

Flexxbotics Developer Guide

Notice of Rights

Copyright © 2026 by Flexxbotics, Inc. and/or its affiliates. This work is licensed under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/) except where otherwise noted. This document is protected by U.S. and international copyright laws and conventions.

Flexxbotics and the Flexxbotics "X" logo are registered trademarks of Flexxbotics, Inc. in the United States and other countries. All other trademarks referenced herein are the property of their respective owners.



Contents

- Devices, Transformers, APIs, and Runtime Concepts.....2**
- 1. Core Architecture Overview & Studio..... 2**
- 2. Devices..... 2**
 - What is a Device?..... 2
 - Device Creation & Assignment..... 3
- 3. Machine Models..... 3**
 - Machine Model Definitions..... 3
- 4. Transformers..... 4**
 - What is a Transformer?..... 4
 - Transformer JSON..... 4
 - Runtime Behavior..... 4
- 5. Extensions..... 4**
 - What are Extensions?..... 4
- 6. Device Assignment to Layouts..... 5**
 - Machine Layouts..... 5
 - Purpose..... 5
- 7. Key Concept: Primary Device & Polling..... 6**
 - Primary Device Concept..... 6
 - Workcell Transformer..... 6
 - Status Events..... 6
- 8. Key Concept: Monitoring & Poll Interval..... 6**
 - Poll Interval Configuration..... 6
 - Polling Methods..... 7
 - Use Cases..... 7
 - Best Practice..... 7
- 9. Protocols..... 7**
 - What are Protocols?..... 7
 - Common Interface..... 8

- 10. Key Concept: Device API & Transformer Method Mapping..... 8**
 - Overview..... 8
 - Key Endpoints..... 8
 - Clarifier: Execute Command..... 9
 - Listing Devices To Retrieve device_id..... 9
- 11. External Communication to Flexxbotics: REST & Adapters..... 10**
 - Direct REST Access..... 10
 - Adapters..... 10
 - TCP Adapter..... 10
 - TCP Adapter Message Format..... 10
 - Other Types of Adapters that can be created:..... 10
 - Existing Extensions to Communicate via TCP to Flexxbotics..... 10
- 12. Scripts..... 11**
 - What are Scripts?..... 11
 - Script Launching..... 11
 - Core Script Components..... 11
 - Script Execution Model..... 12
- 13. Recommended Design Patterns..... 12**

Devices, Transformers, APIs, and Runtime Concepts

1. Core Architecture Overview & Studio

Flexxbotics is built around **Devices** and **Transformers**, with supporting concepts such as **extensions**, **protocols**, **scripts**, and **adapters**. Together, these components enable communication with equipment, runtime execution, visualization, and autonomous control.

These entities are completely editable via the **Studio** development environment. The studio environment can be accessed in Flexx Edge by navigating to the top right menu > Devices & Transformers > Studio tab.

2. Devices

What is a Device?

A **Device** is the top-level runtime object created from the **Devices** screen in the Flexxbotics UI. Devices represent physical or logical equipment (PLCs, machines, robots, safety systems, cameras, inspection systems, workcells, etc.) within the platform.

Device Creation & Assignment

- Devices are created based on **machine model options**.
- Machine models determine:
 - Equipment OEM
 - Equipment model
 - Controller type
 - Which **Transformer** is used for communication

3. Machine Models

Machine Model Definitions

Machine models are defined in JSON files located in the `models` folder of a transformer.

Each machine model JSON specifies:

- OEM
- Equipment model
- Controller
- The transformer responsible for communicating with the equipment

Machine models act as the configuration layer that binds physical equipment to a specific transformer implementation.

4. Transformers

What is a Transformer?

A **Transformer** is the software component responsible for communicating with equipment.

Transformer JSON

Each transformer has a JSON definition that includes:

- Metadata for the transformer
- The Python file name
- The Python class name

Runtime Behavior

- When a device is created, an instance of its transformer is:
 - Assigned to the device
 - Instantiated in the runtime environment
 - The transformer methods become the execution points for:
 - Status reads
 - Variable reads/writes
 - Commands
 - Polling logic
 - And more...
-

5. Extensions

What are Extensions?

Extensions are external software components that run *outside* the Flexxbotics runtime.

Common characteristics:

- Installed directly on:
 - Equipment controllers
 - PCs controlling the equipment
- Often implemented as servers
- Translate native equipment data into the Flexxbotics API when needed

Extensions are commonly used when:

- Native equipment protocol interfaces are not readily exposed external to the device and instead live in API's and software written with SDK's on the 3rd party device through the OEM's 3rd party developer tools.
 - Highly specific functionality for a particular device is needed and not readily exposed via an interface but can be adapted (file manipulation, TCP clients, etc)
-

6. Device Assignment to Layouts

Machine Layouts

- Machine layouts are defined in **Flexx Control**
- Equipment placed in a layout can be assigned a device

Purpose

- Enables visualization of device status

- Links runtime devices to physical cell layouts
 - Used primarily for monitoring and UI representation
-

7. Key Concept: Primary Device & Polling

Primary Device Concept

Only **one device** should be marked as the **Primary Device**.

The primary device:

- Represents the main controller of a workcell
- Drives **status events**
- Determines overall workcell status

Workcell Transformer

- A **Workcell Transformer** is a generic transformer used to coordinate multiple devices
- It can communicate with all other transformers that are defined and instantiated in the runtime.

Example:

A workcell transformer may query both a robot and a safety PLC, combine their states, and determine the overall cell status.

Status Events

- Generated only by the **primary device**
 - Appear in the **Events** tab on the Analytics page
 - Used for workcell-level monitoring and analytics
-

8. Key Concept: Monitoring & Poll Interval

Poll Interval Configuration

Monitoring allows a device to be polled at a fixed interval.

Typical usage:

- Only **one device** is polled
- Usually the **primary device**

Polling Methods

On each poll interval, the following transformer methods are called:

- `_read_interval_data`
- `_read_status`

Use Cases

- Automated data collection
- Closed-loop control
- Status reconciliation
- Autonomous logic execution

Best Practice

Use a **workcell transformer** for polling logic so:

- All cell-level logic resides in one place
- Other transformers remain simple connectors

9. Protocols

What are Protocols?

Protocols are reusable Python components used inside transformers to communicate with equipment.

Common Interface

Most protocols implement four core methods:

- `connect`
- `disconnect`
- `send`
- `receive` (automatically invoked by `send`)

Protocols abstract low-level communication details and enable consistent transformer implementations.

10. Key Concept: Device API & Transformer Method Mapping

Overview

The Device API allows developers to directly interact with devices and test transformer logic.

Base URL:

<http://flexxcore.app.local:7081/api/v2e/devices/>

The API can be viewed in a browser at: <http://flexxcore.app.local:7081/api/#/>

Each endpoint maps directly to a transformer method.

Key Endpoints

API Endpoint	Transformer Method
GET /api/v2e/devices/{device_id}/status	_read_status
GET /api/v2e/devices/{device_id}/variables/{variable_name}	_read_variable
PATCH /api/v2e/devices/{device_id}/variables/{variable_name}	_write_variable
GET /api/v2e/devices/{device_id}/parameters/{parameter_name}	_read_parameter
PATCH /api/v2e/devices/{device_id}/parameters/{parameter_name}	_write_parameter
POST /api/v2e/devices/{device_id}/execute_command/{command_name}	execute_command_v2

Clarifier: Execute Command

- Acts as a generic “catch-all” command endpoint
- Logic is implemented inside `execute_command_v2`
- Uses:
 - `command_name` to select behavior
 - `args` (JSON payload) to pass parameters

Listing Devices To Retrieve device_id

To retrieve all devices and their metadata:

GET /api/v2e/devices

11. External Communication to Flexxbotics: REST & Adapters

Direct REST Access

Any system with REST capabilities can communicate directly with the Device API.

Adapters

For devices without REST support, **Adapters** are used.

TCP Adapter

- Translates TCP socket communication into REST
- Runs in a Docker container on port **7082**
- Always available by default

TCP Adapter Message Format

```
{  
  "type": "GET | PATCH | POST",  
  "endpoint": "/api/v2e/devices/{device_id}/status",  
  "body": {}  
}
```

Other Types of Adapters that can be created:

- OPC-UA
- Ethernet
- Always available by default

Existing Extensions to Communicate via TCP to Flexxbotics

Pre-written integrations are available for:

- Universal Robots (URCap)
- FANUC (KAREL)

These reside in their respective transformer directories and use the adapter pattern.

12. Scripts

What are Scripts?

Scripts are Python files executed inside the Flexxbotics runtime to perform higher-level tasks.

Common use cases:

- Custom operator workflows
- Autonomous control loops
- Closed-loop device coordination
- Data-driven automation

Script Launching

- Scripts are launched via HMI controls
- Configured through the **Advanced** tab

Core Script Components

Scripts typically include:

1. **FlexxCoreClient**
 - API client for interacting with Flexxbotics
2. **FlexxGUI**

- UI framework for building custom operator interfaces

3. Main Script Class

- Main process logic and control flow

Script Execution Model

- Scripts run in a **Script Executor Queue**
- Only **one script can run at a time**
- Continuous applications must:
 - Encapsulate all logic in a single script
 - Manage loops and UI prompts internally

Example:

The APC demo runs a continuous background loop and occasionally prompts operators through the Flexx GUI.

13. Recommended Design Patterns

- Use **workcell transformers** for cell-level logic monitoring
- Keep individual device transformers as thin connectors
- Poll only the primary device
- Centralize autonomous logic in:
 - Workcell transformers
 - Scripts (when UI or orchestration is required)