose in the total field.

**Attention:** The index position starts counting from 1, and the index position 1 is the camera instruction!

- **Robot Name:** The name used to identify the robot service, which needs to be consistent with the robot name in Mech-Viz.
- **Dof of Robot:** the degree of freedom of the robot. Only 4-axis and 6-axis robots are supported currently. Please select the corresponding robot degree of freedom according to the actual project.

### Stream Send Config—Fields to Send

Please set the format of the poses to be sent in this step, and then click on *Next*.



Field	Data Type	Status Code
Status Code	SHORT	Success 1100
Num of Pose	CHAR	
Pose	FLOAT	

Labels: [Dropdown]

Need Check Cloud

Need Send Object Label [Load File]

Need Send Num of Pose

Need Fixed Body For example: 0x010x0A

Need Fixed Tail For example: 0x010x0A

Note: If format is HEX, fill as this: 0x01; 0xis fixed, followed by hex data

[Preview]

[Save Project As...] [Finish]

**Parameters:**

- **Field Separator** and **Sub-Field Separator**: only need to be set when the communication format is ASCII. If there are more than two fields, you need to fill in the field separator; if there are additional separators in the additional information, the sub-field separator is also necessary, and you can specify the start and end range of the sub-field.
- **Status Code**: Set the sending status, each status corresponds to a unique status code.
- **Need Check Cloud**: After checking, the point clouds will be checked, if the point clouds does not exist, the corresponding status code will be output.
- **Need Send Object Label**: Sending object labels means to send to the peer device according to the label recognized by Vision, each label is connected to the Pose; when the peer device is inconvenient to parse the label string, you can also specify the code of the corresponding label, which needs to load the label file of all label strings. It should be noted that the label file format must be in JSON array format.
- **Need Send Num of Pose**: Sends the number of Pose of this time.
- **Need Fixed Body**: When the vision is not recognized, send a message to the peer device (the message after the error code).
- **Need Fixed Tail**: After checking it, a fixed tail mark will be added after the data.

**Attention:** When the communication format is hexadecimal (HEX), it is necessary to set the status code, the number of poses, and the numeric type of poses.

After configuring all the above settings, click *Finish* or *Save project as* to save the Adapter project.

Please refer to [Deployment](#) for instructions on how to deploy the Adapter project.

For more information about programming and development on Adapter, please refer to [Adapter Programming Guide](#) and [Adapter Programming Examples](#).

### 3.4.3 Adapter Programming Guide

This chapter covers the programming style and syntax of Adapter.

#### Target Reader

This chapter is targeted for developers of Adapter, who should be familiar with:

- Basic syntax of Python
- JSON data format

---

It is recommended to read **Adapter Programming Styles** first.

## Adapter Programming Styles

Adapter is written in Python, and therefore developers of Adapter should adhere to [Style Guide for Python Code](#).

Besides, to improve code quality and readability of the program, developers of Adapter should adhere to the conventions defined in this section as well.

- *Naming Conventions*
- *Comment*
- *Log*

### Naming Conventions

1. Use camel case to name the class, as shown below.

```
class AdapterWidget
```

2. Use lowercase letters connected with underscores to name the member variables and member functions, as shown below.

```
self.pick_count # Note that variables are generally nouns
def start_viz(self): # Note that the function name is generally a verb + object
    pass
```

3. A underscore before the name of a member variable or member function suggests that it is recommended to use these variables and functions only inside the class. However, you can still use the member variables and functions outside the class if necessary.

```
self._socket = socket() # Indicates that the _socket variable is only used inside the
↳class
def _init_widget(self): # Indicates that the _init_widget function is only used inside
↳the class
    pass
```

4. Use uppercase letters connected with underscores to name constants, as shown below.

```
ADAPTER_DIR = "D:/adapter_for_communication"
```

### Comment

Please only add comments when the logic is complex or the information of the comment is critical.

- Single line comments

Use the # to start the comment

- Multiline comments

Use triple quotes (""") to quote the multiline comments. Please note that the "" that ends a multiline comment should be on a line by itself.

- The introduction of a function, class, or package

The introduction of a function or class should be put at the end of the function or class, while the introduction of a package should be put in the front.

```
def viz_is_running(self):

    """
    Will be called when find viz is running during starting viz.
    """

class Adapter(QObject):

    """
    Base class which encapsulates Viz/Vision/Hub/Robserver inter faces.
    """

    """
    Service base classes.
    """

import logging
```

## Log

Log can be used for troubleshooting when an error occurs. The log should be output at a proper time, for example, when a key function is called and the reference data is returned.

Adapter supports two methods to output the log:

- print: This method can only output the log in the console, which is convenient to view the running status of the project in real time. However, the log messages will be lost if an error occurs, and therefore it is not recommended to use this method in actual production.
- logging: This method is more recommended as it enables to set the display format of the log and save the log messages in a file (optional).

Then you can proceed on the **Abstract Parent Interface** and **Util** package involved in Adapter programming. It is expected that you can well understand the basic parent categories and common functions in Adapter.

## Abstract Parent Class Interfaces

Abstract Parent Class Interfaces are the functions that can be added with a child class that will inherit from its parent class. You can rewrite the functions according to actual requirements. This section will introduce you to the following abstract parent classes.

- *Communication*

- *Communication Class*
- *TcpServer Class*
- *TcpClinet Class*
- *Adapter*
  - *Adapter Base Class*
  - *TcpServerAdapter Class*
  - *TcpClientAdapter Class*
  - *TcpMultiplexingServerAdapter Class*
  - *IOAdapter Class*
  - *AdapterWidget Class*
- *Service*
  - *NotifyService Class*
  - *VisionResultSelectedAtService Class*
  - *RobotService Class*
  - *OuterMoveService Class*
  - *Register Service*

### Communication

The source code belongs to communication category is stored in the `/src/interface/communication.py` file in the installation directory of Mech-Center.

### Communication Class

Communication class is a basis class used for communication. It provides a series of interface functions which should be rewritten on the server or client.

Class Function	Description
<code>is_connected()</code>	Determine if the current connection is successful.
<code>set_rcv_size()</code>	Set the length of the data received each time; the default value is 1024 bytes.
<code>send()</code>	Interface function which is used to send data.
<code>rcv()</code>	Interface function which is used to receive data.
<code>close()</code>	Interface function which is used to close the socket communication.
<code>before_rcv()</code>	Interface function which you can add an logic according to actual needs before receiving the data. Function overriding is available.
<code>after_rcv()</code>	Interface function which you can add an logic according to actual needs after receiving the data. Function overriding is available.
<code>after_handle()</code>	Interface function which you can add an logic according to actual needs after processing the data. Function overriding is available.



### TcpServer Class

TcpServer class encapsulates a TCP/IP Socket server.

Class Function	Description
bind_and_listen()	Bind the port
local_socket()	Provide the local Socket information
remote_socket()	Provide the remote Socket information
accept()	Accept the connection with the client
send()	Send the data
recv()	Receive the data
close()	Close the socket
close_client()	Disconnect with the client

### TcpClient Class

TcpClient class encapsulates a TCP/IP Socket client.

Class Property	Description
is_bind_port	Whether to bind the port. If there is a port restriction on the client, the value should be set to True.

Class Function	Description
send()	Send the data
recv()	Receive the data
close()	Close the socket
set_timeout()	Timeout period in seconds
reconnect_server()	Reconnect to the server
after_connect_server()	Interface function which is used to specify what to do after the first successful connection with the server
after_reconnect_server()	Interface function which is used to specify what to do after reconnecting with the server
after_timeout()	Interface function which is used to specify what to do after the timeout period

### Adapter

The source code belongs to Adapter category is stored in the `/src/interface/adapter.py` file in the installation directory of Mech-Center.

### Adapter Base Class

Adapter encapsulates functions related to Mech-Viz, Mech-Vision, Mech-Center, Robserver, including `start_viz()`, `stop_viz()`, `set_task_property()`, and `set_step_property()`. When an Adapter program is used to control Mech-Viz or Mech-Vision, the child class of Adapter base class must be used.

The class properties of Adapter are shown in the table below.

Class Property	Description
<code>viz_project_dir</code>	Directory of the current Mech-Viz project
<code>vision_project_name</code>	Name of the current Mech-Vision project
<code>is_simulate</code>	Whether to run Mech-Viz in simulation mode
<code>is_keep_viz_state</code>	Whether to keep the state when Mech-Viz stopped last time
<code>is_save_executor_data</code>	Whether to save the data of Mech-Viz executor
<code>is_force_simulate</code>	Whether to force Mech-Viz to run in simulation mode
<code>is_force_real_run</code>	Whether to force Mech-Viz to run and control the real robot
<code>code_signal</code>	The signal of displaying Adapter information in the main interface of Mech-Center (including error codes)
<code>msg_signal</code>	The signal of displaying Adapter information in the main interface of Mech-Center (NOT including error codes)
<code>i_code_signal</code>	The signal of displaying Mech-Interface information in the main interface of Mech-Center (including error codes)
<code>i_msg_signal</code>	The signal of displaying Mech-Interface information in the main interface of Mech-Center (NOT including error codes)
<code>viz_finished_signal</code>	The signal indicating that Mech-Viz is stopped normally or with an error
<code>connect_robot_signal</code>	Connect/disconnect with the robot signal
<code>start_adapter_signal</code>	Start Adapter signal
<code>service_name_changed</code>	The signal of displaying statuses of Mech-Viz and Mech-Vision in the main interface of Mech-Center
<code>setting_infos</code>	The configuration information of Mech-Center
<code>service_name</code>	Name of the registered service

The Adapter class functions are as shown in the table below.

Class Function	Description
<code>on_exec_status_changed()</code>	Receive the status information from Mech-Viz and Mech-Vision
<code>register_self_service()</code>	Register Adapter service
<code>vision_project_dirs(self):</code>	Check the directory of Mech-Vision project
<code>vision_project_names()</code>	Check all project names in Mech-Vision
<code>vision_project_names_in_center()</code>	Check the names of all Mech-Vision projects loaded in Mech-Center
<code>is_viz_registered()</code>	Check whether Mech-Viz is registered in Mech-Center
<code>is_viz_in_running()</code>	Check whether Mech-Viz is running
<code>is_vision_started()</code>	Check whether Mech-Vision project is registered in Mech-Center
<code>find_services()</code>	Check the service
<code>before_start_viz()</code>	This function will be called before starting Mech-Viz
<code>after_start_viz()</code>	This function will be called after starting Mech-Viz
<code>viz_not_registerd()</code>	This function will be called if Mech-Viz is not registered after being started

continues c

Table 2 – continued from previous page

viz_is_running()	This function will be called if Mech-Viz is already running when starting Mech-Viz
viz_run_error()	This function will be called if an error occurs when running Mech-Viz
viz_run_finished()	This function will be called after Mech-Viz is stopped
viz_plan_failed()	This function will be called after Mech-Viz fails to plan a path
viz_unreachable_targets()	This function will be called when there is an unreachable target in the path planned
viz_collision_checked()	This function will be called when a collision is detected in the path planned by Mech
parse_viz_reply()	Parse the reply from Mech-Viz
wait_viz_result()	Wait for Mech-Viz to reply
start_viz()	Start Mech-Viz
stop_viz()	Stop Mech-Viz
pause_viz()	Pause Mech-Viz
find_vision_pose()	Trigger Mech-Vision project to capture images
async_call_vision_run()	Trigger Mech-Vision project to capture images asynchronously
async_get_vision_callback()	Receive the result from Mech-Vision asynchronously
deal_vision_result()	Process the result from Mech-Vision
set_step_property()	Set Step parameters in Mech-Vision
read_step_property()	Read Step parameters in Mech-Vision
select_parameter_group()	Select the parameter recipe in Mech-Vision
set_task_property()	Set Task parameters in Mech-Viz
read_task_property()	Read Task parameters in Mech-Viz
get_digital_in()	Obtain DI
set_digital_out()	Set DO
before_start_adapter()	This function will be called before starting Adapter
start()	Start Adapter
close()	Stop Adapter
handle_command()	Process commands received from external devices

### TcpServerAdapter Class

As shown below, TcpServerAdapter class inherits from Adapter, and it encapsulates functions of TcpServer.

```

class TcpServerAdapter(Adapter):
    def __init__(self, host_address, server=TcpServer):
        super(TcpServerAdapter, self).__init__()
        self.init_server(host_address, server)

    def init_server(self, host_address, server=TcpServer):
        self._server = server(host_address)

    def set_recv_size(self, size):
        self._server.set_recv_size(size)

    def send(self, msg, is_logging=True):
        return self._server.send(msg, is_logging)

    def recv(self):
        return self._server.recv()

    def start(self):

```

(continues on next page)

(continued from previous page)

```

self.before_start_adapter()
while not self.is_stop_adapter:
    try:
        self._server.before_recv()
        cmds = self._server.recv()
        logging.info("Received raw data from client:{}".format(cmds))
        if not cmds:
            logging.warning("Adapter client is disconnected!")
            self.code_signal.emit(logging.WARNING, CENTER_CLIENT_DISCONNECTED)
            self._server.close_client()
            self.accept()
            continue
        self._server.after_recv()
    except socket.error:
        logging.warning("Adapter client is closed!")
        self.code_signal.emit(logging.WARNING, CENTER_CLIENT_DISCONNECTED)
        self._server.close_client()
        self.accept()
    except Exception as e:
        logging.exception("Exception occurred when receiving data from client: {}".format(e))
    else:
        try:
            self.handle_command(cmds)
            self._server.after_handle()
        except Exception as e:
            self.msg_signal.emit(logging.ERROR, _translate("messages", "Handle command
↳exception: {}".format(e)))
            logging.exception("Adapter exception in handle_command(): {}".format(e))

def close(self):
    super().close()
    self._server.close()

def before_start_adapter(self):
    super().before_start_adapter()
    self.accept()

def accept(self):
    if self.is_stop_adapter:
        return
    self.code_signal.emit(logging.INFO, CENTER_WAIT_FOR_CLIENT)
    self._server.accept()
    if self._server.is_connected():
        self.code_signal.emit(logging.INFO, CENTER_CLIENT_CONNECTED)
        self.msg_signal.emit(logging.INFO, _translate("messages", "Client address is") + "
↳{}".format(self._server.remote_socket()[1]))

```

## TcpClientAdapter Class

As shown below, TcpClientAdapter class inherits from Adapter, and it encapsulates functions of TcpClient.

```
class TcpClientAdapter(Adapter):

    def __init__(self, host_address):
        super().__init__()
        self.init_client(host_address)

    def init_client(self, host_address, client=TcpClient):
        self._client = client(host_address)

    def set_bind_port(self, is_bind=True):
        self._client.is_bind_port = is_bind

    def set_recv_size(self, size):
        self._client.set_recv_size(size)

    def send(self, msg, is_logging=True):
        self._client.send(msg, is_logging)

    def recv(self):
        return self._client.recv()

    def start(self):
        self.reconnect_server(False)
        while not self.is_stop_adapter:
            try:
                self._client.before_recv()
                cmds = self._client.recv()
                if not cmds:
                    self.reconnect_server()
                    continue
                logging.info("Received command from server:{}".format(cmds))
                self._client.after_recv()
            except socket.timeout:
                logging.warning("Socket timeout")
                self._client.after_timeout()
            except socket.error:
                sleep(5)
                self.reconnect_server()
            except Exception as e:
                logging.exception("Exception occurred when receiving from server: {}".
↳format(e))
            else:
                try:
                    self.handle_command(cmds)
                except Exception as e:
                    self.msg_signal.emit(logging.ERROR, _translate("messages", "Handle command
↳exception: {}".format(e)))
                    logging.exception("Adapter exception in handle_command(): {}".format(e))

    def close(self):
        super().close()
```

(continues on next page)

(continued from previous page)

```

self._client.close()

def reconnect_server(self, is_reconnect=True):
    self._client.reconnect_server()
    if self.is_stop_adapter:
        return
    if self._client.is_connected():
        self.code_signal.emit(logging.INFO, CENTER_CONNECT_TO_SERVER)
    else:
        self.code_signal.emit(logging.WARNING, CENTER_SERVER_DISCONNECTED)

```

### TcpMultiplexingServerAdapter Class

As shown below, TcpMultiplexingServerAdapter class inherits from Adapter, and it is mainly used to connect multiple clients.

```

class TcpMultiThreadingServerAdapter(Adapter):
    def __init__(self, address):
        super().__init__()
        self._servers = {}
        self.add_server(address)
        self.sockets = {}
        self.clients_ip = {}
        self.thread_pool = ThreadPoolExecutor(max_workers=4, thread_name_prefix="tcp_multi_
↪server_thread")
        self.thread_id_socket_dict = {}
        self.set_rcv_size()

    def set_rcv_size(self, size=1024):
        self.rcv_size = size

    def _find_client_ip(self, sock):
        for k, v in self.sockets.items():
            if v == sock:
                return k

    def _find_server(self, sock):
        for k, v in self._servers.items():
            if v == sock:
                return k

    def add_server(self, host_address):
        server = TcpServer(host_address)
        server.bind_and_listen()
        self._servers[server] = server.local_socket()

    def set_clients_ip(self, clients_ip):
        """
        Must be called before start().
        `clients_ip` is a dict(key is client ip, value is client description).
        """
        self.clients_ip = clients_ip

```

(continues on next page)

(continued from previous page)

```

def add_connection(self, ip_port, sock):
    self.sockets[ip_port] = sock
    logging.info("Add {}, connections: {}".format(ip_port, self.sockets))
    self.msg_signal.emit(logging.INFO, _translate("messages", "The client {} gets online.
↪").format(ip_port))

def del_connection(self, ip):
    logging.info("Del {}, connections: {}".format(ip, self.sockets))
    if self.client_connection(ip):
        self.client_connection(ip).close()
        self.sockets.pop(ip)
    self.msg_signal.emit(logging.WARNING, _translate("messages", "The client {} gets_
↪offline.").format(ip))

def client_connection(self, client_ip):
    return self.sockets.get(client_ip)

def check_read_events(self, rs):
    for s in rs:
        if s in self._servers.values(): # recv connection
            server = self._find_server(s)
            if self.is_stop_adapter:
                return
            server.accept()
            client_socket, client_addr = server.remote_socket()
            ip_port = "{}:{}".format(str(client_addr[0]), str(client_addr[1]))
            self.add_connection(ip_port, client_socket)
        elif s in self.sockets.values(): # recv data
            client_ip = self._find_client_ip(s)
            if not client_ip:
                continue
            msg = self.recv_by_s(s)
            if not msg:
                self.del_connection(client_ip)
                return
            try:
                future = self.thread_pool.submit(self.handle_command_thread, s, msg)
            except Exception as e:
                logging.exception("Adapter exception in handle_command(): {}".format(e))

def handle_command_thread(self, s, msg):
    thread_id = threading.get_ident()
    self.thread_id_socket_dict[thread_id] = s
    self.handle_command(msg)
    # del self.thread_id_socket_dict[thread_id]

def send(self, msg, is_logging=True):
    thread_id = threading.get_ident()
    sock = self.thread_id_socket_dict.get(thread_id)
    len_total = len(msg)
    while msg:
        if sock:
            len_sent = sock.send(msg)

```

(continues on next page)

(continued from previous page)

```

        else:
            for v in self.sockets.values():
                try:
                    len_sent = v.send(msg)
                except Exception as e:
                    logging.warning(e)
            if not len_sent:
                logging.warning("Connection lost, close the client connection.")
                return len_sent
            if is_logging:
                logging.info("Server send: {}, len_sent: {}".format(msg, len_sent))
            msg = msg[len_sent:]
        return len_total

    def recv(self):
        thread_id = threading.get_ident()
        sock = self.thread_id_socket_dict.get(thread_id)
        return self.recv_by_s(sock)

    def recv_by_s(self, sock):
        msg = b""
        try:
            msg = sock.recv(self.recv_size)
        except socket.error:
            logging.error("The client is closed!")
        if msg:
            logging.info("Received message: {}".format(msg))
        return msg

    def check_task(self):
        """
        Interface.
        """

    def close(self):
        super().close()
        for server in self._servers.keys():
            server.close()
        for client_ip in self.sockets.keys():
            try:
                self.client_connection(client_ip).close()
                logging.info("Close socket :{}".format(client_ip))
            except Exception as e:
                logging.warning("Close socket error: {}, exception: {}".format(client_ip, e))
        self.sockets = {}

    def start(self):
        self.before_start_adapter()
        while not self.is_stop_adapter:
            available_sockets = list(self.sockets.values()) + list(self._servers.values())
            rs, _, _ = select(available_sockets, [], [], 0.1)
            self.check_read_events(rs)
            try:
                self.check_task()
            
```

(continues on next page)



(continued from previous page)

```

    except Exception as e:
        self.msg_signal.emit(logging.ERROR,
                              _translate("messages", "Handle command exception: {}".
←format(e)))
        logging.exception("Exception when check task:{}".format(e))
        sleep(5)

```

## IOAdapter Class

As shown below, IOAdapter class inherits from Adapter, and it uses a while loop in obtaining DI.

```

class IOAdapter(Adapter):
    robot_name = None
    check_rate = 0.5

    def __init__(self, host_address):
        super().__init__()
        self.last_gi = 0

    def get_digital_in(self, timeout=None):
        return super().get_digital_in(self.robot_name, timeout)

    def set_digital_out(self, port, value, timeout=None):
        super().set_digital_out(self.robot_name, port, value, timeout)

    def _check_gi(self):
        gi_js = self.get_digital_in()
        gi = int(json.loads(gi_js.decode())["value"])
        if self.last_gi != gi:
            self.last_gi = gi
            logging.info("Check GI signal status: {}".format(gi))
            self.handle_gi(gi)

    def start(self):
        self.before_start_adapter()
        while not self.is_stop_adapter:
            try:
                self._check_gi()
            except Exception as e:
                logging.exception(e)
                self.check_gi_failed()
            sleep(self.check_rate)

    def handle_gi(self, gi):
        """
        Interface.
        """

    def check_gi_failed(self):
        """
        Interface.
        """

```

## AdapterWidget Class

As shown below, AdapterWidget class is a parent class, and all functions related to user interface customization must be its child classes.

```
class AdapterWidget(QWidget):

    def set_adapter(self, adapter):
        self.adapter = adapter
        self.after_set_adapter()

    def after_set_adapter(self):
        """
        Interface.
        """

    def close(self):
        super().close()
        """
        Interface.
        """
```

## Service

The source code belongs to service category is stored in the `/src/interface/services.py` file in the installation directory of Mech-Center.

## NotifyService Class

The code of NotifyService class is shown below.

```
class NotifyService(JsonService):
    service_type = "notify"
    service_name = "adapter"

    def handle_message(self, msg):
        """
        Interface.
        """

    def notify(self, request, _):
        msg = request["notify_message"]
        logging.info("notify message:{}".format(msg))
        return self.handle_message(msg)
```

The default service name is "adapter". If multiple notify services are needed in the project, you can override the `service_name` to distinguish different services.

The descriptions of class functions are shown in the table below.

Class Function	Description
handle_message()	Interface function; the child class can be overridden
notify()	Parse the message; the child class usually does not need to be overridden

### VisionResultSelectedAtService Class

The code of VisionResultSelectedAtService class is shown below.

```
class VisionResultSelectedAtService(JsonService):
    service_type = "vision_watcher"
    service_name = "vision_watcher_adapter"

    def __init__(self):
        self.poses = None

    def poses_found(self, result):
        """
            Interface.
        """

    def posesFound(self, request, _):
        logging.info("{} result:{}".format(jk.mech_vision, request))
        self.poses_found(request)

    def poses_planned(self, result):
        """
            Interface.
        """

    def posesPlanned(self, request, _):
        logging.info("Plan result:{}".format(request))
        self.poses_planned(request)

    def multiPickCombination(self, request, _):
        logging.info("multiPickCombination:{}".format(request))
```

The default service type is vision\_watcher, which cannot be modified. The default service name is vision\_watcher\_adapter. If multiple vision\_watcher services are needed in the project, you can modify the service\_name in the child class to distinguish different services.

The descriptions of class functions are shown in the table below.

Class Function	Description
poses_found()	Interface function whose child class can be overridden; the parameter is the recognition result of Mech-Vision
poses-Found()	Parse the recognition result sent from Mech-Vision; child class function usually does not need to be overridden
poses_planned()	Interface function; the parameter is the vision point in the path planned by Mech-Viz
poses-Planned()	Parse the planning message sent from Mech-Viz

## RobotService Class

The RobotService class is shown below.

```
class RobotService(JsonService):
    service_type = "robot"
    service_name = "robot"
    jps = [0, 0, 0, 0, 0, 0]
    pose = [0, 0, 0, 1, 0, 0, 0]

    def getJ(self, *_):
        return {"joint_positions": self.jps}

    def setJ(self, jps):
        logging.info("setJ: {}".format(jps))
        self.jps = jps

    def getL(self, *_):
        return {"tcp_pose": self.pose}

    def getFL(self, *_):
        return {"flange_pose": self.pose}

    def setL(self, pose):
        logging.info("setL: {}".format(pose))
        self.pose = pose

    def moveXs(self, params, _):
        pass

    def stop(self, *_):
        pass

    def setTcp(self, *_):
        pass

    def setDigitalOut(self, params, _):
        pass

    def getDigitalIn(self, *_):
        pass

    def switchPauseContinue(self, *_):
        pass
```

The default service type is **robot**, which cannot be modified. The default service name is **robot**, which should be modified into the name of the real robot in the child class. You will need to set a home position in JPs or TCP in the child class. Please make sure that this pose will not cause collision with the scene objects.

The descriptions of class functions are shown in the table below.

Class Function	Description
getJ()	Return JPs to Mech-Viz/Mech-Vision
setJ()	Set JPs in radians for external srvcies
getL()	Return TCP to Mech-Viz/Mech-Vision
getFL()	Return flange pose to Mech-Viz/Mech-Vision
setL()	Set flange pose in quaternions for external srvcies; the unit is meter
moveXs()	This function will be called after Mech-Viz finishes planning the path. The parameter contains property of targets from move Tasks. Please note that if Tasks that may interrupt pre-planning, such as check_di, branch_by_msg, Mech-Viz will call this function for multiple times.
stop()	Stop the robot (this function is seldom called)
setTcp()	Set TCP (this function is seldom called)
setDigitalOut()	Set DO (this function is seldom called)
getDigitalIn()	Obtain DI (this function is seldom called)
switch-PauseContinue()	Pause/continue the robot (this function is seldom called)

### OuterMoveService Class

The code of OuterMoveService class is shown below.

```
class OuterMoveService(JsonService):
    service_type = "outer_move"
    service_name = "outer_move"
    move_target_type = TCP_POSE
    velocity = 0.25
    acceleration = 0.25
    blend_radius = 0.05
    motion_type = MOVEJ
    is_tcp_pose = False
    pick_or_place = 0

    def __init__(self):
        self.targets = []

    def gather_targets(self, di, jps, flange_pose):
        """
        Interface.

        Please add targets to `self.targets` here if needed.
        """

    def add_target(self, move_target_type, target):
        self.targets.append({"move_target_type": move_target_type, "target": target})
```

(continues on next page)

(continued from previous page)

```

def getMoveTargets(self, params, *_):
    """
    @return: targets(move_target_type 0:jps, 1:tcp_pose, 2:obj_pose)
            velocity(default 0.25)
            acceleration(default 0.25)
            blend_radius(default 0.05)
            motion_type(default moveJ 'J':moveJ, 'L':moveL)
            is_tcp_pose(default False)
    """
    di = params["di"]
    jps = params["joint_positions"]
    flange_pose = params["pose"]
    logging.info("getMoveTargets: di={}, jps={}, flange_pose={}".format(di, jps, flange_
    ↪pose))

    self.gather_targets(di, jps, flange_pose)
    targets = self.targets[:]
    self.targets.clear()
    logging.info("Targets: {}".format(targets))
    ↪return {"targets": targets, "velocity": self.velocity, "acceleration": self.
    ↪acceleration, "blend_radius": self.blend_radius,
            "motion_type": self.motion_type, "is_tcp_pose": self.is_tcp_pose, "pick_or_
    ↪place": self.pick_or_place}
  
```

Both the default service type and service name are outer\_move. If multiple outer\_move services are needed in the project, you can modify the service\_name in the child class to distinguish different services.

The descriptions of class functions are shown in the table below.

Class Function	Description
move_target_type()	Type of the target; 0:jps, 1:tcp_pose, 2:obj_pose.
velocity()	Speed of the target; the default value is 0.25.
acceleration()	Acceleration of the target; the default value is 0.25.
blend_radius()	The turning radius of the target; the default value is 0.05m.
motion_type()	The motion type of the target: 'J':moveJ, 'L':moveL.
is_tcp_pose()	Whether the pose of the target is in TCP.
gather_targets()	Interface function, which collects all the targets. The parameter contains current JPs, flange pose, and DI value of the robot. The child class can be overridden according to actual requirements.
add_target()	Add a target; this function can be called in the child class to add a target.
getMoveTargets()	This function will be called when Mech-Viz proceeds on the outer_move Task. The parameter contains current JPs, flange pose, and DI value of the robot.

## Register Service

The services corresponding to the above 4 classes are only available after being registered. The function used for service registration is shown below.

```
def register_service(hub_caller, service, other_info=None):
    server, port = start_server(service)
    if service.service_type == "robot":
        other_info["from_adapter"] = True
        other_info["simulate"] = False
    hub_caller.register_service(service.service_type, service.service_name, port, other_info)
    return server, port
```

## Adapter util Package

Adapter util package is stored in `/src/util` of Mech-Center's installation directory. It contains various modules and some general functions. During programming, you can check the util package and see if there is an existing function that you can use directly. If there is not such function, but it is a commonly-used one, you can abstract the function and add it into the util package.

The introductions of each module are as follows.

- *database Module*
- *json\_keys Module*
- *message\_box Module*
- *timestamp Module*
- *transforms Module*
- *util\_file Module*
- *timer Module*
- *pose Module*

### database Module

database module contains functions about database. A `mechmind.db` file will be created by default when running Mech-Center, and it is used to store the running log. database Module provides SQL statements which can be used to query one or all recorded entries.

### json\_keys Module

json\_keys module stores all JSON key/value strings used in Mech-Center, which can be directly imported and used in other modules.

### message\_box Module

message\_box module provides functions about popup boxes, which can be divided into types as information, warning, and critical.

### timestamp Module

timestamp module enables to return the current timestamp.

### transforms Module

transforms module provides functions used for converting Euler angles to quaternions, converting quaternions to Euler angles, pose multiplication, converting object pose to TCP, converting TCP to object pose, and calculation of object rotation. The third-party library transforms3d also provides functions as converting Euler angles to quaternions and converting quaternions to Euler angles. However, in actual use, the conversion value output by transforms3d can be wrong in some cases. In practice, you can use transforms3d library first. If the output result is incorrect, the custom conversion functions provided in transforms module can be used.

### util\_file Module

util\_file module provides functions of file reading and writing, such as reading and writing JSON file.

### timer Module

The timer module provides a convenient timer class. When a timing function is required, you can generate a Timer object, pass it into the callback function, and then call start(). There is no need to destroy the Timer object after use, as it will be destroyed automatically when the program exits.

### pose Module

The pose module provides a class with the same expression as mECH-Viz, including translation (in meters) and rotation (in quaternions). It can perform inversion and multiplication operations, and provide conversion from list or to list. In addition, several small unit conversion functions about pose are also provided, including millimeters to meters, meters to millimeters, radians to degrees, degrees to radians, quaternions to Euler angles, and Euler angles to quaternions.

---

In the end, you can implement abstract parent interface according to requirements, and therefore establish communications with Mech-Vision, Mech-Viz, and external devices.



## Get Interface

- *Get Tasks used in the current Mech-Viz project*
- *Get parameters in Mech-Viz or Mech-Vision project*

### Get Tasks used in the current Mech-Viz project

The function to get Tasks used in the current Mech-Viz project is as follows.

```
def get_viz_task_names(self, msg={}, timeout=None):
    result = self.call_viz("getAllTaskNames", msg, timeout)
    logging.info("Property result: {}".format(json.loads(result)))
    return result
```

After `get_viz_task_names()` is called, a string in JSON format that represents all obtained Tasks will be returned.

### Get parameters in Mech-Viz or Mech-Vision project

The function to get parameters in Mech-Viz or Mech-Vision project is as follows.

```
def get_property_info(self, msg={}, get_viz=True, timeout=None):
    result = (self.call_viz if get_viz else self.call_vision)("getPropertyInfo", msg, timeout)
    logging.info("{} property result: {}".format("Viz" if get_viz else "Vision", json.
↵loads(result)))
    return result
```

When the function is called with no "type" specified for the `msg` parameter, all parameters will be obtained. If a certain type is specified, only parameters of the corresponding Task/Step will be obtained. For example, after `get_property_info(msg={"type": "move"})` is called, a string in JSON format that represents parameters of the move Task will be returned.

## Mech-Vision Interface

This section introduces Adapter interfaces related to Mech-Vision, as listed below.

- *Get Vision Target(s)*
- *Set Step Parameters*
- *Read Step Parameters*
- *Switch Parameter Recipe*

## Get Vision Target(s)

The vision result from Mech-Vision can be obtained by calling the `find_vision_pose()` function in `adapter.py`.

```
def find_vision_pose(self, project_name=None, timeout=default_vision_timeout):
    vision_result = self.call_vision("findPoses", project_name=project_name, timeout=timeout)
    logging.info("Find vision result: {}".format(vision_result))
    return vision_result
```

The vision results output by Mech-Vision are usually object poses (`obj_pose`) in quaternions. If you need the vision results in TCP, please adjust the settings in Mech-Vision.

### Example

A function needs to be created in Adapter to convert object poses into TCP (`tcp_pose`), and convert the quaternions into Euler angles and radians into degrees, and the converted data will be packed and send to the robot. The data format and unit should be consistent with those used on the robot side. Besides, Adapter can check the vision points output by Mech-Vision, as shown below.

```
def check_vision_result(self, vision_result):
    if vision_result["noCloudInRoi"]:
        # Determine whether it is an empty
    ↪ bin
    logging.info("Layer has no objects")
    self.send(pack('>2B6i', CODE_NO_CLOUD, vision_num, *EMPTY_PLACEHOLDER))
    return
    poses = vision_result.get("poses", [])
    if len(poses) == 0:
        # Get pose
        # Determine whether there is a
    ↪ vision point
    logging.warning("No pose from vision")
    self.send(pack('>2B6i', CODE_NO_POSE, vision_num, *EMPTY_PLACEHOLDER))
    return
    self.send(pack_pose(poses[0], vision_num))
    # Send after format conversion
```

In the above example, `vision_result` is obtained from `find_vision_pose()`, and the statement is shown below.

```
self.check_vision_result(json.loads(self.find_vision_pose().decode()))
```

After `vision_result` is passed into the `check_vision_result` function, for projects with an ROI, the function will determine if it is an empty bin first, and then obtain the vision point. If there is nothing wrong with the vision point, it will be passed into the `pack_pose()` function and send out.

## Set Step Parameters

To set Step parameters in Mech-Vision dynamically, usually only the `set_step_property()` function in the `adapter.py` need to be called.

```
def set_step_property(self, msg, project_name=None, timeout=None):
    return self.call_vision("setStepProperties", msg, project_name, timeout)
```

In the above statement, `msg` determines the specific Step name and the parameter which need to be configured.

### Example

You can create a function as shown below to set `msg` when the point cloud model file need to be set dynamically according to different types of workpieces.

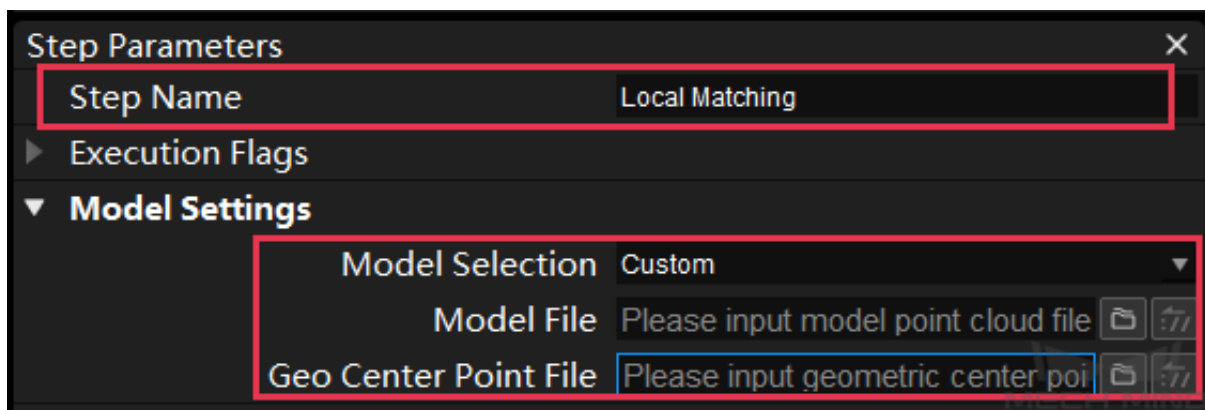
```
def _step_matching_model_cell(step_name, model_type):
    msg = {"name": step_name,
          "values":
            {"modelFile": model_type["ply"],
             "pickPointFilePath": model_type["json"]}}
    return msg
```

`step_name` is the name of the Step to be configured; `model_type` is the file directory of the corresponding workpiece. The directory of point cloud model file in PLY and JSON format can be obtained via `ply` and `json`, and they will be assigned to the parameters in corresponding Step.

If the `step_name` is "Local Matching", then the statement is shown below.

```
msg = _step_matching_model_cell("Local Matching", model_type)
self.set_step_property(msg)
```

The Step Parameters panel in Mech-Vision is shown in the figure below.



### Read Step Parameters

You can obtain the parameters of a certain Step by calling `read_step_property()` in `adapter.py`.

```
def read_step_property(self, msg):
    result = self.call_vision("readStepProperties", msg)
    logging.info("Property result: {}".format(result))
    return result
```

In the above function, `msg` determines the Step name and the parameters to be obtained. You can create a function to rewrite the `msg`.

#### Example

An example to obtain the camera IP address is shown below.

```
def read_camera_property(self):
    msg = {"type": "Camera",
          "properties": ["MechEye"]}
```

(continues on next page)

(continued from previous page)

```
property_results = json.loads(self.read_step_property(msg).decode())
camera_ip = property_results["MechEye"]["NetCamIp"]
```

If only one camera is used in the Mech-Vision project, the Step can be found according to **type**. If there are multiple Steps of the same type in the project, and you would like to obtain or configure parameters of a certain Step, the **name** can be used to find the Step. All parameters of the camera can be obtained by calling the `read_step_property()` function and converting the data into JSON format.

```
Property result:
{
  "MechEye": {
    "NetCamIp": "127.0.0.1",
    "TimeOut": "10",
    "configGroup": "",
  }
}
```

In this example, the camera IP address (127.0.0.1) is obtained according to the specific parameter field "MechEye" and "NetCamIp".

### Switch Parameter Recipe

When using Mech-Vision, you can switch parameter recipes between different projects by calling the `select_parameter_group()` function in `adapter.py` if the workflow of the projects are the same, but the parameters are configured differently.

```
def select_parameter_group(self, project_name, group_index, timeout=None):
    msg = {"parameter_group_idx": group_index}
    result = self.call_vision("selectParameterGroup", msg, project_name, timeout)
    logging.info("selectParameterGroup result: {}".format(result))
    return result
```

In the above function, `project_name` is the Mech-Vision project name, and `group_index` is the number of recipe.

### Example

When the parameter recipe need to be switched in a project, you can call the `select_parameter_group()` function as shown below and use it for error handling.

```
try:
    result = self.select_parameter_group(self.vision_project_name, model_code-1)
    if result:
        result = result.decode()
        if result.startswith("CV-E0401"):
            return -1
        elif result.startswith("CV-E0403"):
            return -1
        raise RuntimeError(result)
    except Exception as e:
        logging.exception('Exception when switch model: {}'.format(e))
        return -1
    return 0
```

In this example, `self.vision_project_name` passes the Mech-Vision project name, and `model_code-1` passes the number of the recipe.

### Mech-Viz Interface

This section introduces Adapter interfaces related to Mech-Viz, as listed below.

- *Start Mech-Viz Project*
- *Stop Mech-Viz Project*
- *Pause and Continue Mech-Viz*
- *Set Task Parameters*
  - *move*
  - *move\_list/move\_grid*
  - *outer\_move*
  - *Pallet*
  - *branch\_by\_msg*
  - *counter*
- *Read Task Parameters*
- *Set TCP*
- *Set Velocity*
- *Set Point Cloud Collision Parameters*
- *Mech-Viz Return Value*

### Start Mech-Viz Project

A function to start Mech-Viz project has been defined in the Adapter class in the `adapter.py` file, and therefore you can directly call the `start_viz()` function. Besides, you can rewrite `self.before_start_viz()` and `self.after_start_viz()` according to actual needs of the project to define the process before and after starting the Mech-Viz project.

#### Function Definition

```
def start_viz(self, in_new_thread=True, timeout=None):
    if not self.is_viz_registered():
        logging.error("{} has not registered in {}".format(jk.mech_viz, jk.mech_center))
        self.code_signal.emit(ERROR, VIZ_NOT_REGISTERED)
        self.viz_finished_signal.emit(True)
        self.viz_not_registerd()
        return False
    if self.is_viz_in_running():
        logging.info("{} is already running.".format(jk.mech_viz))
```

(continues on next page)

(continued from previous page)

```

        self.code_signal.emit(WARNING, VIZ_IS_RUNNING)
        self.viz_finished_signal.emit(False)
        self.viz_is_running()
        return False
    self._read_viz_settings()
    if not self.viz_project_dir:
        self.msg_signal.emit(ERROR, _translate("messages", "The project of {0} is not
↔registered. Please make sure Autoload Project is selected in {0}.").format(jk.mech_viz))
        self.viz_finished_signal.emit(True)
        return False
    msg = {"simulate": self.is_simulate, "project_dir": self.viz_project_dir}
    if self.is_keep_viz_state:
        msg["keep_exec_state"] = self.is_keep_viz_state
    if self.is_save_executor_data:
        msg["save_executor_data"] = self.is_save_executor_data
    self.before_start_viz()
    self.viz_finished_signal.emit(False)
    if in_new_thread:
        threading.Thread(target=self.wait_viz_result, args=(msg, timeout)).start()
    else:
        self.wait_viz_result(msg, timeout)
    self.after_start_viz()
    return True

```

By default, `start_viz()` will wait for Mech-Viz to finish execution in a new thread, which is to avoid affecting operations other than starting Mech-Viz project.

The example below shows how to set Step parameters dynamically by calling the `self.before_start_viz()` function.

```

def before_start_viz(self):
    self.set_move_offset(x, y, z)

```

Before starting Mech-Viz project, the X, Y, or Z-axis offset of a target will be configured according to the read data.

### Stop Mech-Viz Project

A function to stop Mech-Viz project has been defined in the Adapter class in the `adapter.py` file, and therefore you can directly call the `stop_viz()` function.

#### Function Definition

```

def stop_viz(self, timeout=None):
    if not self.is_viz_registered():
        self.code_signal.emit(WARNING, VIZ_NOT_REGISTERED)
        return False
    self.call_viz("stop", timeout=timeout)
    self.code_signal.emit(INFO, VIZ_STOP_OK)
    return True

```

## Pause and Continue Mech-Viz

A function to pause and continue Mech-Viz project has been defined in the Adapter class in the `adapter.py` file. The `pause_viz` function corresponds to the Pause/Continue button in Mech-Viz and can only be used for simulation.

### Function Definition

```
def pause_viz(self, msg, timeout=None):
    if not self.is_viz_registered():
        self.code_signal.emit(WARNING, ADAPTER_CANCEL_PAUSE)
        return
    self.call_viz("switchPauseContinue", msg, timeout)
    self.code_signal.emit(INFO, ADAPTER_PAUSE_VIZ if msg.get(
        "to_pause") else ADAPTER_CONTINUE_VIZ)
```

## Set Task Parameters

Usually, you can directly call the `set_task_property()` function in the Adapter class to set Task parameters dynamically.

### Function Definition

```
def set_task_property(self, msg, timeout=None):
    return self.call_viz("setTaskProperties", msg, timeout)
```

In the above function, `msg` determines the parameters configured in different Tasks.

## move

When running a Mech-Viz project, sometimes you need to fine-tune the X-axis, Y-axis, and Z-axis offset in the move Task, which can be enabled by adding the function below in the main program.

### Example

```
def set_move_offset(self, name, x_offset, y_offset, z_offset):
    msg = {"name": name,
          "values": {"xOffset": x_offset / UNIT_PER_METER,
                    "yOffset": y_offset / UNIT_PER_METER,
                    "zOffset": z_offset / UNIT_PER_METER}}
    self.set_task_property(msg)
```

In the above example, `name` is the name of the move Task and `UNIT_PER_METER` is 1000. Since the unit of `x_offset`, `y_offset`, and `z_offset` is usually millimeter, and the data unit used in Mech-Viz is meter, you need to use `UNIT_PER_METER` to convert the unit.

If the following `set_move_offset()` function is called, the X-axis, Y-axis, and Z-axis offset in the "move\_1" Task will be changed accordingly.

```
self.set_move_offset("move_1", 100, 200, 300)
```

## move\_list/move\_grid

The move\_list and move\_grid Tasks should be configured in Mech-Viz in advance, and then Adapter will modify the index according to the logic. The method to create and call the function is the same as those of the move and pallet interface functions.

## outer\_move

The outer\_move Task can be used when multiple targets from external services are sent to Mech-Viz for path planning. You can set JPs, TCP, and the object pose in the Task, as shown below.

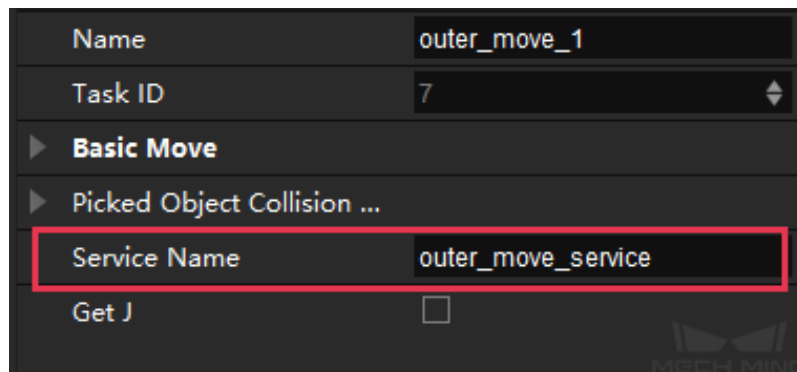
### Example

```
class CustomOuterMoveService(OuterMoveService):
    def gather_targets(self, di, jps, flange_pose):
        self.add_target(self.move_target_type, [0.189430,-0.455540,0.529460,-0.079367,0.294292,
        ↵↵-0.952178,0.021236])
```

When Mech-Viz executes to the outer\_move Task, the function `getMoveTargets()` will be called. Different outer\_move Tasks can be distinguished by their service names.

```
def _register_service(self):
    self.outer_move_service = CustomOuterMoveService()
    self._outer_move_server, port = register_service(outer_move_service, port)
```

The Setting Parameters panel of the outer\_move Task in Mech-Viz is shown in the figure below.



**Attention:** A Task to define picking should be put before the outer\_move Task. Otherwise, an error message “- The posture of the object is invalid when the object is not held!” will pop up.



## Pallet

When running a Mech-Viz project, sometimes you need to configure parameters for different pallet-type Tasks. The specific Task to be configured can be found according to the name of the pallet-type Task. All parameters displayed in the Task parameters panel in Mech-Viz can be configured.

### Example

For example, the parameters **Current Index** and **File Name** for the Task `custom_pallet_pattern` usually need to be modified, and you can add a function as shown below in the main program.

---

**Hint:** The **Dir Path** and **File Name** will be displayed only when the **Dynamic Load** is checked.

---

```
def set_stack_pallet(self, name, curIndex, fileName):
    msg = {
        "name": name,
        "values": {
            "curIndex": curIndex,
            "fileName": fileName,
        }
    }
    self.set_task_property(msg)
```

In the above example, "curIndex" corresponds to the parameter name **Current Index**, and "fileName" corresponds to the parameter name **File Name** in the Task parameters panel in Mech-Viz.

The statement as shown below is used to call the `set_stack_pallet()` function.

```
self.set_stack_pallet("common_pallet_1", 2, "re.json")
```

For the Task `predefined_pallet_pattern`, the parameter values of **Current Index**, **Pallet Type**, **Carton Length**, **Carton Width**, **Carton Height**, **Rows/Unit**, **Cols/Unit**, and **Layer Count** usually need to be modified. You can add a function as shown below in the main program.

```
def set_stack_pallet(self, name, curIndex, stack_type):
    pallet_info = self.box_data_info[stack_type]
    """
    pallet_info: Length(mm), Width(mm), Height(mm), pallet type, rows, columns, layers
    """
    msg = {
        "function": "setTaskProperties",
        "name": name,
        "values": {
            "curIndex": curIndex,
            "palletType": pallet_info[3],
            "cartonLength": pallet_info[0] / UNIT_PER_METER,
            "cartonWidth": pallet_info[1] / UNIT_PER_METER,
            "cartonHeight": pallet_info[2] / UNIT_PER_METER,
            "cylinderRows": pallet_info[4],
            "cylinderCols": pallet_info[5],
            "layerNum": pallet_info[6]
        }
    }
    self.set_task_property(msg)
```

Since there are multiple parameters need to be configured, you will usually write the parameters into an Excel file first, and then the data in the Excel file can be read by the program and recorded into the `self.box_data_info`, and the index of `stack_type` can be used to obtain certain parameter values of the Task.

When comparing the `msg` value in the `custom_pallet_pattern` and `predefined_pallet_pattern` class, you can find that the biggest difference is the "values". If you need to configure the parameters of a specific Task, you can add corresponding parameter names and values in "values". You can refer to the above examples to configure parameters of other pallet-type Tasks.

### branch\_by\_msg

When the Mech-Viz project executes to the `branch_by_msg` Task, it will wait for an external signal sent by Adapter to specify an exit port to take. You can use the function as shown below to control the `branch_by_msg` Task.

#### Example

```
def set_branch(self, name, area):
    time.sleep(1) # The delay of 1s here is to wait for the Mech-Viz executor to fully start
    try:
        info = {"out_port": area, "other_info": []}
        msg = {"name": name,
              "values": {"info": json.dumps(info)}}
        self.set_task_property(msg)
    except Exception as e:
        logging.exception(e)
```

In the above example, `name` is the name of the `branch_by_msg` Task, `area` specifies the exit port. The serial number of the exit port starts from 0, and will be added one if a port is added. If the Task is asked to take the first port on the left, then `area=0`. If the `set_branch` function is called before starting a Mech-Viz project, an "no actuator" error message will be returned.

### counter

When using the counter Task, you will need to set the total count and current count. The function used to configure parameters in this Task is shown below.

#### Example

```
def set_counter_property(self, name, count, curCount):
    msg = {"name": name,
          "values": {"count": count, "currentCount": curCount}}
    self.set_task_property(msg)
```

The statement to call the function is shown below.

```
self.set_counter_property("counter_1", 5, self.success_stack_num)
```

`self.success_stack_num` is the number of successfully palletized cartons. If a carton falls during palletizing, and Mech-Viz is stopped with manual intervention, the value of "currentCount" will not be stored in "counter\_1" Task. After restarting Mech-Viz, the current count of the counter Task can be reset via `self.success_stack_num`.

## Read Task Parameters

If you need to read parameters of a specific Task when running the Mech-Viz project, you can add a function as shown below in the main program.

### Example

```
def read_move_pallet(self, name):
    msg = {"name": name,
          "properties": ["xOffset", "yOffset", "zOffset", ]}
    return read_task_property(msg)
```

In the above example, "name" is used to locate the Task whose parameters need to be read. The parameters after "properties" can be modified according to actual needs. In this example, the values of parameters "xOffset", "yOffset", and "zOffset" need to be read. The statement to call the function is shown below.

```
self.read_move_pallet("move_3")
```

The result is shown below.

```
{'zOffset': -0.23, 'xOffset': -0.12, 'yOffset': -0.15}
```

## Set TCP

You only need to specify the corresponding index in Mech-Viz's end effector list to set the TCP. The end effector list index in Mech-Viz starts from 0, and the index number is an integer. Do not use an index number out of the boundary. The function to set TCP is shown below.

### Example

```
def set_tcp(self, index):
    msg = {"function": "setTcp", "index": index}
    self.call("executor", msg)
```

## Set Velocity

To adjust the robot velocity dynamically, you can add a function as shown below in the main program.

### Example

```
def set_vel(self, vel_scale):
    msg = {"function": "setConfig",
          "velScale": vel_scale / 100, "accScale": vel_scale / 100}
    self.call("executor", msg)
```

If the velocity needs to be set as 80%, the following function can be called.

```
self.set_vel(80)
```

**Attention:** This function must be called after starting the Mech-Viz project, or else an error may occur. The prerequisite for this function is the same as that of `set_branch()`. Therefore, the `set_vel()` function is usually called in the `set_branch()` function instead of calling it in a new thread. An example is shown below.

```
def set_branch(self, name, area):
    time.sleep(1)
    if self.box_data_info[int(self.pallet_info)][7] <= 10:
        self.set_vel(100)
    else:
        self.set_vel(80)
    try:
        info = {"out_port": area, "other_info": []}
        msg = {"function": "setTaskProperties",
              "name": name,
              "values": {"info": json.dumps(info)}}
        self.set_task("executor", msg)
    except Exception as e:
        logging.exception(e)
```

### Set Point Cloud Collision Parameters

The `setConfig()` interface function corresponds to the parameters related to collision detection in Mech-Viz. The method to set cloud collision parameters interface is the same as that of setting velocity, and the main difference is the `msg` value. An example is shown below.

#### Example

```
msg = {}
msg["function"] = "setConfig"
msg["check_pcl_collision"] = True
msg["collided_point_thre"] = 5000
msg["collide_area_thre"] = 20
msg["pcl_resolution_mm"] = 2
self.call("executor", msg)
```

## Mech-Viz Return Value

The return values of Mech-Viz are as shown in the table below.

Return Value	Description
Finished	The execution has finished properly. Please note that when vision_move Task takes the exit port on the right (Other failures), this value will be returned as well.
Command Stop	The Stop button has been clicked on or the <b>stop_viz()</b> function has been called.
No targets	There were no vision points returned by Mech-Vision.
No proper vision poses	The vision point was unreachable as the robot could not move to that point or the robot would collide with others.
PlanFail	Mech-Viz failed to plan the path.
SceneCollision	A collision was detected.

According to the value returned by Mech-Viz, corresponding interface functions are provided in base class of Adapter.

## RobotService Interface

This section introduces Adapter interfaces related to RobotService, as listed below.

- *Simulate RobotServer Service*
- *getJ*
- *getFL*
- *moveXs*

## Simulate RobotServer Service

Simulate RobotServer Service is usually used in scenarios where the robot is full-controlled by Mech-Mind, but the robot is not directly controlled via RobotServer. Instead, the RobotService in Adapter simulates RobotServer and therefore is used to control the robot. After RobotService is registered in Mech-Center, Mech-Viz will interact with RobotService as it interacts with the real robot.

Simulated RobotServer service can be used to:

- Obtain JPs of the robot for computation in Mech-Vision (in Eye In Hand mode)
- Obtain the pose from the real robot and send it RobotService

### Example

The RobotService object in a child class in Adapter is called as follows:

```

def _register_service(self):
    """
    register_service
    :return:
    """
    if self.robot_service:
        return

    self.robot_service = RobotService(self)
    other_info = {'robot_type': self.robot_service.service_name}
    self.server, _ = register_service(self.hub_caller, self.robot_service, other_info)

    self.robot_service.setJ([0.0, 0.0, 0.0, 0.0, 0.0, 0.0])
    
```

The RobotService class is shown below.

```

class RobotService(JsonService):
    service_type = "robot"
    service_name = "robot"
    jps = [0, 0, 0, 0, 0, 0]
    pose = [0, 0, 0, 1, 0, 0, 0]

    def getJ(self, *_):
        return {"joint_positions": self.jps}

    def setJ(self, jps):
        logging.info("setJ:{}".format(jps))
        self.jps = jps

    def getL(self, *_):
        return {"tcp_pose": self.pose}

    def getFL(self, *_):
        return {"flange_pose": self.pose}

    def setL(self, pose):
        logging.info("setL:{}".format(pose))
        self.pose = pose

    def moveXs(self, params, _):
        pass

    def stop(self, *_):
        pass

    def setTcp(self, *_):
        pass

    def setDigitalOut(self, params, _):
        pass

    def getDigitalIn(self, *_):
        pass

    def switchPauseContinue(self, *_):
        pass
    
```

## getJ

getJ() is used to obtain the current JPs in Mech-Viz and Mech-Vision. Usually, the current JPs are set in setJ() first and then getJ() will be called.

### Example

```
def getJ(self, *_) :
    pose = {"joint_positions": self._jps}
    return pose
```

1. Eye In Hand: write the JPs sent by the robot into setJ().

```
def setJ(self, jps):
    assert len(jps) == 6
    for i in range(6):
        jps[i] = deg2rad(float(jps[i]))
    self._jps = jps
    logging.info("SetJ: {}".format(self._jps))
```

jps is the JPs sent by the robot, its value will be assigned to `self._jps`, which will be called by the `getJ()` function. Since the `getJ()` requires data in radians, please note unit conversions here.

2. Eye To Hand: It is not necessary to set the current JPs according to the pose of the real robot. However, a Home position that will not lead to collision needs to be set to simulate the real robot, or else the value will be assigned randomly and an error may occur.

```
def getJ(self, *_) :
    return {"joint_positions": [1.246689,-0.525868,-0.789761,-1.330814, 0.922581, 4.
↪364021]}
```

## getFL

getFL() provides the flange pose when the robot captures images. Since in Eye In Hand mode, the calibrated positional relationship is between the flange and camera, and the data finally used is in the robot base reference frame, the flange pose when the robot captures images is needed.

```
def getFL(self, *_) :
    return {"flange_pose": self.pose}
```

Please note the following details when using Eye In Hand.

1. If the robot returns image capture pose in JPs, you can directly call `setJ()` in RobotService (the unit is radians) and the function will return to [].
2. If the robot returns image capture pose in flange pose, then:
  - make sure that `is_eye_in_hand` in the external parameter file "`extri_param.json`" in the Mech-Vision project is set to `True`
  - call `setFL()` of RobotService (the pose is represented by quaternions and the unit is meter)

## moveXs

RobotService uses `moveXs()` to receive targets of the path planned by Mech-Viz.

### Function Definition

```
def moveXs(self, params, _):
    with self.lock:
        for move in params["moves"]:
            self.targets.append(move)
    return {"finished": True}
```

`params` passes all the parameters in the Mech-Viz project. All target poses, including targets from `vision_move`, `relative_move`, etc., can be obtained via `params["moves"]`. The returned poses are in JPs by default, and will be passed into `self.targets` for subsequent calls.

### Example

Usually, this function is used together with `notify` function. When Adapter receives a message, it will pass `self.targets` into the function and send it to the robot after converting and packing.

```
def notify(self, request, _):
    msg = request["notify_message"]
    logging.info("{} notify message: {}".format(self.service_name, msg))
    if msg == "started":
        with self.lock:
            self.move_targets.clear()
    elif msg == "finished":
        with self.lock:
            targets = self.move_targets[:]
            self.move_targets.clear()
        self.send_moves(targets)
```

When the notify message "started" is received, the targets in the list will be cleared, which aims to avoid duplicate targets caused by unexpected disconnection and restart of Mech-Viz. When the notify message "finished" is received, the target list will be passed into the `pack_move` function, which will consolidate the data and then send out the data.

```
def pack_move(self, move_param):
    move_list = []
    for i, move in enumerate(move_param):
        target = move["target"]
        move_list.append(target)
    logging.info("move list num: {}".format(len(move_list)))
    logging.info("move list: {}".format(*move_list))
    motion_cmd = pack('>24f', *move_list)
    self.send(motion_cmd)
```

According to actual needs of the project, you can choose to send all targets planned by Mech-Viz or select several targets from the list to send according to the index. `pack_move()` usually consolidates the data according to the specific format required by different robot brands, and the required data format is usually specified in the communication protocol in advance.



## Other Interfaces

This section introduces other interfaces of Adapter, as listed below.

- *Notify Service*
- *VisionWatcher Service*

## Notify Service

When a Mech-Viz project proceeds to a particular branch or Task, and a corresponding function in Adapter program needs to be called, you can add a notify\_viz Task in the project.

### Example

For example, a function is written to increase the number of depalletized parts in the Adapter program. Then we can add a notify\_viz Task after the last Task of the depalletizing process. When the project executes to the notify\_viz Task, the Adapter can be triggered to call the function. The example function is shown below.

1. Create a class that inherits the parent class **NotifyService**.

```

from interface.services import NotifyService, register_service

class NotifyService(NotifyService):
    service_type = "notify"
    service_name = "FANUC_M410IC_185_COMPACT"

    def __init__(self, update_success_num, update_fail_num):
        self.update_success_num = update_success_num
        self.update_fail_num = update_fail_num

    def handle_message(self, msg):
        if msg == "Success":
            self.update_success_num()
        elif msg == "Fail":
            self.update_fail_num()
    
```

When Mech-Viz executes to the notify\_viz Task and a message "Success" is sent, the Adapter will call the `update_success_num()` function, while a message "Fail" will trigger the Adapter to call the `update_fail_num()` function.

2. Instantiate the **NotifyService** class and register the service in the class that controls the main program of Mech-Viz.

```

class MyClient(TcpClientAdapter):

    def __init__(self, host_address):
        super().__init__(host_address)
        self._register_service()

    def _register_service(self):
        self.robot_service = NotifyService(self.update_success_num, self.update_fail_num)
    
```

(continues on next page)

(continued from previous page)

```

self.server, port = register_service(self.hub_caller, self.robot_service)

def update_success_num(self):
    # the num of unstack successfully plus 1
    self.success_num += 1

def update_fail_num(self):
    # the num of unstack fiplus 1
    self.fail_num += 1
    
```

3. Add notify\_viz Task in a proper position in the workflow of Mech-Viz.

Please note that the **Adapter Name** and **Message** in the notify\_viz Task correspond to **service\_name** and **msg** set in the **NotifyService** class, and therefore the parameters set in Mech-Viz should be configured accordingly.

Name	notify_1
Task ID	1
▼ <b>Basic Non-Move</b>	
Skip Execution	None
Out Port When Skip	0
▶ Receiver	
Adapter Name	FANUC_M410IC_185_COMPACT
Message	Fail
Action When Failed	Warning
Need Robot Stop	<input checked="" type="checkbox"/>
Timeout	1000 ms

After the project is executed, if the notify service is successfully registered, the **service\_type** and **service\_name** will be displayed in the interface of Mech-Center.

## VisionWatcher Service

After Mech-Vision finished running, some results will be output, such as vision result: { 'noCloudInRoi': False, 'function': 'posesFound', 'vision\_name': 'TJTVision-3'}. If some problems occur, Adapter can send out error messages via VisionWatcher service.

### Example

1. Create a class that inherits the class `VisionResultSelectedAtService`.

```
from interface.services import VisionResultSelectedAtService, register_service

class VisionWatcher(VisionResultSelectedAtService):
    def __init__(self, send_err_no_cloud):
        super().__init__()
        self.send_err_no_cloud = send_err_no_cloud

    def poses_found(self, result):
        has_cloud_in_roi = not result.get("noCloudInRoi", False)

        if not has_cloud_in_roi:
            time.sleep(2)
            self.send_err_no_cloud()
```

The child class `VisionWatcher` needs to override the `poses_found()` function of the parent class, so the logic of calling the function `send_err_no_cloud()` that sends error messages in Adapter will be changed in `poses_found()`. During operation, the value of vision result will be passed to the parameter `result` in `poses_found()`.

2. Instantiate the `VisionWatcher` in the class that controls the main program of Mech-Viz.

```
class MyClient(TcpClientAdapter):
    def __init__(self, host_address):
        super().__init__(host_address)
        self._register_service()

    def _register_service(self):
        self.robot_service = VisionWatcher(self.send_err_no_cloud)
        self.server, port = register_service(self.hub_caller, self.robot_service)

    def send_err_no_cloud(self):
        # send no cloud error message
        self.send("12,NoCloudErr,done".encode())
```

In the instantiation of `VisionWatcher` class, the `send_err_no_cloud()` function is passed to `VisionWatcher()` as a parameter. When there are no point clouds, according to the logic in `poses_found()`, the function that sends error messages will be called.

In running of the project, if the `VisionWatcher` service is successfully registered, it will be displayed in the interface of Mech-Center.

### 3.4.4 Adapter Programming Examples

After getting familiar with the Adapter programming syntax and logic, you can write your own Adapter program based on the programming examples provided in this section.

- *Use Only Mech-Vision to Send Vision Points*

#### Use Only Mech-Vision to Send Vision Points

This section elaborates an Adapter example program that used only Mech-Vision to send vision points. This section includes the following contents:

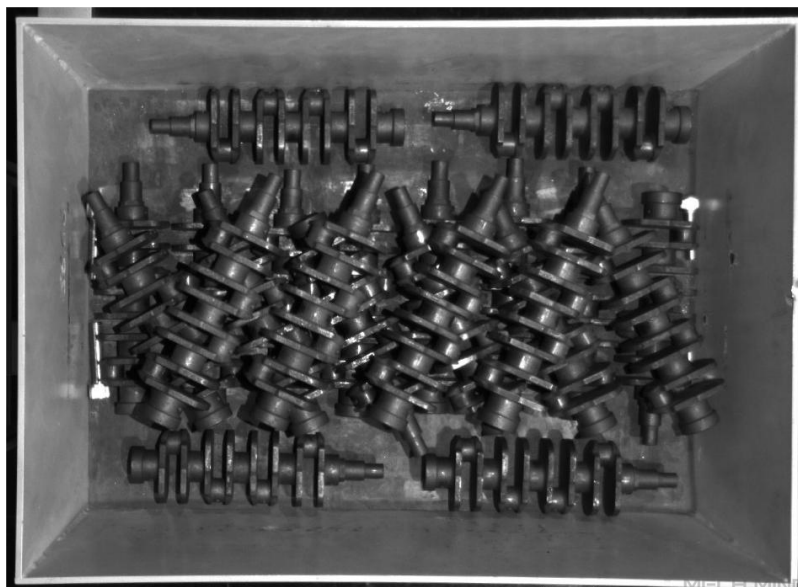
- *Background Introduction*
- *Communication Solution*
- *Packet Format*
- *Programming Guidelines*
- *Explanations to Example Program*

#### Background Introduction

This section introduces an Adapter example program written for a crankshaft feeding scenario, in which the camera was mounted on a bracket above the bin and Mech-Vision output the pose of the workobject to pick after capturing the image.

This Adapter example program worked with the Mech-Vision example project "Large Non-Planar Workpieces" (*File → View Example Projects → Machine Tending → Large Non-Planar Workpieces*).

This Mech-Vision example project adopted 3D model matching algorithms, and set different model files and pick points for different workobjects. Therefore, the parameter recipe needed to be set when the recipe ID (workobject ID) was sent together with the image capturing command from the robot side.

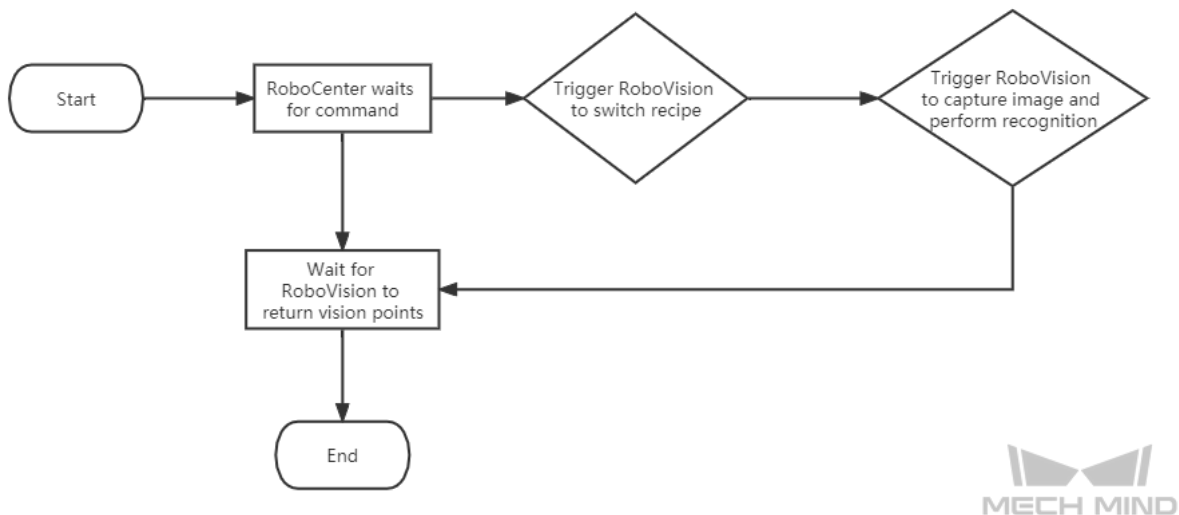


### Communication Solution

The robot and the industrial PC (IPC) installing Mech-Mind vision-series software communicated with each other through the TCP/IP Socket protocol. Data were exchanged in ASCII string format with the comma (,) as the separator.

During the communication, Mech-Mind vision-series software acted as the server while the robot as the client.

The following figure shows the communication process.



The communication process is detailed as follows:

1. Mech-Center waited for the robot to send the image capturing command P and the recipe ID.
2. Mech-Center triggered Mech-Vision to switch the parameter recipe.
3. Mech-Center triggered Mech-Vision to capture the image and perform recognition.
4. Mech-Vision returned the status code and the pose of the workobject to pick after successful image capturing and recognition.
5. Mech-Center returned the status code and the robot Tool Center Point (TCP) pose to the robot.

---

**Note:** To facilitate the robot's picking, Mech-Center converted the pose of the workobject to pick to the robot TCP.

---

### Packet Format

The following table shows the packet format used for communication.

	Request Command	Workobject ID
Sending (Robot -> IPC)	P	An integer, value range: 1~100
	Status Code	Vision Point (TCP)
Receiving (IPC -> Robot)	An integer, value range: 0~4; 0-Normal recognition; 1-Illegal command; 2-Vision project not registered; 3-No vision points; 4-No point clouds	6 floating-point values separated by comma (,) in the format of x,y,z,a,b,c

**Note:** The length of the response packet is fixed. If the status code in a response packet is a value from 1 to 4, the vision point data will be filled in with zeros (0).

### Packet Example

Request

P,1

Normal response

0,1994.9217,-192.198,506.4646,-23.5336,-0.2311,173.6517

**Note:** This example indicates that Mech-Vision successfully recognized the workobject and returned the TCP: 1994.9217,-192.198,506.4646,-23.5336,-0.2311,173.6517.

Abnormal response: illegal command

1,0,0,0,0,0,0

Abnormal response: Vision project not registered

2,0,0,0,0,0,0

Abnormal response: No vision points

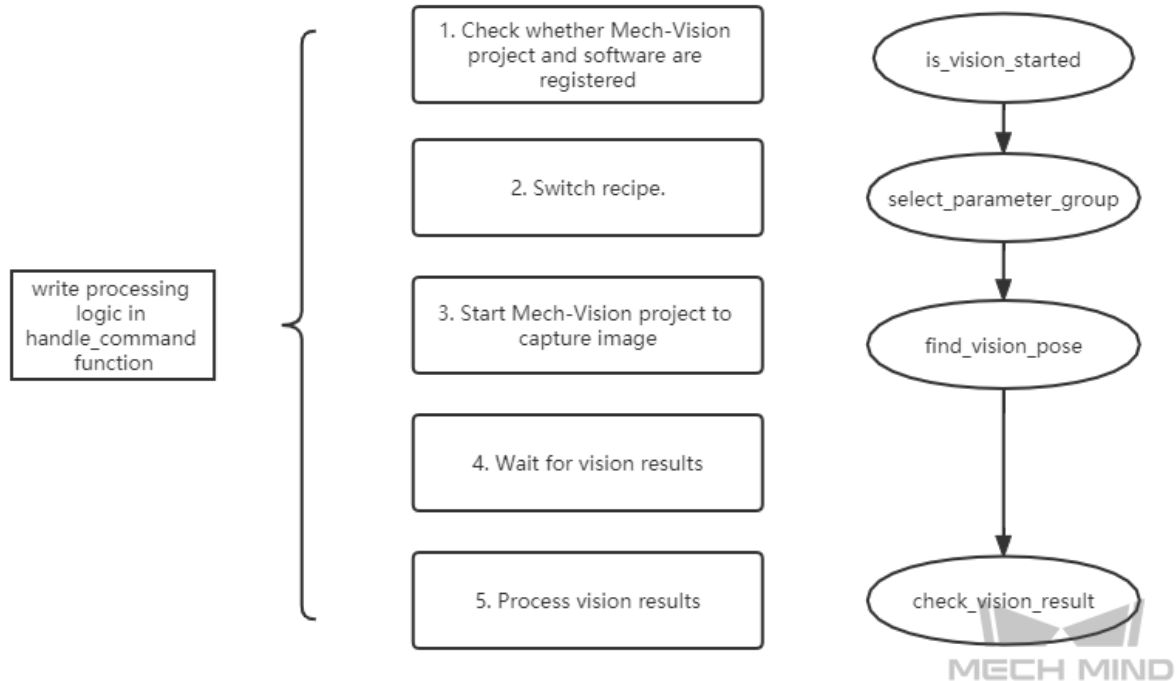
3,0,0,0,0,0,0

Abnormal response: No point clouds

4,0,0,0,0,0,0

### Programming Guidelines

To fulfill project objectives, the Adapter example program has been written according to the following guidelines.



The preceding figure lists only the core packet processing logic. The next section will explain the Adapter example program in detail.

### Explanations to Example Program

---

**Note:** You can download the Adapter example program [here](#) .

---

#### Import Python packages

Import all modules depended by the Adapter program.

```
import json
import logging
import math
import sys
from time import sleep
import os

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), "..", "..")))
from transforms3d import euler
```

(continues on next page)

(continued from previous page)

```

from interface.adapter import TcpServerAdapter, TcpClientAdapter
from util.transforms import object2tcp
    
```

### Define class

Define the sub-class "TestAdapter" that inherits the parent class "TcpServerAdapter".

```

class TestAdapter(TcpServerAdapter):
    vision_project_name = "Large_Non_Planar_Workpieces"
    # vision_project_name = 'Vis-2StationR7-WorkobjectRecognition-L1'
    is_force_real_run = True
    service_name = "test Adapter"

    def __init__(self, address):
        super().__init__(address)
        self.robot_service = None
        self.set_recv_size(1024)
    
```

**Note:** This example defined the Adapter program as the TCP/IP Socket server.

### Set how to receive command and the packet processing logic

Set the processing logic of the received request, including the image capturing command and parameter recipe ID.

```

# Receive command _create_received_section
def handle_command(self, cmds):
    photo_cmd, *extra_cmds = cmds.decode().split(',')
    recipe = extra_cmds[0]
    # Check command validity _check_cmd_validity_section
    if photo_cmd != 'P':
        self.msg_signal.emit(logging.ERROR, 'Illegal command: {}'.format(photo_cmd))
        self.send(('1' + ' ' + ' ').encode())
        return
    # Check whether vision is registered _check_vision_service_section
    if not self.is_vision_started():
        self.msg_signal.emit(logging.ERROR, 'Vision project not registered: {}'.
    ←format(self.vision_project_name))
        self.send(('2' + ' ' + ' ').encode())
        return
    # Change TODO parameter "extra_cmds" according to actual conditions
    sleep(0.1) # wait for a cycle of getting in Vision
    # _check_vision_result_function_section

    try:
        result = self.select_parameter_group(self.vision_project_name, int(recipe) -
    ←1)
        if result:
            result = result.decode()
            if result.startswith("CV-E0401"):
                self.send(('5' + ' ' + ' ').encode())
                return
            elif result.startswith("CV-E0403"):
    
```

(continues on next page)



(continued from previous page)

```

        self.send(('5' + ' ' + ' ').encode())
        return
        raise RuntimeError(result)
except Exception as e:
    logging.exception('Exception happened when switching model: {}'.format(e))
    self.send(('5' + ' ' + ' ').encode())
    return
self.show_custom_message(logging.INFO, "Switched model for project successfully")

self.msg_signal.emit(logging.WARNING, 'Started capturing image')
try:
    self.check_vision_result(json.loads(self.find_vision_pose().decode()))

except Exception as e:
    self.msg_signal.emit(logging.ERROR, 'Calling project timed out. Please check
↳whether the project is correct: {}'.format(e))
    self.send(('2' + ' ' + ' ').encode())

```

**Note:** "handle\_command" function the start point for the TCP/IP Socket server to process received packets.

### Define the check of Mech-Vision vision results

Set the Adapter to check Mech-Vision vision results.

```

# Check vision results
def check_vision_result(self, vision_result, at=None):

    noCloudInRoi = vision_result.get('noCloudInRoi', True)
    if noCloudInRoi:
        self.msg_signal.emit(logging.ERROR, 'No point clouds')
        self.send(('4' + ' ' + ' ').encode())
        return

    poses = vision_result.get('poses')
    labels = vision_result.get('labels')
    if not poses or not poses[0]:
        self.msg_signal.emit(logging.ERROR, 'No vision points')
        self.send(('3' + ' ' + ' ').encode())
        return

    self.send(self.pack_pose(poses, labels).encode())
    self.msg_signal.emit(logging.INFO, 'Sent TCP successfully')

```

### Set the output format of vision points

Set the output format of the vision points.

```

# Pack pose _pack_pose_section
def pack_pose(self, poses, labels, at=None):

    pack_count = min(len(poses), 1)
    msg_body = ''

```

(continues on next page)

(continued from previous page)

```

for i in range(pack_count):
    pose = poses[i]
    object2tcp(pose)
    t = [p * 1000 for p in pose[:3]]
    r = [math.degrees(p) for p in euler.quat2euler(pose[3:], 'rzyx')]
    p = t + r
    self.msg_signal.emit(logging.INFO, 'Sent pose: {}'.format(p))
    msg_body += ('{: .4f}', * (len(p) - 1) + '{: .4f}').format(*p)

    if i != (pack_count - 1):
        msg_body += ','
return '{}'.format(0) + msg_body + '

```

### Define the close operation of Adapter

Define how to close Adapter.

```

def close(self):
    super().close()

```