EPSON

Epson RC+ 8.0扩展功能
RC+ Extensions 8.0

翻译版

Rev.1
SCM263S8583F

目录

1. 简介

1.1 简介

感谢您购买本公司的机器人系统。

本手册记载了正确使用 RC+ Extensions 的所需事项。

使用该机器人系统前，请仔细阅读本手册与其他相关手册。

阅读之后请妥善保管，以便随时取阅，如有不明之处，请再次阅读。

本公司的产品均通过严格的测试和检查，以确保机器人系统的性能符合本公司的标准。但是如果在超出本手册所描述的环境中使用本产品，则可能会影响产品的基本性能。

本手册阐述了本公司可以预见的危险和问题。请务必遵守本手册中的安全注意事项，安全正确地使用机器人系统。

1.2 商标

Microsoft、Windows、Windows标识、Visual Basic、及Visual C++为美国Microsoft Corporation在美国或其它国家的注册商标或商标。

其它品牌与产品名称均为各公司的注册商标或商标。

1.3 关于标记

Microsoft® Windows® 10 Operating system
Microsoft® Windows® 11 Operating system
本使用说明书将上述操作系统分别标记为Windows 10、Windows 11。另外，有时可能将Windows 10、Windows 11统一标记为Windows。

1.4 注意

禁止擅自复印或转载本手册的部分或全部内容。
本手册记载的内容将来可能会随时变更，恕不事先通告。
如您发现本手册的内容有误或需要改进之处，请不吝斧正。

1.5 制造商

SEIKO EPSON CORPORATION

1.6 联系方式

联系方式的详细内容登载于以下手册中的“销售商”处。
各地区的咨询处有所不同，敬请注意。
"安全手册”- 联系方式"

从以下网站也可浏览安全手册。
URL： https://download.epson.biz/robots/

1.7 操作入门

本节介绍了您在阅读本手册之前应了解的事项。

关于Epson RC+ 8.0的安装文件夹

Epson RC+ 8.0可安装在任意指定的路径。本手册中是默认Epson RC+ 8.0被安装C:\EpsonRC80中进行说明。

符号含义

使用下述标记来记载安全注意事项。请务必阅读。

⚠ 警告

如果用户忽视该指示或处理不当，可能会导致死亡或重伤。

⚠ 警告

如果用户忽略该指示或处理不当，可能会因触电而受伤。

⚠ 注意

如果用户忽略该指示或处理不当，可能会导致人身伤害或财产损失。

2. 概述

2.1 概述

RC+ Extensions在视频中也有说明。

Title	Link
1. New Feature: RC+ Extensions	

要点

- 若要浏览视频，需要连接互联网。
- 会播放声音。
- 如果要以母语显示，请使用YouTube的自动翻译字幕功能。

RC+ Extensions 是一个能够根据用户的需求和业务流程，对 Epson RC+ 8.0 进行自定义，并通过与外部设备及系统联动，实现灵活扩展的平台。

除了 Epson 提供的扩展功能 (Extension) 之外，用户还可以使用 Visual Studio 开发专有扩展功能 (Extension)，并将其安装到 Epson RC+ 8.0 中进行使用。
通过上述方式，Epson RC+ 8.0 能够持续且灵活地进行功能演进，所开发的扩展功能 (Extension) 也便于在组织内部进行复用和共享。

RC+ Extensions 的功能：

- 扩展功能的使用：可安装并使用 Epson 官方发布的扩展功能 (Extension)，或用户自行开发的扩展功能 (Extension)。
- 扩展功能的开发：可利用 Visual Studio 和 RC+ Extensions SDK 创建专有扩展功能 (Extension)。

要点

需要 Epson RC+ 8.0 Ver8.1.3.0 及以上版本，以及 Epson RC+ Premium Edition 许可证。

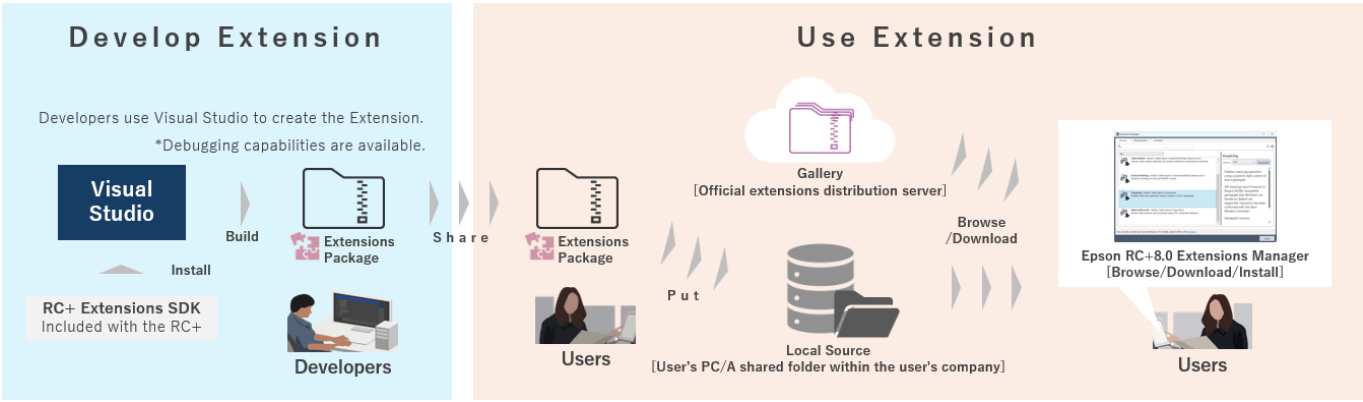
术语说明

术语	描述
RC+ Extensions	用于扩展 Epson RC+ 8.0 的平台。除了 Epson 提供的扩展功能，用户也可以自行开发扩展功能，并将其添加到 Epson RC+ 中实现扩展。通过使用 RC+ Extensions，可以根据需求灵活自定义 Epson RC+ 的 UI 扩展、外部设备联动、嵌入专有处理等功能。
Extension	指通过 RC+ Extensions 开发并可添加到 Epson RC+ 的扩展功能。使用 Visual Studio 和 RC+ Extensions SDK 进行开发，并以 .rcxpkg 格式的包输出用于分发。Extension 安装到 RC+ 后，可以实现 UI、菜单的新增、SPEL+ 联动、外部设备控制等功能。

术语	描述
扩展功能管理器	Epson RC+ 内置的 Extension 管理界面。可对 Extension 执行以下操作： *另见 *下载 / 删除 *安装 / 卸载 *启用 / 禁用 *本地源文件夹设置 可通过 Epson RC+ 主菜单的 [扩展]→[扩展功能管理器] 启动。
Gallery	Epson 官方提供的 Extension 在线分发服务器。在扩展功能管理器的 [浏览] 标签页会自动显示列表，用户可选择所需的 Extension 进行下载。需要互联网连接。用户开发的 Extension 无法发布到Gallery (用户分发需使用本地源)。
本地源	指用于存放用户或组织管理的 Extension 包 (.rcxpkg) 的电脑本地文件夹或网络共享文件夹。通过扩展功能管理器的“本地源文件夹设置”注册文件夹后，该文件夹内的 Extension 会在 [浏览] 标签页显示列表，用户可像Gallery一样进行下载和安装。

下图展示了 RC+ Extensions 的使用方法及开发流程。

- 用户：可通过 Epson RC+ 的扩展功能管理器，浏览、下载并安装 Epson 官方发布的Gallery中的扩展功能 (Extension)。指定本地源后，也可以同样使用开发者创建的扩展功能 (Extension)。
- 开发者：可利用 Visual Studio 和 RC+ Extensions SDK 创建扩展功能 (Extension)，并进行打包和共享。



在 RC+ Extensions 中，可以使用和开发以下类型的扩展功能。各自的目标功能和开发方式有所不同。

1. **RC+ 自定义**：自定义 Epson RC+ 的 UI 和处理流程
   - 可为 Epson RC+ 添加专有操作界面。此外，还可通过 SDK 提供的标准功能与外部软件、硬件联动，实现新的功能。
2. **PC 视觉 自定义视觉对象**：自定义视觉对象
   - 可为 VisionGuide 的 PC 视觉功能添加专有视觉对象。由此可以嵌入专有的图像处理或检测逻辑。

有多种方式可以利用 Epson RC+。可根据目的选择不同的功能。

功能	主要用途
RC+ Extensions	用于添加 Epson RC+ 的界面和功能，自定义UI和工作流程。此外，也可将开发的扩展功能打包，在组织内部进行复用和共享。
RC+ Library Builder	用于将 SPEL+ 程序库化并实现复用。
RC+ API	用于通过外部应用程序利用 Epson RC+ 控制机器人 (外部操作 RC+)。

3. Extension的使用

3.1 概述

要在 Epson RC+ 中使用 Extension，需要通过扩展功能管理器执行以下操作。

- Extension 的下载
- Extension 的安装
- Extension 的启用

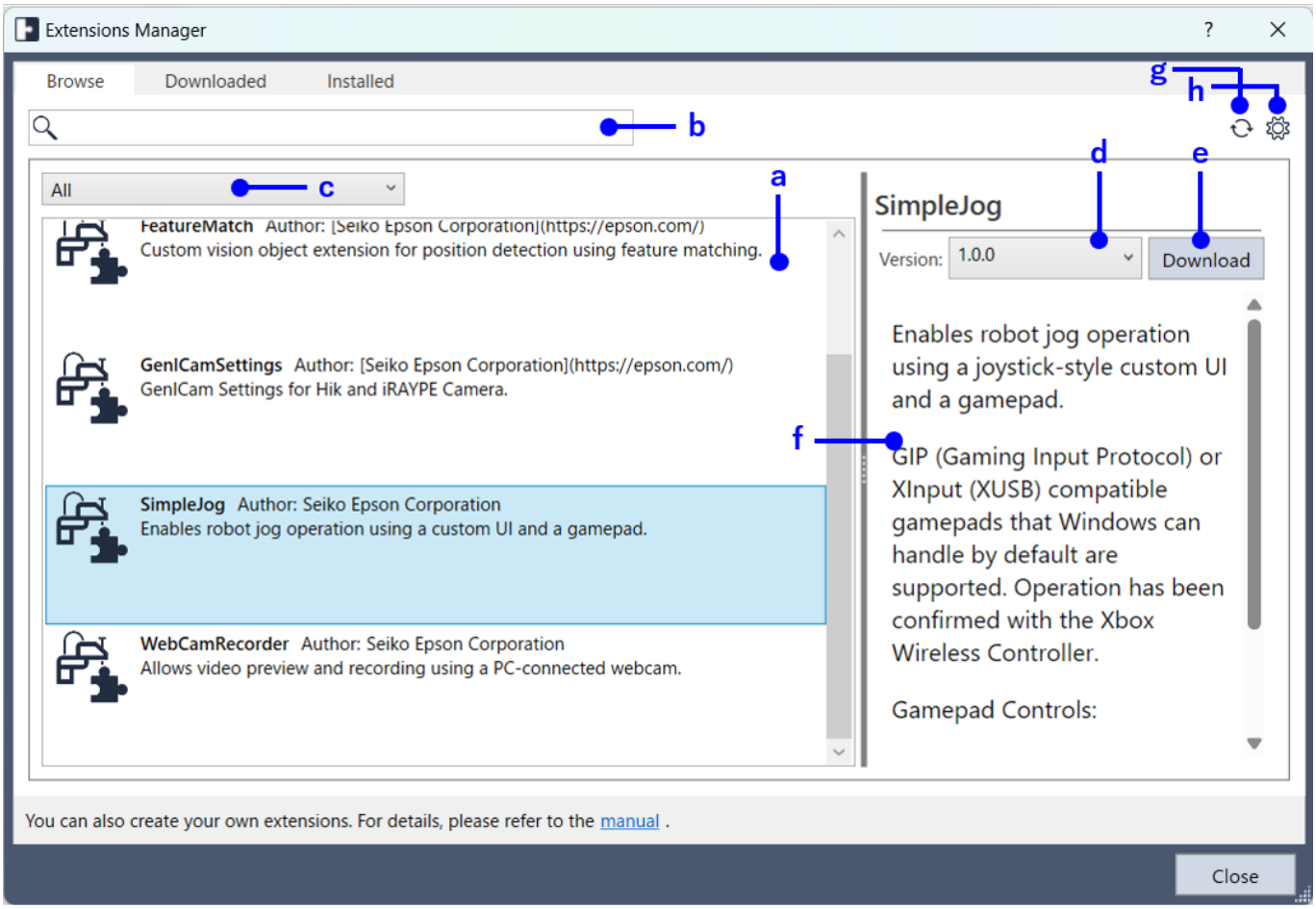
要点

安装 Extension 后，默认处于启用状态。

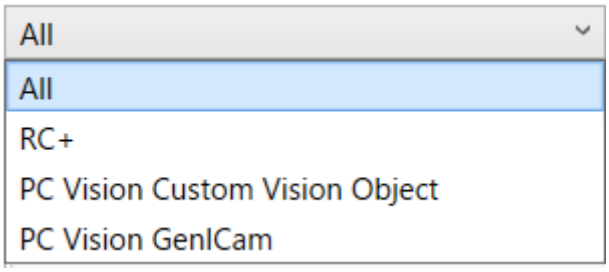
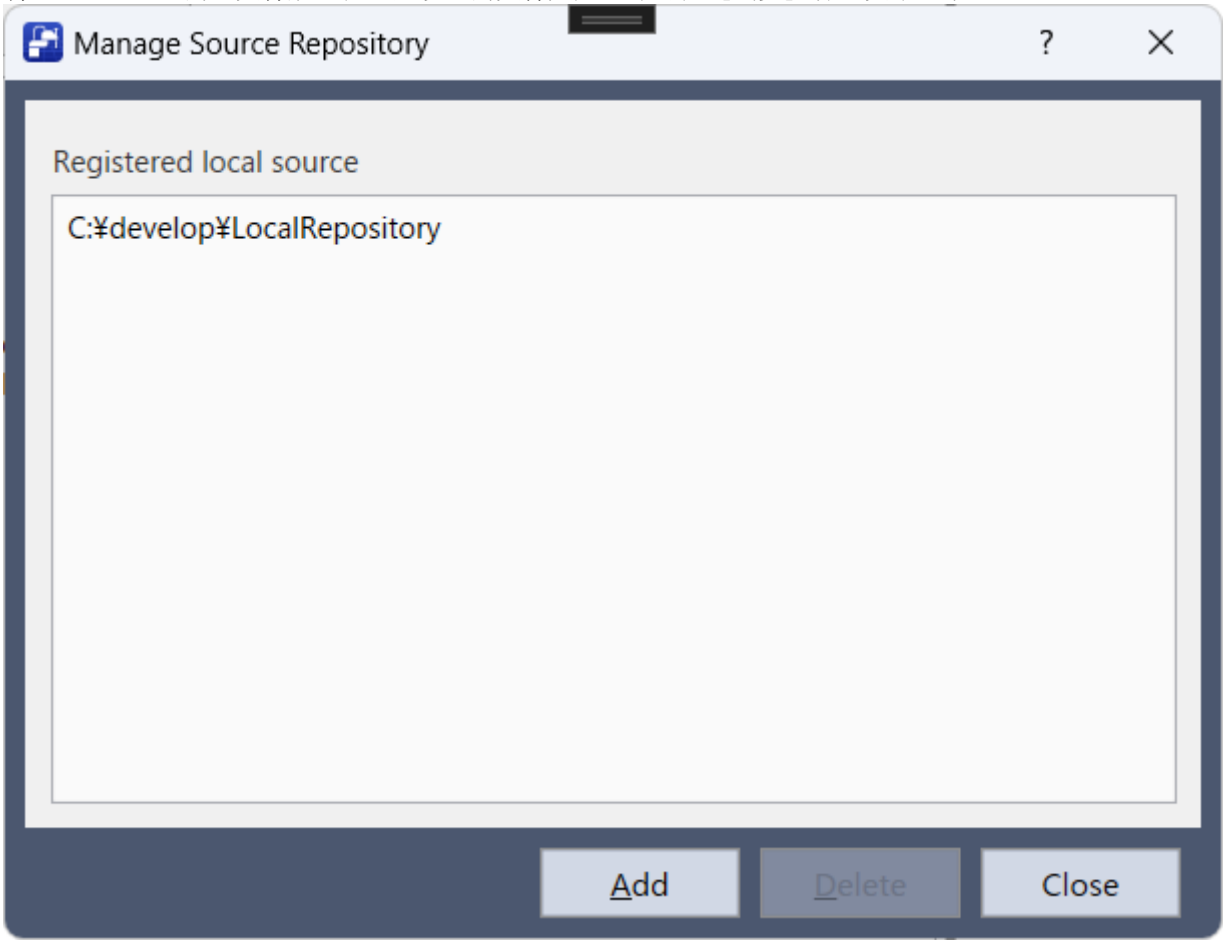
扩展功能管理器可通过在［扩展］菜单中选择［扩展功能管理器］启动。

3.2 Extension 的下载

Extension 的下载操作需通过扩展功能管理器的[参照]标签页进行。
请选择要下载的 Extension 及其版本，然后按下[下载]按钮。

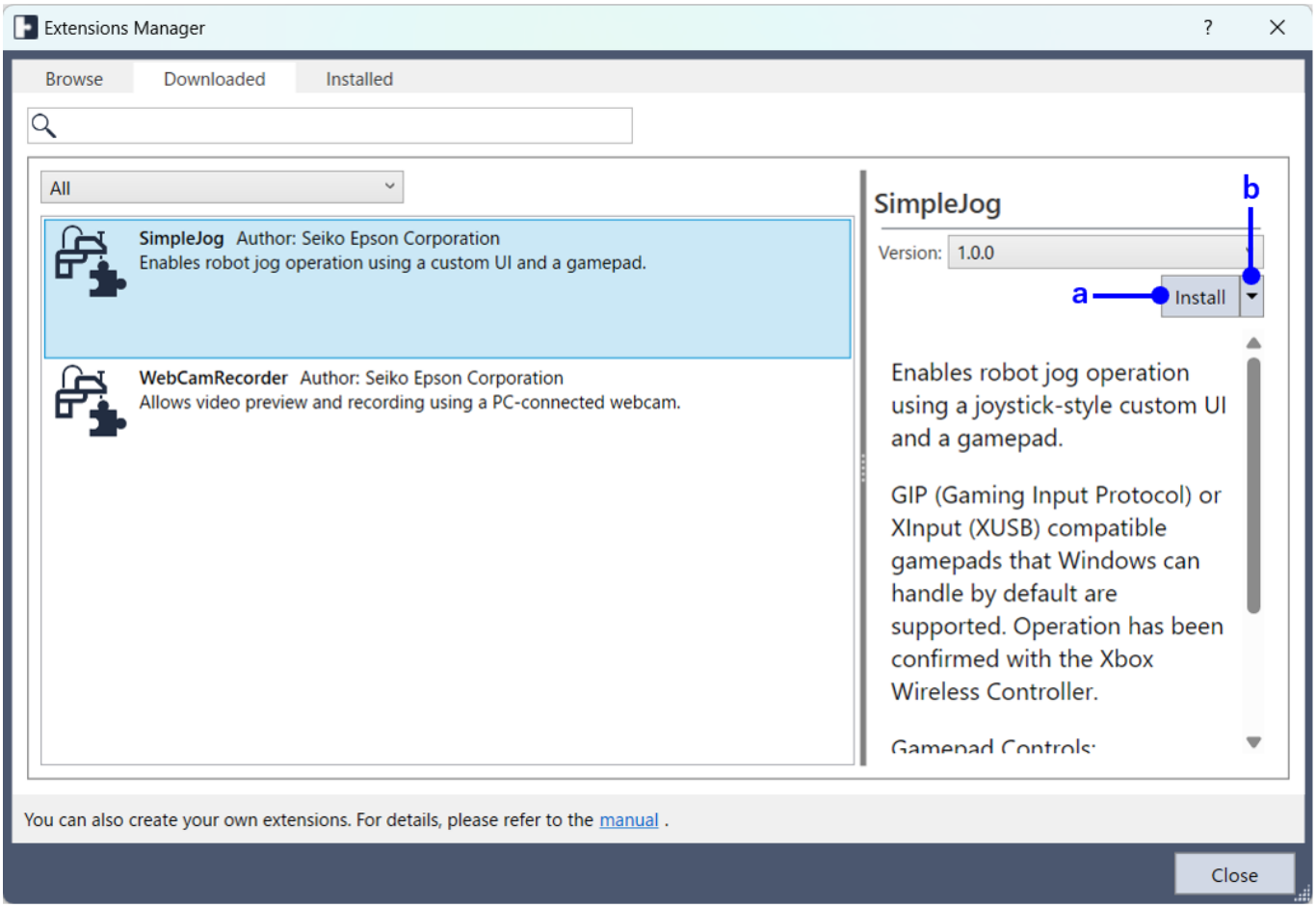


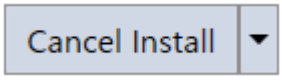
符号	描述
a	将显示以下 Extension。 - 由 Epson 公开发布的 Extension - 部署于本地源的 Extension

符号	描述
b	包含所输入字符的 Extension 会显示在“a”中。 若未输入任何内容，将显示所有 Extension。
c	请选择在“a”中显示的 Extension 类型。 
d	请选择 Extension 的版本。
e	将下载在“a”中选定的 Extension 及在“d”中选定的版本。 下载完成的 Extension 会显示在［已下载］标签页中。
f	将显示所选 Extension 的说明。
g	更新“a”的显示内容。
h	设置本地源的文件夹。 将 Extension 的包文件放置在此处设置的文件夹后，即可在［浏览］标签页中显示。 

3.3 Extension 的安装

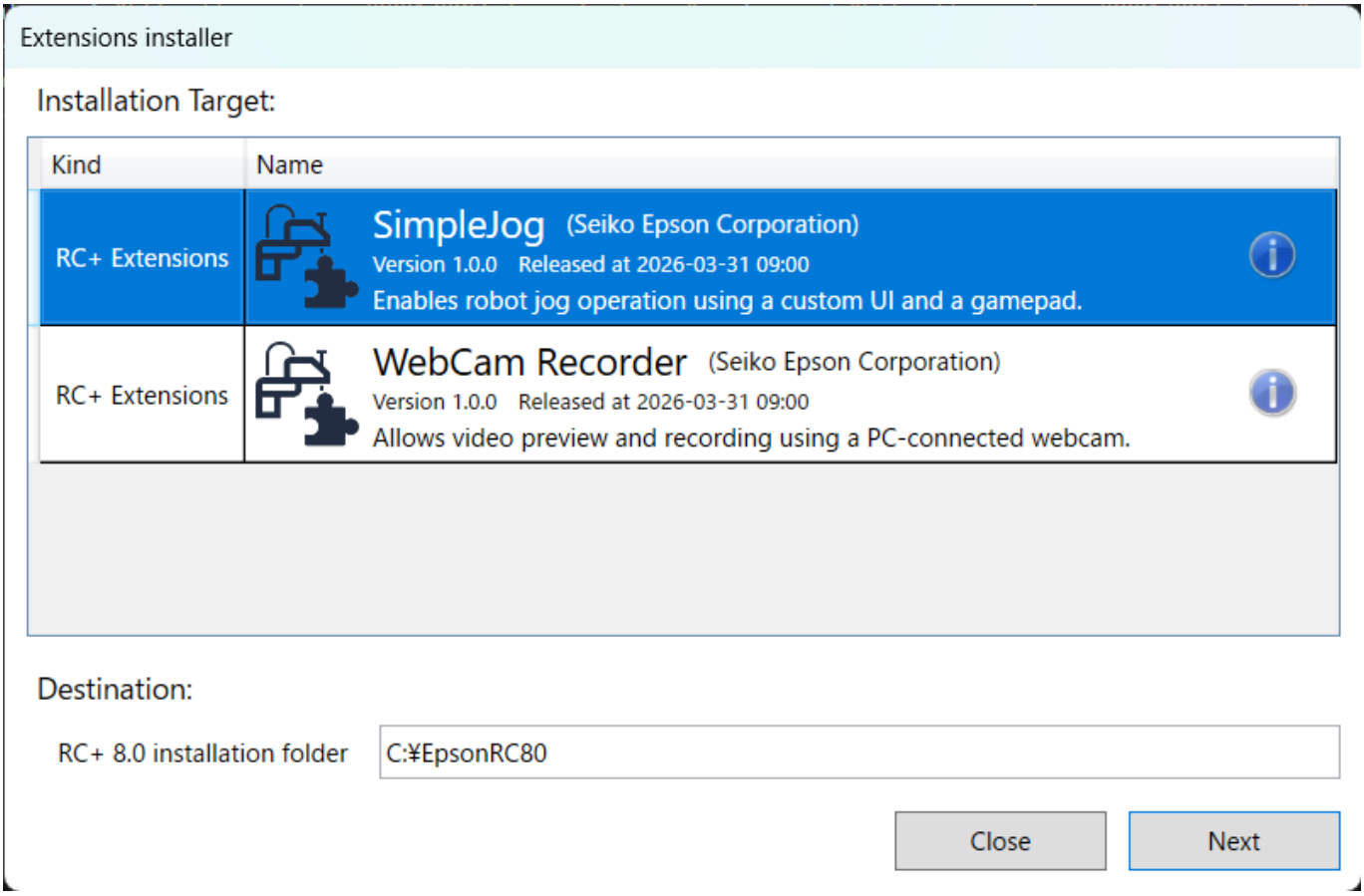
Extension 的安装通过扩展功能管理器的［已下载］标签页进行。
请选择要安装的 Extension 及其版本，然后按下［安装］按钮。
在 Epson RC+ 关闭时，将启动［扩展功能安装程序］，并执行 Extension 的安装。



符号	描述
a	安装 Extension。 按钮被按下一次后，按钮显示将变为［取消安装］。再次按下按钮即可取消安装。 
b	将显示以下菜单。  - 安装：与 a 相同。 - 删除：可以删除已下载的 Extension。

扩展功能安装程序

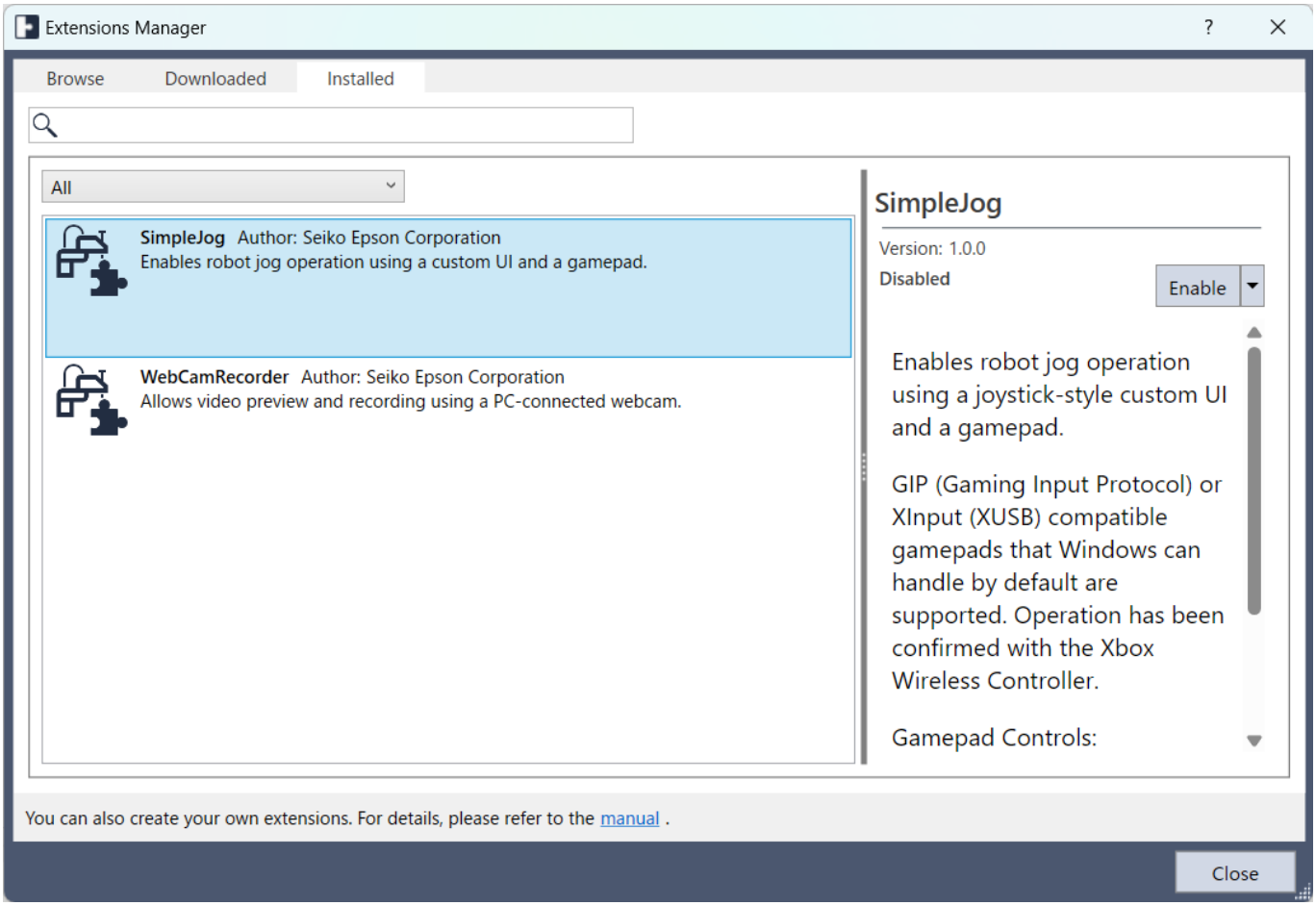
在 Epson RC+ 关闭时，扩展功能安装程序将启动。按下［下一步］按钮，进行安装。



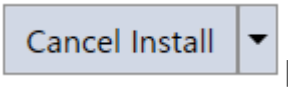
Extension 的启用与禁用

Extension 的启用通过扩展功能管理器的［已安装］标签页进行。

请选择要启用的 Extension，然后按下［启用］按钮。
在 Epson RC+ 关闭时，Extension 将被启用，并可在下次启动时使用。



按下一次［启用］按钮后，按钮显示将变为［取消启用］。再次按下按钮即可取消启用。

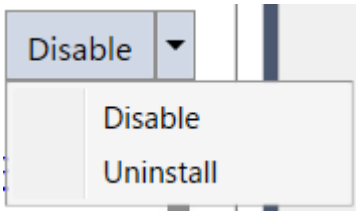


Extension 的禁用也采用相同的步骤。
如需临时禁用所有 Extension，请指定以下命令行选项启动 Epson RC+。

```
erc80.EXE /DISABLE_EXT
```

3.4 Extension 的卸载

Extension 的卸载操作需在扩展功能管理器的［已安装］标签页中进行。
请选择需要卸载的 Extension，按下［启用］/［禁用］按钮旁的［▼］，然后点击卸载菜单。
在 Epson RC+ 关闭时，Extension 将会被卸载。



4. Extension的种类

4.1 概述

在 RC+ Extensions 中，可以开发以下扩展功能。各自的目标功能和开发方式有所不同。

Extension的种类	主要使用的语言 / 技术	概述
RC+ 自定义: 可对 Epson RC+ 的用户界面以及处理流程进行自定义。	C#, WPF, XAML	可为 Epson RC+ 添加专有操作界面。此外，通过将 SDK 提供的标准功能与外部软件或硬件进行集成，可以实现新的功能。
PC 视觉 自定义视觉对象: 可自定义视觉对象。	C++, (图像处理知识)	可为 VisionGuide 的 PC 视觉功能添加专有视觉对象。由此可以嵌入专有的图像处理或检测逻辑。

要点

需要 Epson RC+ 8.0 Ver8.1.3.0 及以上版本，以及 Epson RC+ Premium Edition 许可证。

开发流程如下:

1. 开发环境准备: 参阅 [安装]
2. 创建 RC+ Extensions 项目: 参阅[开始使用]
3. 实现与调试: 参阅 [教程]
4. 打包与共享: 参阅 [Extension 的分发]

从下一节开始，将介绍各类型的开发方式。

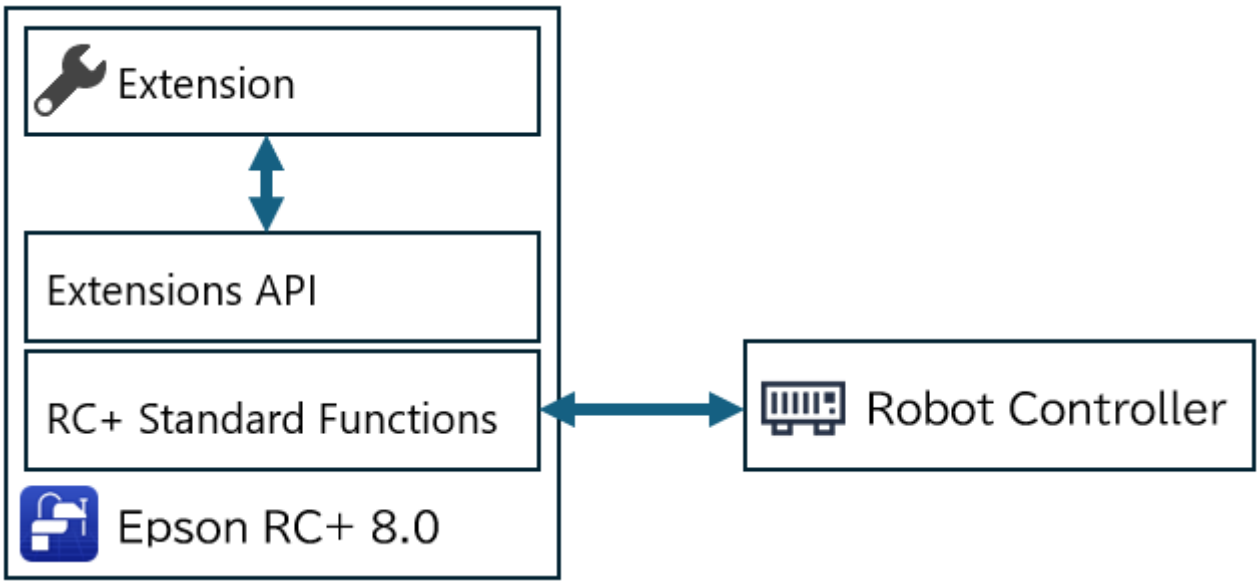
5. RC+自定义的开发

5.1 概述

RC+ 自定义能够扩展 Epson RC+ 的标准功能，以满足用户特定需求。可以实现以下扩展功能。

- **UI 扩展**: 可以在 Epson RC+ 界面上添加专有窗口。
- **标准功能调用**: 可以调用Epson RC+的标准功能，如点编辑、控制器连接等。
- **SPEL+ 联动**: 通过 Declare 语句，可以从 SPEL+ 程序调用 Extension 功能。
- **Web 内容集成**: 利用 C# WPF 的 WebView2，在 Epson RC+ 上显示网页。此外，Extension 与 Web 之间可以实现双向消息通信。

系统结构如下所示。



Extension 在 RC+ 本体进程内 (进程内) 运行。Extension 通过 RC+ Extensions SDK 的 Extensions API 公开接口访问 RC+ 标准功能，因此无法直接访问 RC+ 内部。通过该设计，既能保证 RC+ 标准功能的一致性，又能安全地实现扩展的 UI 添加和功能集成。

5.2 安装

开始开发前，需要进行以下准备。

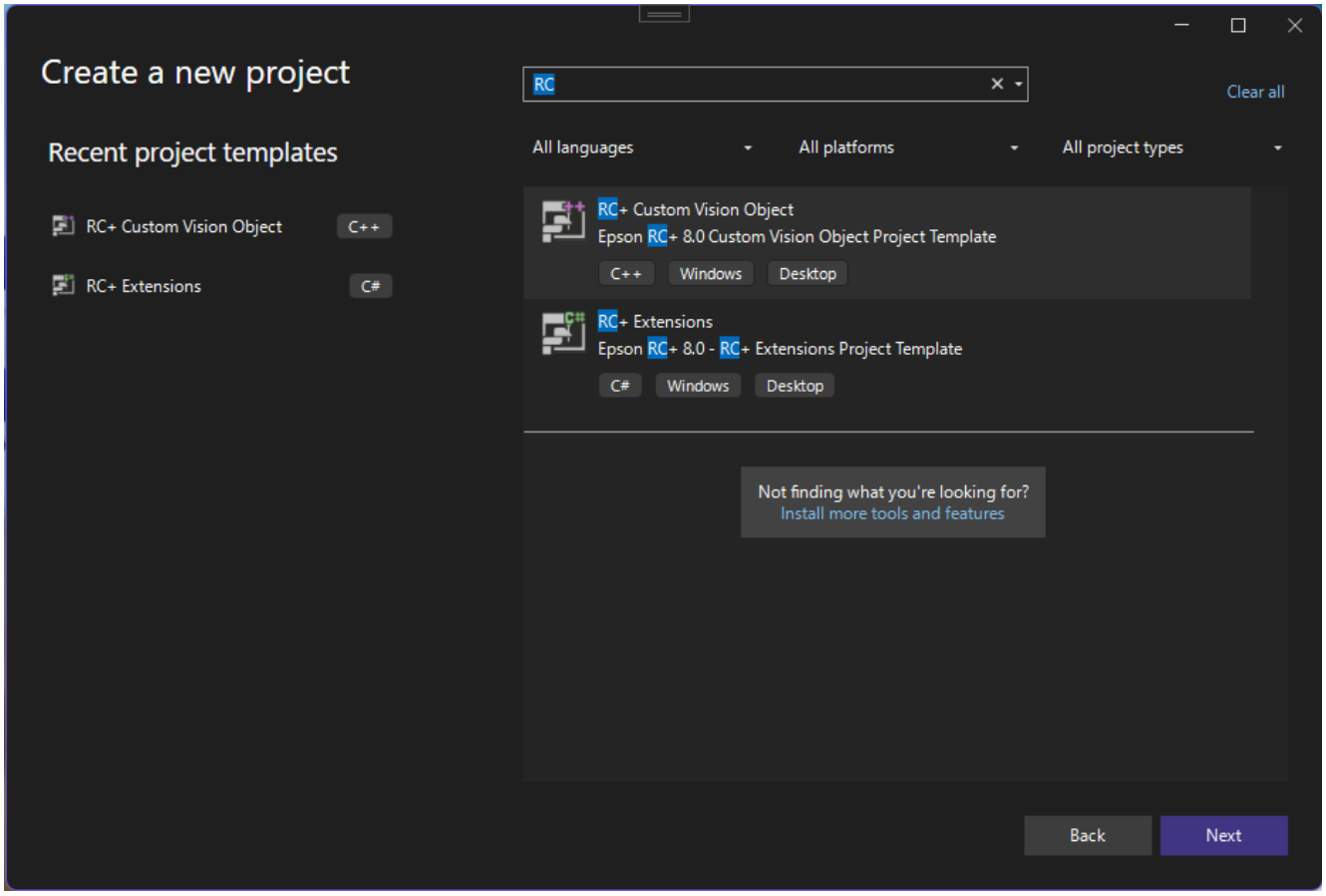
1. 安装Epson RC+ 8.0。
   - 版本：8.1.3.0或以上
   - 需要的许可证：Epson RC+ Premium Edition。
   - 安装路径可任意指定。本手册假设使用C:\EpsonRC80。
2. 安装Visual Studio。
   - 版本：Visual Studio 2022 17.14或以上（包括Enterprise、Professional或Community版本）
   - 必需组件：.NET桌面开发、使用C++的桌面开发
3. 安装.NET 8 SDK。
   - 版本：8.0.307或以上。
4. 向Visual Studio安装扩展功能(VSIX)。
   - 双击C:\EpsonRC80\Extensions\SDKs\ExtensionsGenerator.vsix。VSIX安装程序将会启动，按照画面上的指示进行安装。
     *使用VSIX需要Premium Edition。启动使用Premium Edition认证的RC+，VSIX将自动展开到上述路径。

- 此扩展功能会在Visual Studio中注册用于RC+ Extensions的“项目模板”和“文件模板”。后续步骤中，将使用这些模板来创建项目。

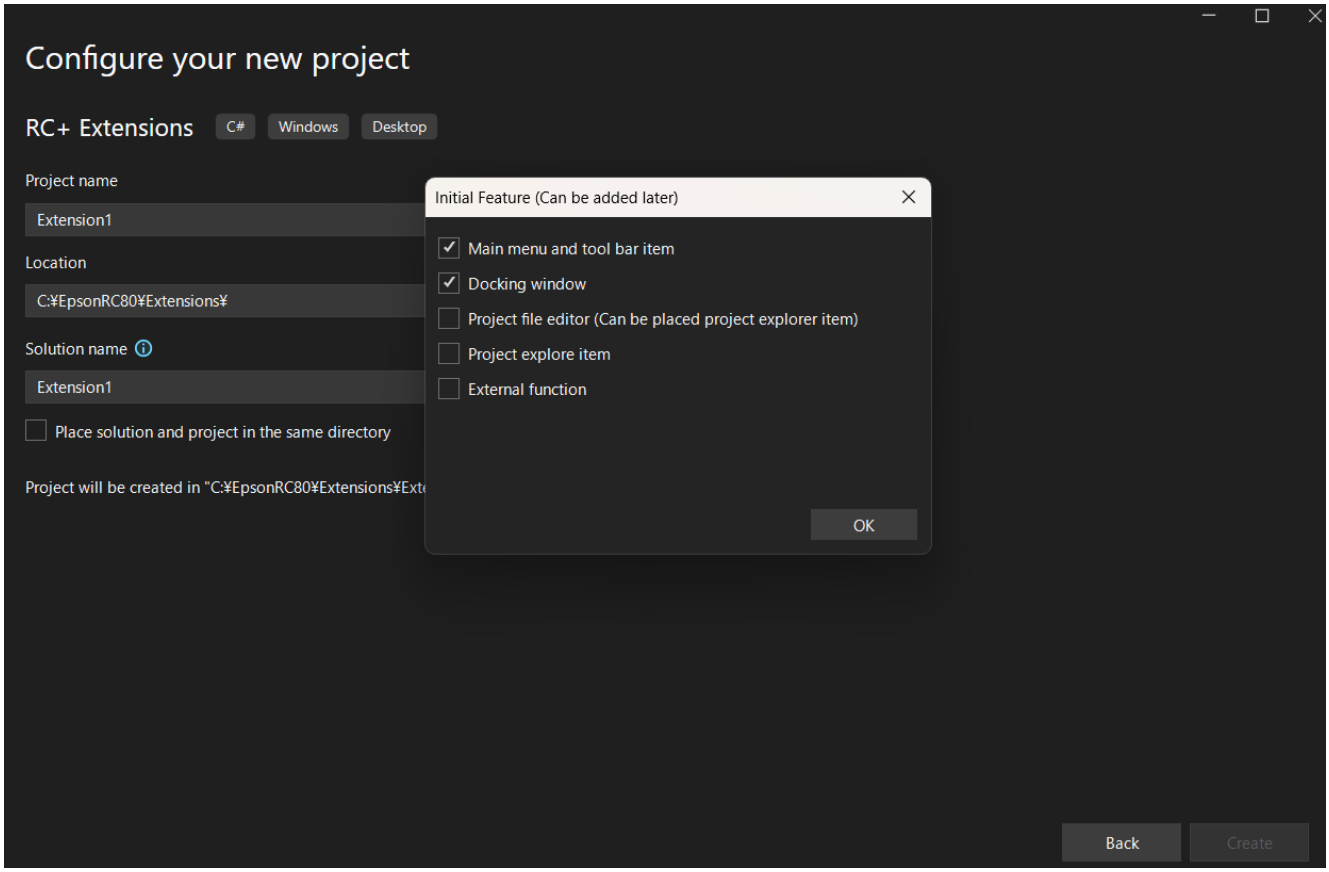
5.3 开始使用

按照以下步骤创建 RC+ Extensions 项目。

1. 在 Visual Studio 中选择［新建项目］。
2. 选择［RC+ Extensions］项目模板。



3. 在 [Initial Feature] 窗口中，选择要使用的功能。

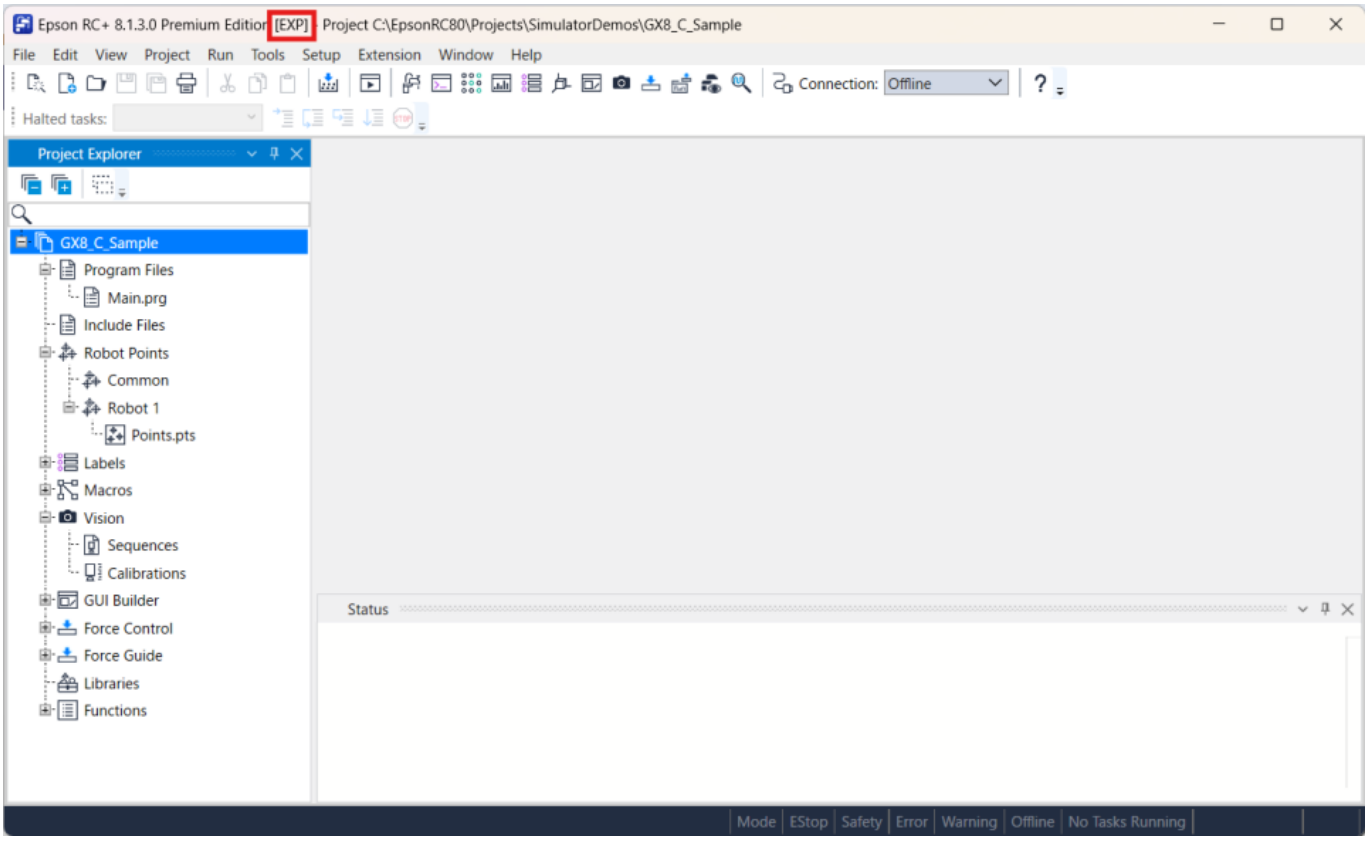


- **主菜单和工具栏项**：可以在 Epson RC+ 主窗口的菜单及工具栏中添加 Extension 专用菜单。
- **停靠窗口**：可以在 Epson RC+ 主窗口中添加 Extension 专用的停靠窗口。
- **项目文件编辑器**：可以在项目管理下添加 Extension 专用的专有文件。
- **项目浏览器项**：可以在项目浏览器中添加 Extension 专用的项目。
- **外部函数**：可以通过 SPEL+ 的 Declare 语句调用 Extension 专用的处理。

要点

后续如需其他功能的文件模板，可以进行添加。右键点击解决方案资源管理器，选择 [添加]-[新建项]，并通过 [RC+] 进行搜索和添加。

创建完成的项目可直接进行构建，并在 RC+ 安装状态下进行调试。
通过选择菜单-[调试]-[开始调试]，可以构建 Extension，完成后 Epson RC+ 将启动。Epson RC+ 启动后，请确认所选功能（窗口或菜单）是否显示。主窗口标题中将显示 [EXP]，表示 Extension 的调试状态。



RC+ 启动时，如存在已运行的 RC+ 实例，请先关闭。

我们已在 GitHub 提供 API 参考文档及示例程序。请参考有关程序实现方法的详细信息。请参阅以下 URL。

https://github.com/Epson-Robots/rcplus-extensions

5.4 教程

本节将按照步骤具体创建 Extension。
为便于阅读，部分编辑文件的内容可能仅展示部分片段。有关文件的完整内容，请参考我们在 GitHub 上提供的全部源代码。
https://github.com/Epson-Robots/rcplus-extensions

本教程假定用户已掌握 Visual Studio 的基本操作以及 C# 等相关开发基础知识，按此前提介绍操作步骤。

本教程可创建的 Extension:

- [网络摄像头录像器]
- [简易Jog]

5.4.1 网络摄像头录像器

在此，我们将尝试创建一个利用作为可连接至 PC 的外部设备之一的网络摄像头的 Extension。

首先，在RC+的停靠窗口中，实现网络摄像头图像预览功能（初级篇）。

接下来，添加图像录制功能（中级篇）。随着 SPEL+ 程序的启动开始录制，程序结束时停止录制。考虑到程序可能长时间运行，每隔 5 秒创建一个新文件进行录制，并只保留最新的 2 个文件。

这旨在为系统添加类似于汽车行车记录仪的功能。在设备启动过程中等场景下，通过监控机器人作业，一旦程序意外停止，可以通过录制的视频后续目视确认发生了什么情况。

如果 PC 已连接至网络，也可以实现如发送异常通知等与录制数据结合的应用。

那么，现在开始吧。

■ 初级篇

1. 按照［开始使用］的步骤，创建新的 Extension 项目。
   - 项目名称设为 WebCamRecorder。
   - 初始功能，勾选**Main menu and tool bar item**和**Docking window**。
   - 在 ARM64 版 Windows 上，请将配置设置为 x64。
2. 在 Visual Studio 上进行构建和调试，确认功能正常运行。
   - 在 RC+ 主菜单的扩展标签下，菜单项中会新增 `WebCamRecorder(xx)` (xx 为显示语言名)，选择菜单项后若能显示停靠窗口则表示操作成功。
3. 模板确认已完成，暂时关闭 RC+。
4. 在 Visual Studio 的解决方案资源管理器中，双击 WebCamRecorder 项目，并进行以下修改。
   - 将 TargetFramework 修改为 net8.0-windows10.0.19041.0。
     - 这样，便可使用 Windows Media Foundation 的 API。以下修改也与此相关。Windows Media Foundation 是作为 DirectShow 的后继，从 Windows Vista 起标准搭载于操作系统中的基于 COM 的 API 集合。目前，.NET 的标准库尚未纳入该 API，但通过使用如 Microsoft.Windows.CsWin32 等工具，可以像标准库一样进行调用。
   - 添加 `<EnableWindowsTargeting>true</EnableWindowsTargeting>` 行。(在 `<PropertyGroup>` 内)
   - 添加 `<AllowUnsafeBlocks>true</AllowUnsafeBlocks>` 行。
5. 选择“工具”>“NuGet 包管理器”>“管理解决方案的 NuGet 包”，打开界面。
   - 在“引用”标签下，搜索 Microsoft.Windows.CsWin32 包，安装最新版稳定版 (本书验证版本为 v0.3.264)。
     - Microsoft.Windows.CsWin32 是用于简化 C# 调用 Windows API 的库。详情见 https://github.com/microsoft/CsWin32 。
6. 在 Visual Studio 的项目中，添加 NativeMethods.txt 和 NativeMethods.json 文件。
   - 这些文件是使用 Microsoft.Windows.CsWin32 调用 Windows API 所必需。
   - NativeMethods.txt

```
MFStartup
MFShutdown
MFCreateAttributes
MFEnumDeviceSources
MFCreateSourceReaderFromMediaSource
MFCreateMediaType
MFCreateSinkWriterFromURL
MFCreateSample
MFCreateAlignedMemoryBuffer

(中略)

CoInitializeEx
CoTaskMemFree
```

```
COINIT
```

- NativeMethods.json

```
{
    "$schema": "https://aka.ms/CsWin32.schema.json",
    "public": true
}
```

7. 向项目中添加以下文件。

- CameraInfo.cs
  - 该文件用于编写表示摄像头的类 CameraInfo。

```
(前略)

namespace WebCamRecorder
{
    /// <summary>
    /// Camera information
    /// </summary>
    public class CameraInfo
    {
        /// <summary>
        /// Friendly name (may not be unique)
        /// </summary>
        public string FriendlyName { get; }

        /// <summary>
        /// Unique symbolic link
        /// </summary>
        public string SymbolicLink { get; }

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="friendlyName">Friendly name</param>
        /// <param name="symbolicLink">Symbolic link</param>
        public CameraInfo(
            string friendlyName,
            string symbolicLink
        )
        {
            FriendlyName = friendlyName;
            SymbolicLink = symbolicLink;
        }
    }
}
```

- CameraInfoCollection.cs
  - 该文件用于编写表示摄像头列表，并获取指定摄像头媒体源（在 Windows Media Foundation 中作为数据处理入口对象）的类 CameraInfoCollection。

```
(前略)

namespace WebCamRecorder
{
    (中略)

    /// <summary>
    /// Camera collection object
    /// </summary>
    public sealed class CameraInfoCollection : IDisposable
    {
        /// <summary>
        /// Camera information
        /// </summary>
        public List<CameraInfo> CameraInfos = [];

        /// <summary>
        /// Source activates
        /// </summary>
        private unsafe IMFActivate_unmanaged** _sourceActivates;

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="sourceActivates">Source activates</param>
        public unsafe CameraInfoCollection(
            IMFActivate_unmanaged** sourceActivates
        )
        {
            _sourceActivates = sourceActivates;
        }

        /// <summary>
        /// Create media source for the specified camera
        /// </summary>
        /// <param name="cameraInfo">Selected camera</param>
        /// <returns>Media source object</returns>
        public unsafe IMFMediaSource? GetMediaSource(
            CameraInfo cameraInfo
        )
        {
            var index = CameraInfos.FindIndex(
                (x) => (
                    x != null
                    && x.FriendlyName == cameraInfo.FriendlyName
                    && x.SymbolicLink == cameraInfo.SymbolicLink
                )
            );
            if (index < 0)
            {
                return null;
            }
```

```
            else
            {
                if
(Marshal.GetObjectForIUnknown((nint)_sourceActivates[index]) is not
IMFActivate managedSourceActivate)
                {
                    return null;
                }

                var mediaSource =
managedSourceActivate.ActivateObject(typeof(IMFMediaSource).GUID) as
IMFMediaSource;

                Marshal.ReleaseComObject(managedSourceActivate);

                return mediaSource;
            }
        }

        (后略)
```

- IFrameProcessor.cs
  - 该文件用于编写处理从摄像头获取图像的接口 IFrameProcessor。

```
(前略)

namespace WebCamRecorder
{
    /// <summary>
    /// Frame processor interface
    /// </summary>
    public interface IFrameProcessor
    {
        /// <summary>
        /// Initialize the processor
        /// </summary>
        /// <param name="width">Frame width</param>
        /// <param name="height">Frame height</param>
        /// <param name="stride">Frame stride</param>
        /// <param name="bitRate">Bit rate</param>
        public void Initialize(
            uint width,
            uint height,
            uint stride,
            uint bitRate
        );

        /// <summary>
        /// Termniate the processor
        /// </summary>
        public void Terminate();

        /// <summary>
```

```
        /// Process the frame
        /// </summary>
        /// <param name="frame">Frame data</param>
        /// <param name="duration">Duration</param>
        public void Process(
            byte[] frame,
            long duration
        );

        /// <summary>
        /// Request stopping
        /// </summary>
        public void Stop();

        /// <summary>
        /// The processing is currently stopping or not
        /// </summary>
        public bool IsStopped { get; }
    }
}
```

- CameraManager.cs
  - 该文件用于编写 CameraManager 类，实现拍摄并依次将图像数据传递给图像处理器（即实现上述 IFrameProcessor 的类实例）。

```
(前略)

namespace WebCamRecorder
{
    (中略)

    /// <summary>
    /// Camera manager
    /// </summary>
    public class CameraManager
    {
        /// <summary>
        /// List of frame processors
        /// </summary>
        public List<IFrameProcessor> FrameProcessors { get; } = [];

        (中略)

        /// <summary>
        /// List available cameras
        /// </summary>
        /// <returns>Collection object</returns>
        public unsafe CameraInfoCollection? ListCameras()
        {

        (中略)

        /// <summary>
```

```
        /// Read a frame from source
        /// </summary>
        /// <param name="sourceReader">Source reader object</param>
        /// <param name="frame">Buffer</param>
        /// <param name="duration">Variable to get duration</param>
        /// <returns>1: got it, 0: not got, -1: error</returns>
        private static unsafe int ReadFrame(
            IMFSourceReader sourceReader,
            byte[] frame,
            out long duration
        )
        {
        (中略)

        /// <summary>
        /// Start the processings
        /// </summary>
        /// <param name="cameraInfo">Selected camera</param>
        /// <returns>Task</returns>
        public async Task Start(
            CameraInfo cameraInfo
        )
        {
            while (!_done)
            {
                const int _waitMSec = 10;

                await Task.Delay(_waitMSec);
            }

            await Task.Run(() =>
            {
                HRESULT hr;

                hr = PInvoke.CoInitializeEx(COINIT.COINIT_MULTITHREADED);
                if (hr.Failed)
                {
                    return;
                }

                hr = PInvoke.MFStartup(PInvoke.MF_VERSION,
PInvoke.MFSTARTUP_FULL);
                if (hr.Succeeded)
                {
                    _done = false;

                    var sourceReader = CreateSourceReader(cameraInfo);
                    if (sourceReader != null)
                    {
                        GetVideoInfos(
                            sourceReader,
                            out var width,
                            out var height,
```

```
                            out var stride,
                            out var bitRate
                        );

                        var frame = new byte[stride * height];

                        foreach (var frameProcessor in FrameProcessors)
                        {
                            frameProcessor.Initialize(width, height, stride,
bitRate);
                        }

                        _stopping = false;
                        while (true)
                        {
                            var status = ReadFrame(sourceReader, frame, out
var duration);
                            if (status < 0)
                            {
                                break;
                            }
                            else if (status > 0)
                            {
                                foreach (var frameProssor in FrameProcessors)
                                {
                                    frameProssor.Process(frame, duration);
                                }
                            }

                            if (_stopping && FrameProcessors.All(x =>
x.IsStopped))
                            {
                                break;
                            }
                        }

                        foreach (var frameProcessor in FrameProcessors)
                        {
                            frameProcessor.Terminate();
                        }

                        Marshal.ReleaseComObject(sourceReader);
                    }

                    _ = PInvoke.MFShutdown();

                    _done = true;
                    _stoppedAction?.Invoke();
                }
            });
        }
```

```
        (后略)
```

- Previewer.cs
  - 该文件用于编写用于预览摄像头图像的类 Previewer。
    - PreviewImage 是用于在窗口显示图像的Image控件。初始化时创建位图数据，并设置为 PreviewImage 的 Source，之后将 CameraManager 传递的图像数据写入位图。

```
(前略)

namespace WebCamRecorder
{
    (中略)

    /// <summary>
    /// Image previewer for the camera
    /// </summary>
    public class Previewer : IFrameProcessor
    {
        /// <summary>
        /// Image control
        /// </summary>
        public Image? PreviewImage;

        /// <summary>
        /// Bitmap
        /// </summary>
        private WriteableBitmap? _bitmap;

        (中略)

        /// <inheritdoc />
        public void Initialize(
            uint width,
            uint height,
            uint stride,
            uint bitRate
        )
        {
            _width = (int)width;
            _height = (int)height;
            _stride = (int)stride;

            Application.Current.Dispatcher.Invoke(() =>
            {
                _bitmap = new(
                    _width, _height,
                    96, 96,
                    PixelFormats.Bgr32,
                    null
                );

                if (PreviewImage != null)
                {
```

```
                    PreviewImage.Source = _bitmap;
                }
            });
        }

        (中略)

        /// <inheritdoc />
        public void Process(
            byte[] frame,
            long duration
        )
        {
            if (_bitmap == null)
            {
                return;
            }

            Application.Current.Dispatcher.Invoke(() =>
            {
                _bitmap.WritePixels(
                    new Int32Rect(0, 0, _width, _height),
                    frame,
                    _stride,
                    0
                );
            });
        }

        (后略)
```

8. 编辑 DockingWindow 文件夹下的 DockingWindowContent.xaml 文件。

   - 考虑到图像可能大于窗口，配置 ScrollViewer，并将 DockPanel 移至其内部。
   - 删除原有的 TextBlock 和 Grid。
   - 添加包含 Label 和 ComboBox 的 StackPanel。
     - 在本 Extension 中，显示 ComboBox 下拉菜单时（若已执行），会停止拍摄处理，并获取已连接摄像头的列表。从下拉菜单选择摄像头后，开始拍摄处理。这些处理稍后会描述，预期将以下属性和命令添加至视图模型，并绑定至相应位置。
       - 表示摄像头列表的 Cameras(`ReactiveCollection<CameraInfo>`)
       - 表示所选摄像头在列表中的索引 SelectedCameraIndex(`ReactivePropertySlim<int>`)
       - 用于(重新)获取摄像头列表的命令 RefreshCamerasCommand(ReactiveCommand)
     - 请关注 Label 的 Content。绑定至 Captions[CaptionCamera].Value。在 Extension 中，需根据 RC+ 的显示语言对需本地化的字符串进行管理，相关内容需记录在 Captions.xlsx 文件中。详细内容后述，通过在 Captions.xlsx 的 symbol 列定义名称（此处为 CaptionCamera），以同样方式绑定，实现 RC+ 显示语言下的本地化。
   - 添加名为 PreviewImage 的 Image 控件。

```
<UserControl x:Class="WebCamRecorder.DockingWindow.DockingWindowContent"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```
            xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
            xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
            xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
            xmlns:i="http://schemas.microsoft.com/xaml/behaviors"
            xmlns:local="clr-namespace:WebCamRecorder.DockingWindow"
            mc:Ignorable="d"
            d:DesignHeight="450" d:DesignWidth="800">

    <UserControl.DataContext>
        <local:DockingWindowContentViewModel />
    </UserControl.DataContext>

    <ScrollViewer
        VerticalScrollBarVisibility="Auto"
        HorizontalScrollBarVisibility="Auto">

        <DockPanel
            Background="White"
            LastChildFill="True">

            <StackPanel
                DockPanel.Dock="Top"
                Orientation="Horizontal"
                Margin="10">

                <Label
                    Content="{Binding Captions[CaptionCamera].Value}" />
                <ComboBox
                    ItemsSource="{Binding Cameras}"
                    SelectedIndex="{Binding SelectedCameraIndex.Value}"
                    IsReadOnly="True"
                    DisplayMemberPath="FriendlyName"
                    MinWidth="200"
                    Margin="10,0,0,0">
                    <i:Interaction.Triggers>
                        <i:EventTrigger
                            EventName="DropDownOpened">
                            <i:InvokeCommandAction
                                Command="{Binding RefreshCamerasCommand}" />
                        </i:EventTrigger>
                    </i:Interaction.Triggers>
                </ComboBox>

            </StackPanel>

            <Image
                x:Name="PreviewImage"
                Width="640"
                Height="480"
                Stretch="UniformToFill"
                HorizontalAlignment="Left" />
```

```
        </DockPanel>
    </ScrollViewer>

</UserControl>
```

9. 编辑 DockingWindow 文件夹下的 DockingWindowContentViewModelAddition.cs 文件。
   - DockingWindow 文件夹中还包含 DockingWindowContentViewModel.cs 文件，将在这两个文件中描述 DockingWindowContentViewModel 类。
     - DockingWindowContentViewModel.cs 包含关闭、保存等功能，以及内容编辑用的复制、剪切、粘贴等方法，可按需编写处理逻辑。

       –

       在本 Extension 中，仅在窗口关闭时添加停止拍摄处理的逻辑。

       ```
       (前略)

       /// <inheritdoc />
       public Task<bool> CloseAsync()
       {
           _cameraManager.Stop();

           return Task.FromResult(true);
       }

       (后略)
       ```

     - DockingWindowContentViewModelAddition.cs 包含视图模型构造函数和窗口生成后仅调用一次的 WindowCreated 方法。将窗口专有的属性、命令及相关 API 调用集中编写于此文件，可提升视图模型整体的可读性。

       ```
       (前略)

       namespace WebCamRecorder.DockingWindow
       {
           (中略)

           /// <summary>
           /// Extension : Docking Window (Specific Part)
           /// </summary>
           internal partial class DockingWindowContentViewModel
           {
               /// <summary>
               /// Camera list
               /// </summary>
               public ReactiveCollection<CameraInfo> Cameras { get; } = [];

               /// <summary>
               /// Index of the selected camera
               /// </summary>
               public ReactivePropertySlim<int> SelectedCameraIndex { get; } =
       new(-1);
       ```

```
        /// <summary>
        /// Refresh camera list command
        /// </summary>
        public ReactiveCommand RefreshCamerasCommand { get; } = new();

        (中略)

        /// <summary>
        /// Camera manager
        /// </summary>
        private readonly CameraManager _cameraManager = new();

        /// <summary>
        /// Previewer
        /// </summary>
        private readonly Previewer _previewer = new();

        (中略)

        /// <summary>
        /// Refresh camera list
        /// </summary>
        private void OnRefreshCameras()
        {
            SelectedCameraIndex.Value = -1;

            Cameras.Clear();
            var cameraInfoCollection = _cameraManager.ListCameras();
            if (cameraInfoCollection != null)
            {
                foreach (var cameraInfo in cameraInfoCollection.CameraInfos)
                {
                    Cameras.Add(cameraInfo);
                }
                cameraInfoCollection.Dispose();
            }
        }

        /// <summary>
        /// Change camera
        /// </summary>
        /// <param name="index">The index of the selected camera</param>
        /// <returns>Task</returns>
        private async Task OnSelectedCameraChanged(
            int index
        )
        {
            _cameraManager.Stop();

            if (index >= 0)
            {
                await _cameraManager.Start(Cameras[index]);
            }
```

```
        }

        /// <summary>
        /// Set image control for previewer
        /// </summary>
        /// <param name="previewImage">Image control for previewing</param>
        public void SetPreviewImage(
            Image previewImage
        )
        {
            _previewer.PreviewImage = previewImage;
        }

        /// <summary>
        /// Constructor
        /// </summary>
        public DockingWindowContentViewModel()
        {
            _cameraManager.FrameProcessors.Add(_previewer);


RefreshCamerasCommand.Subscribe(OnRefreshCameras).AddTo(_disposables);

            SelectedCameraIndex.Subscribe(async (index) =>
            {
                await OnSelectedCameraChanged(index);
            })
            .AddTo(_disposables);
        }

        (后略)
```

10. 编辑 DockingWindow 文件夹下的 DockingWindowContent.xaml.cs 文件。

    - 将用于预览的 Image 控件的引用传递给视图模型。

```
(前略)

if (DataContext is DockingWindowContentViewModel viewModel)
{
    viewModel.SetPreviewImage(PreviewImage);
}

(后略)
```

11. 打开并编辑 Captions.xlsx 文件。

- 如前所述，需根据 RC+ 显示语言对需本地化的字符串进行管理，相关内容需记录在此文件中。

	A	B	C	D	E	F	G	H
1	ID	description	symbol	DEL	English	Japanese	German	French
2					English_en	Japanese_ja	German_de	French_fr
3								
4	0	Exttension Name	ExtensionName		WebCamRecorder	ウェブカメラレコーダー	WebCamRecorder	WebCamEnregistreur
5	1	Main Menu & ToolBar Item	MainMenu		WebCam Recorder	ウェブカメラレコーダー	WebCam Recorder	Enregistreur de WebC
6								
7	400	Docking Window Title	WindowTitle		WebCam Recorder	ウェブカメラレコーダー	WebCam Recorder	Enregistreur de WebC
8	401	"Camera:" Caption	CaptionCamera		Camera:	カメラ:	Kamera:	Caméra:

  - ID 为标题编号。请确保在此文件中编号不重复。
  - description 为注释。可自由填写。
  - symbol 为 Extension 源代码（.xaml、.cs）中引用的名称。编辑 Captions.xlsx 文件并构建项目后，会生成将 symbol 与 ID 关联的常量定义文件 Captions.cs。请勿直接编辑该文件。

```
// <auto-generated>

namespace WebCamRecorder
{
    using System.Reflection;

    internal class Constants
    {
        internal class Caption
        {
            (中略)

            public const int ExtensionName = 0;
            public const int MainMenu = 1;
            public const int WindowTitle = 400;
            public const int CaptionCamera = 401;
        }
    }
}
```

  - English、Japanese 等各列需填写对应语言的显示字符串。

12. 进行构建和调试。
    - 连接 PC 的网络摄像头，显示 WebCamRecorder 窗口，选择摄像头并显示图像即为成功。

■ 中级篇

在初级篇中，除生成的解决方案所用部分外，并未使用 Extensions API。

在中级篇中，将尝试使用提供下述功能的 Extensions API。

- 获取已打开项目的项目文件夹路径（项目 API）。
- 获取 SPEL+ 的任务列表（程序执行 API）。

结合丰富的 .NET 库和 Windows API，并使用必要的 Extensions API，可创建与 RC+ 和 SPEL+ 程序紧密联动的专属应用作为 Extension 使用。

那么，继续操作。

1. 若 RC+ 已启动，请先关闭，启动 Visual Studio 并打开初级篇创建的解决方案。

2. 添加 Recorder.cs 文件。

   - Recorder 类与 Previewer 类一样，实现了 IFrameProcessor。Recorder 会根据 CameraManager 传递的图像数据生成 H.264 格式的视频文件。
   - 视频文件约每 5 秒在 Recorder 实例指定的文件夹下新建，命名为 `Video_N.mp4` (N 为 000～999，达到 999 后重新从 000 开始)，为避免占用过多存储，仅保留最新的 2 个文件。
     - 本 Extension 在开始新录制时，会删除文件夹内所有视频。
   - 此外，为实现类似行车记录仪的功能，在指示停止录制后，仍会继续录制 2 秒。(即最后保存的视频最长约为 7 秒。)
   - 录制模式包括与 SPEL+ 程序启动、结束联动的自动模式 (Auto)，以及可在任意时机开始、停止的手动模式 (Manual)。
   - 由于摄像头图像预览无需显式启动 (选择摄像头即开始)，IFrameProcessor 仅提供停止指令的 Stop 方法。因此，Recorder 需通过 Mode 属性持有上述录制模式，并根据 Mode 设置启动录制。此外，Mode 更改时会触发 PropertyChanged 事件，便于程序捕捉录制实际停止的时机。

```
(前略)

namespace WebCamRecorder
{
    (中略)

    /// <summary>
    /// Recorder
    /// </summary>
    public class Recorder : IFrameProcessor, INotifyPropertyChanged
    {
        /// <summary>
        /// Recording mode definitions
        /// </summary>
        public enum RecordingMode
        {
            Stop,
            Auto,
            Manual,
        }

        /// <inheritdoc />
        public event PropertyChangedEventHandler? PropertyChanged;

        /// <summary>
        /// Recording mode
        /// </summary>
        public RecordingMode Mode
        {
            get
            {
                return _mode;
            }
```

```
            set
            {
                if (_sinkWriter == null)
                {
                    _shouldStop = false;

                    _mode = value;
                    RaisePropertyChanged();
                }
            }
        }

        /// <summary>
        /// Folder for video files
        /// </summary>
        public string VideoFolder
        {
            get
            {
                return _videoFolder;
            }
            set
            {
                if (_sinkWriter == null)
                {
                    try
                    {
                        Directory.CreateDirectory(value);
                        _videoFolder = value;
                    }
                    catch (Exception)
                    {
                        // EMPTY
                    }
                }
            }
        }

        (中略)

        /// <inheritdoc />
        public void Process(
            byte[] frame,
            long duration
        )
        {
            if (_sinkWriter == null)
            {
                if (_mode == RecordingMode.Stop)
                {
                    return;
                }
```

```
                _sinkWriter = CreateSinkWriter(GetNextSegmentFile());

                _recordTime = 0;
                _segmentSpan = _initialSegmentSpan;
            }

            if (_sinkWriter != null && _sample != null)
            {
                SetFlippedFrame(frame);

                _sample.SetSampleTime(_recordTime);
                _sample.SetSampleDuration(duration);

                _sinkWriter.WriteSample(_streamIndex, _sample);

                _recordTime += duration;
                if (_recordTime > _segmentSpan)
                {
                    _sinkWriter.Finalize();
                    Marshal.ReleaseComObject(_sinkWriter);
                    _sinkWriter = null;

                    if (_shouldStop)
                    {
                        Mode = RecordingMode.Stop;
                    }
                }
            }
        }

        /// <inheritdoc />
        public void Stop()
        {
            const long _minAdditionalTime = 20_000_000;

            if (_segmentSpan - _recordTime < _minAdditionalTime)
            {
                _segmentSpan = _recordTime + _minAdditionalTime;
            }

            _shouldStop = true;
        }

        (后略)
```

3. 编辑 DockingWindow 文件夹下的 DockingWindowContent.xaml 文件。

   - 在界面上，添加录制中指示器显示，并增加按钮以便手动 (Manual 模式) 启动和停止录制。
   - 视图模型后续将添加以下属性和命令。
     - 表示录制中的 IsRecording(`ReactivePropertySlim<bool>`)

- 表示可启动录制的 CanStartRecording(`ReactivePropertySlim<bool>`) 及启动录制命令 StartRecordingCommand(`ReactiveCommand`)
- 表示可停止录制的 CanStopRecording(`ReactivePropertySlim<bool>`) 及停止录制命令 StopRecordingCommand(`ReactiveCommand`)
- 本 Extension 中，自动模式录制时不可手动启动或停止录制，反之手动模式录制时自动模式录制无效。
  - 但无论哪种模式，关闭停靠窗口均会停止录制。

```
(前略)

</ComboBox>
<Border
    CornerRadius="10"
    Width="60"
    Height="20"
    Margin="20,0,0,0"
    VerticalAlignment="Center">
    <TextBlock
        Text="REC"
        HorizontalAlignment="Center"
        VerticalAlignment="Center">
        <TextBlock.Style>
            <Style
                TargetType="TextBlock">
                <Style.Triggers>
                    <DataTrigger
                        Binding="{Binding IsRecording.Value}"
                        Value="True">
                        <Setter
                            Property="Foreground"
                            Value="White" />
                    </DataTrigger>
                    <DataTrigger
                        Binding="{Binding IsRecording.Value}"
                        Value="False">
                        <Setter
                            Property="Foreground"
                            Value="Black" />
                    </DataTrigger>
                </Style.Triggers>
            </Style>
        </TextBlock.Style>
    </TextBlock>
    <Border.Style>
        <Style
            TargetType="Border">
            <Style.Triggers>
                <DataTrigger
                    Binding="{Binding IsRecording.Value}"
                    Value="True">
                    <Setter
                        Property="Background"
```

```
                        Value="Red" />
                </DataTrigger>
                <DataTrigger
                    Binding="{Binding IsRecording.Value}"
                    Value="False">
                    <Setter
                        Property="Background"
                        Value="LightGray" />
                </DataTrigger>
            </Style.Triggers>
        </Style>
    </Border.Style>
</Border>
<Button
    Command="{Binding StartRecordingCommand}"
    IsEnabled="{Binding CanStartRecording.Value}"
    Content="{Binding Captions[LabelStart].Value}"
    Width="80"
    VerticalAlignment="Center"
    Margin="10,0,0,0" />
<Button
    Command="{Binding StopRecordingCommand}"
    IsEnabled="{Binding CanStopRecording.Value}"
    Content="{Binding Captions[LabelStop].Value}"
    Width="80"
    VerticalAlignment="Center"
    Margin="10,0,0,0" />
```

4. 编辑 DockingWindow 文件夹下的 DockingWindowContentViewModelAddition.cs 文件。

   - 添加在视图（.xaml）绑定的属性和命令。
   - Recorder 类实例也添加至 CameraManager。
   - 请关注 WindowCreated 方法。此处使用 Extensions API 的项目 API，获取已打开项目的项目文件夹(路径名)。
     - API 对象通过 Main.GetAPI 方法获取。
     - 项目 API 对象的 ProjectFolder 属性为已打开项目的项目文件夹路径名。若未打开项目，则为 null。
       - 未打开项目时，将使用 Windows 登录用户的“视频”文件夹代替项目文件夹。
     - 录制文件将保存在项目文件夹或登录用户的“视频”文件夹下新建的 WebCamRecorder 子文件夹中，并设置给 Recorder。

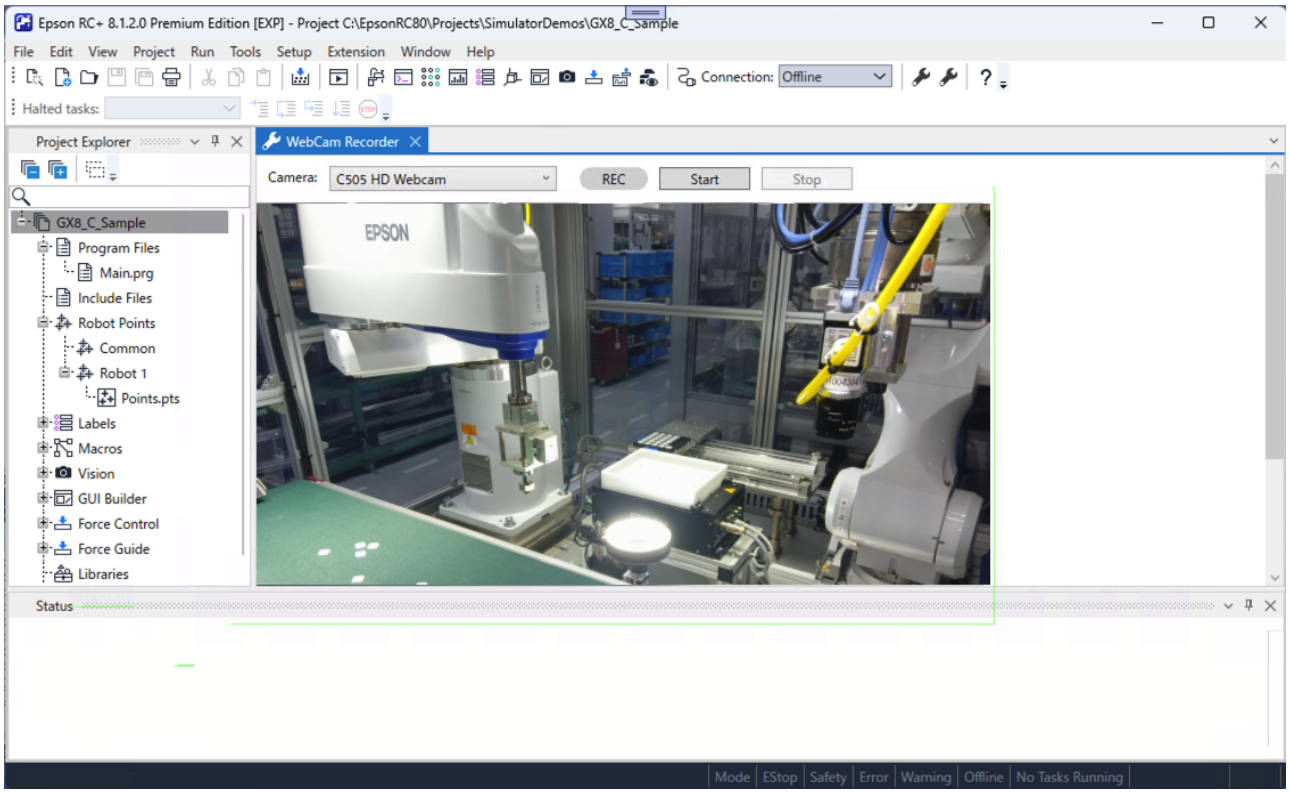
```
(前略)
/// <summary>
/// Recording in progress or not
/// </summary>
public ReactivePropertySlim<bool> IsRecording { get; } = new(false);

/// <summary>
/// Can start recording or not
/// </summary>
public ReactivePropertySlim<bool> CanStartRecording { get; } = new(false);
```

```
/// <summary>
/// Start recording command
/// </summary>
public ReactiveCommand StartRecordingCommand { get; }

/// <summary>
/// Can stop recording or not
/// </summary>
public ReactivePropertySlim<bool> CanStopRecording { get; } = new(false);

/// <summary>
/// Stop recording command
/// </summary>
public ReactiveCommand StopRecordingCommand { get; }

(中略)

/// <summary>
/// Recorder
/// </summary>
private readonly Recorder _recorder = new();

(中略)

/// <summary>
/// Change camera
/// </summary>
/// <param name="index">The index of the selected camera</param>
/// <returns>Task</returns>
private async Task OnSelectedCameraChanged(
    int index
)
{
    EnableOrDisableRecordingCommands();

    _cameraManager.Stop();

    if (index >= 0)
    {
        await _cameraManager.Start(Cameras[index]);
    }
}

(中略)

/// <summary>
/// Update recording command possibilities
/// </summary>
private void EnableOrDisableRecordingCommands()
{
    CanStartRecording.Value = (SelectedCameraIndex.Value >= 0 &&
_recorder.Mode == Recorder.RecordingMode.Stop);
    CanStopRecording.Value = (_recorder.Mode ==
```

```
Recorder.RecordingMode.Manual);
}
/// <summary>
/// Start recording
/// </summary>
private void OnStartRecording(
    bool isAuto
)
{
    if (_recorder.Mode == Recorder.RecordingMode.Stop)
    {
        try
        {
            var files = Directory.EnumerateFiles(
                _recorder.VideoFolder,
                $"*{Recorder.VideoFileExtension}"
            );
            foreach (var file in files)
            {
                File.Delete(file);
            }
        }
        catch (Exception)
        {
            // IGNORE
        }
        _recorder.Mode = isAuto ? Recorder.RecordingMode.Auto :
Recorder.RecordingMode.Manual;

        EnableOrDisableRecordingCommands();
    }
}

/// <summary>
/// Stop recording
/// </summary>
private void OnStopRecording()
{
    _recorder.Stop();

    CanStopRecording.Value = false;
}

/// <summary>
/// Constructor
/// </summary>
public DockingWindowContentViewModel()
{
    _cameraManager.FrameProcessors.Add(_previewer);
    _cameraManager.FrameProcessors.Add(_recorder);

    (中略)
```

```
    _recorder.PropertyChanged += (_, _) =>
    {
        IsRecording.Value = (_recorder.Mode != Recorder.RecordingMode.Stop);
        EnableOrDisableRecordingCommands();
    };

    (中略)

}

/// <inheritdoc />
public Task WindowCreated()
{
    string videoFolder;

    var projectAPI = Main.GetAPI<IRCXProjectAPI>();
    if (projectAPI != null && projectAPI.ProjectFolder != null)
    {
        videoFolder = projectAPI.ProjectFolder;
    }
    else
    {
        videoFolder =
Environment.GetFolderPath(Environment.SpecialFolder.MyVideos);
    }
    _recorder.VideoFolder = Path.Combine(videoFolder, "WebCamRecorder");

    (后略)
```

5. 进行构建和调试。
   - 打开停靠窗口，选择摄像头，尝试录制的开始与停止。
   -



- 在指定文件夹※中保存视频文件，并可播放即为成功。
  ※若已打开项目，则为项目文件夹下的“WebCamRecorder”文件夹
  未打开项目时，则为 Windows 登录用户的“视频”文件夹

6. 编辑 DockingWindow 文件夹下的 DockingWindowContentViewModelAddition.cs 文件。

- 添加自动模式录制功能。为此，需要在 Extension 中捕捉 SPEL+ 程序的执行开始和执行结束的时机。
  - SPEL+ 程序支持多任务处理。在本 Extension 中，将 SPEL+ 程序的开始与结束视为普通任务的开始与结束。
  - 任务列表可通过程序执行 API 对象的 Tasks 属性获取。
    - Tasks 是 `IEnumerable<IRCXTask>` 类型的集合，IRCXTask 实例拥有表示任务状态的 State 属性和表示任务类型的 Kind 属性。
    - 当 SPEL+ 的任务发生任何变更时，Tasks 属性的 PropertyChanged 事件会被触发。

- 请在 WindowCreated 方法中添加以下代码，用于监控正在执行的普通任务是否存在，并更新 `ReactivePropertySlim<bool>` 类型的 _isProgramRunning。

```
(前略)

/// <summary>
/// Program execution API object
/// </summary>
private IRCXProgramExecutionAPI? _programExecutionAPI;

/// <summary>
/// Program running state
/// </summary>
private readonly ReactivePropertySlim<bool> _isProgramRunning = new(false,
ReactivePropertyMode.DistinctUntilChanged);

(中略)

/// <inheritdoc />
public Task WindowCreated()
{
    (中略)

    _programExecutionAPI = Main.GetAPI<IRCXProgramExecutionAPI>();

    _programExecutionAPI?.ObserveProperty(x => x.Tasks).Subscribe((tasks) =>
    {
        _isProgramRunning.Value = tasks
        .Any(
            x => (
                x.Kind == IRCXProgramExecutionAPI.IRCXTask.RCXTaskKind.Normal
                && x.State ==
IRCXProgramExecutionAPI.IRCXTask.RCXTaskState.Run
            )
        );
    })
    .AddTo(_disposables);
```

```
(后略)
```

- 此外，在构造函数中添加代码，通过检测上述 _isProgramRunning 属性的变更，调用录制的开始和结束。

```
(前略)

/// <summary>
/// Constructor
/// </summary>
public DockingWindowContentViewModel()
{

(中略)
    _isProgramRunning.Subscribe((isRunning) =>
    {
        if (SelectedCameraIndex.Value >= 0)
        {
            if (isRunning)
            {
                OnStartRecording(isAuto: true);
            }
            else
            {
                OnStopRecording();
            }

            EnableOrDisableRecordingCommands();
        }
    })
    .AddTo(_disposables);
}

(后略)
```

7. 进行构建和调试。
   - 打开 Extension 的窗口，选择相机，并确认预览图像已显示。
   - 打开 Run 窗口并执行程序。
     - 若程序开始时录制启动，结束后约 2 秒录制停止，并且视频文件已保存到指定文件夹，则操作成功。

5.4.2 简易Jog

RC+ 配备了可通过机器人管理器等调用的全功能“微动&示教”，但在创建 Extension 时，可以只调用该“微动&示教”的必要功能，实现自定义微动面板（窗口）。
根据使用场景，通过创建自定义微动面板，有可能提升示教操作的效率。
此外，不仅限于自定义微动面板，通过 RC+ Extensions 对 RC+ 进行自定义，可以打造专属的 RC+，从而实现更加舒适的作业体验。

在本教程的初级篇中，将制作如下简单的微动面板，并说明功能的调用方式。

- 点击电机的“切换”按钮，可切换电机的开 / 关状态。
- 面板上可通过鼠标拖动移动，并分别在左右各配置一个类似游戏手柄“摇杆”风格的控制器。
  - 左侧摇杆可上下操作，实现沿 Z 坐标轴的微动。

  - 右侧摇杆可上下操作实现沿 Y 坐标轴的微动，左右操作实现沿 X 坐标轴的微动。
- 点击“Teach”按钮后，将机器人当前的位置姿态，在目标点文件中选择尚未定义的点，依次进行示教。
  - 对于点，将添加注释，记录在本 Extension 中进行示教的信息以及示教的日期时间。
  - 面板上将显示示教点的日志。

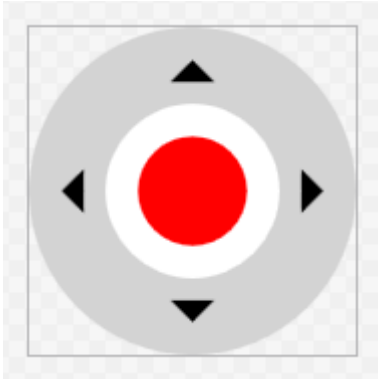
在中级篇中，若实际连接了游戏手柄，则可通过该游戏手柄进行操作。

- 电机的“切换”功能分配给左侧的肩部按钮（也称为 Shoulder）。
- 左右的“摇杆”风格控制器，将实现可通过实际摇杆进行操作。
- “Teach”分配给 A 按钮。

> ⚠ 注意
>
> **在机器人实机上测试此 Extension 时，请基于安全考量进行设计，并务必在安全护栏外侧操作。**

那么，现在开始吧。

■ 初级篇

1. 按照[使用开始]的步骤，创建新的 RC+ Extensions 项目。
   - 名称设为 SimpleJog。
   - 初始功能，勾选**Main menu and tool bar item**和**Docking window**。
   - 在 ARM64 版 Windows 上，请将配置更改为 x64。
2. 在 Visual Studio 上进行构建和调试，确认功能正常运行。
   - 在 RC+ 主菜单的扩展标签菜单项中，将新增 `SimpleJog (xx)` (xx 为显示语言名)，选择菜单项后显示停靠窗口即为正常。
3. 模板确认已完成，暂时关闭 RC+。
4. 在 DockingWindow 文件夹中添加如下文件。
   - Stick.xaml
     - 实现摇杆风格外观的用户控件文件。
       - 当控件处于激活状态时，中央“旋钮”显示为红色，并可通过鼠标拖动进行操作。
     - 

```
<UserControl x:Class="SimpleJog.DockingWindow.Stick"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```

```
            xmlns:i="http://schemas.microsoft.com/xaml/behaviors"
            xmlns:local="clr-namespace:SimpleJog.DockingWindow"
            mc:Ignorable="d"
            d:DesignHeight="300" d:DesignWidth="300">

    <Canvas
        Width="300"
        Height="300">

        (中略)

    </Canvas>

</UserControl>
```

- Stick.xaml.cs
  - 为 Stick 控件添加可通过鼠标移动“旋钮”的代码的代码后台文件。

```
(前略)

namespace SimpleJog.DockingWindow
{
    (中略)

    /// <summary>
    /// Stick.xaml interaction logic
    /// </summary>
    public partial class Stick : UserControl
    {
        (中略)

        /// <summary>
        /// Constructor
        /// </summary>
        public Stick()
        {
            InitializeComponent();

            Knob.Loaded += (_, _) =>
            {
                _radius = Math.Min(KnobRange.RenderSize.Width,
KnobRange.Height) / 2.0 * _limitFactor;
                _deadZone = _radius * _deadZoneFactor;
                _center = new Point(KnobRange.RenderSize.Width / 2.0,
KnobRange.RenderSize.Height / 2.0);
            };

            Knob.MouseLeftButtonDown += (_, ev) =>
            {
                Knob.CaptureMouse();

                _dragging = true;
```

```
                _offset = ev.GetPosition(KnobRange) - _center;
                _smoothed = new Vector();
            };

            (中略)
        }

        /// <summary>
        /// Update knob position
        /// </summary>
        /// <param name="mousePosInRange">Relative mouse position in knob
range</param>
        private void UpdateKnobPosition(
            Point mousePosInRange
        )
        {
            var x = mousePosInRange.X - _center.X;
            var y = mousePosInRange.Y - _center.Y;

            var distanceFromCenter = Math.Sqrt(x * x + y * y);
            if (distanceFromCenter < _deadZone)
            {
                Position = _smoothed = new Vector();
            }
            else if (distanceFromCenter < _radius)
            {
                _smoothed = new Vector(
                    _smoothed.X * (1 - _smoothingFactor) + (x / _radius) *
_smoothingFactor,
                    _smoothed.Y * (1 - _smoothingFactor) + (y / _radius) *
_smoothingFactor
                );
                Position = new Vector(_smoothed.X, -_smoothed.Y);
            }
        }
    }
}
```

- StickProperties.cs
  - 为 Stick 控件添加用于指示“旋钮”位置的 Vector 类型 Position 属性的文件。
    - Vector 的各要素 (X 和 Y) 会被归一化为 -1.0 到 +1.0 的值。

```
(前略)

namespace SimpleJog.DockingWindow
{
    using System.Windows;

    /// <summary>
    /// Stick.xaml dependency properties
    /// </summary>
    public partial class Stick
    {
```

```
        /// <summary>
        /// Normalized position
        /// </summary>
        public Vector Position
        {
            get => (Vector)GetValue(PositionProperty);
            set => SetValue(PositionProperty, value);
        }

        /// <summary>
        /// Field of the "Position"
        /// </summary>
        public static readonly DependencyProperty PositionProperty =
            DependencyProperty.Register(
                nameof(Position),
                typeof(Vector),
                typeof(Stick),
                new FrameworkPropertyMetadata(
                    default(Vector),
                    (FrameworkPropertyMetadataOptions.BindsTwoWayByDefault
                    | FrameworkPropertyMetadataOptions.AffectsRender),
                    OnPositionChanged,
                    CoercePositionNormalized
                )
            );

        /// <summary>
        /// Position changed event handler
        /// </summary>
        /// <param name="d">The object</param>
        /// <param name="ev">The event</param>
        private static void OnPositionChanged(
            DependencyObject d,
            DependencyPropertyChangedEventArgs ev
        )
        {
            if (d is Stick stick)
            {
                stick.UpdateRawPosition();
            }
        }

        /// <summary>
        /// Coerce value of the "Position"
        /// </summary>
        /// <param name="d">The object</param>
        /// <param name="value">The value</param>
        /// <returns>Corrected value</returns>
        private static object CoercePositionNormalized(
            DependencyObject d,
            object value
        )
        {
```

```
            var vector = (Vector)value;

            vector.X = Math.Clamp(vector.X, -1.0, 1.0);
            vector.Y = Math.Clamp(vector.Y, -1.0, 1.0);

            return vector;
        }

        /// <summary>
        /// Field key of the "RawPosition"
        /// </summary>
        private static readonly DependencyPropertyKey RawPositionPropertyKey =
            DependencyProperty.RegisterReadOnly(
                nameof(RawPosition),
                typeof(Vector),
                typeof(Stick),
                new PropertyMetadata(default(Vector))
            );

        /// <summary>
        /// Field of the "RawPosition"
        /// </summary>
        public static readonly DependencyProperty RawPositionProperty =
            RawPositionPropertyKey.DependencyProperty;

        /// <summary>
        /// Raw (pixel) position
        /// </summary>
        public Vector RawPosition => (Vector)GetValue(RawPositionProperty);

        /// <summary>
        /// Set raw position
        /// </summary>
        private void UpdateRawPosition()
        {
            var rawPosition = new Vector(Position.X * _radius, -(Position.Y *
_radius));

            SetValue(RawPositionPropertyKey, rawPosition);
        }
    }
}
```

5. 此处进行构建，使 Stick 能在 .xaml 的设计视图中显示。

6. 编辑 DockingWindow 文件夹下的 DockingWindowContent.xaml 文件。
   - 原有的 DockPanel 删除。
   - 改为配置 3 行 3 列的 Grid，并在各单元格中配置如下内容。以下以 0 为起点，R 行 C 列用 (R, C) 表示。
     - 第 3 行、第 3 列的 Height、Width 设为*。这些为边距，Grid 实际为 2 行 2 列。

- (0, 0): 配置 DockPanel，内部放置 Label“Motor:”、Border、Button、TextBlock。
  - Border 作为指示器，参考 IsMotorOn(`ReactivePropertySlim<bool>`) 和 MotorState(`ReactivePropertySlim<string>`)，用于指示电机状态。电机开启时显示绿色底白字“ON”，关闭时显示浅灰底黑字“OFF”。
  - Button 用于切换电机开 / 关。
    - Content 绑定到 Captions[LabelToggle].Value。
    - Command 绑定到 MotorToggleCommand(ReactiveCommand)。
    - IsEnabled 绑定到 IsOnline.Value。IsOnline(`ReactivePropertySlim<bool>`) 为与机器人控制器连接已建立时为 true 的标志。
  - TextBlock 的 Text 绑定到 APIResult.Value。APIResult(`ReactivePropertySlim<string>`) 用于本 Extension 的调试，显示所调用的 Extensions API 的状态 (RCXResult 类型) 的字符串表示。部分 API 除状态外还会返回附加信息，因此为显示该附加信息，准备了 APIResultAux(ReactivePropertySlim<string>)，并将 APIResultAux.Value 绑定到 ToolTip。
- (1, 0): 配置 3 行 4 列的 Grid，内部放置 6 个用于指示坐标方向的 Label 和 2 个 Stick。
  - Stick 用 Viewbox 包裹，以便可调整大小。
    - IsEnabled 绑定到 IsMotorOn.Value。
    - Position 关于左侧 Stick，绑定到 LeftStickPosition.Value。LeftStickPosition(`ReactivePropertySlim<Vector>`) 为左侧 Stick 的“旋钮”位置。右侧 Stick 同理。
- (0, 1): 为 Label。Content 绑定到 Captions[CaptionLogHeader].Value。
- (1, 1): 配置 DockPanel，内部放置 Button 和 ListBox。
  - Button 用于示教。
    - Content 绑定到 Captions[LabelTeach].Value。
    - Command 绑定到 TeachCommand(ReactiveCommand)。
    - IsEnabled 绑定到 CanTeach.Value。CanTeach(`ReactivePropertySlim<bool>`) 为是否可进行示教的标志。
  - ListBox 用于记录示教点信息的日志。
    - ItemsSource 绑定到 LogItems(`ReactiveCollection<LogItem>`)。LogItem 稍后创建。
    - 为显示末尾新增的最新日志信息，设置 AutoScrollBehavior。AutoScrollBehavior 也将在后续创建。

```
<UserControl x:Class="SimpleJog.DockingWindow.DockingWindowContent"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:i="http://schemas.microsoft.com/xaml/behaviors"
             xmlns:local="clr-namespace:SimpleJog.DockingWindow"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800">

    <UserControl.DataContext>
        <local:DockingWindowContentViewModel />
    </UserControl.DataContext>

    <Grid
        Margin="10">
```

```
        <Grid.RowDefinitions>
            <RowDefinition Height="30" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>

        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>

        <DockPanel
            Grid.Row="0" Grid.Column="0"
            LastChildFill="True">

            <Label
                Content="Motor:"
                VerticalAlignment="Center" />

            <Border
                CornerRadius="10"
                Width="60"
                Height="20"
                Margin="10,0,0,0"
                VerticalAlignment="Center">
                <TextBlock
                    Text="{Binding MotorState.Value}"
                    HorizontalAlignment="Center"
                    VerticalAlignment="Center">
                    <TextBlock.Style>
                        <Style
                            TargetType="TextBlock">
                            <Style.Triggers>
                                <DataTrigger
                                    Binding="{Binding IsMotorOn.Value}"
                                    Value="True">
                                    <Setter
                                        Property="Foreground"
                                        Value="White" />
                                </DataTrigger>
                                <DataTrigger
                                    Binding="{Binding IsMotorOn.Value}"
                                    Value="False">
                                    <Setter
                                        Property="Foreground"
                                        Value="Black" />
                                </DataTrigger>
                        </Style.Triggers>
                        </Style>
                    </TextBlock.Style>
                </TextBlock>
                <Border.Style>
```

```
                    <Style
                        TargetType="Border">
                        <Style.Triggers>
                            <DataTrigger
                                Binding="{Binding IsMotorOn.Value}"
                                Value="True">
                                <Setter
                                    Property="Background"
                                    Value="#00bb00" />
                            </DataTrigger>
                            <DataTrigger
                                Binding="{Binding IsMotorOn.Value}"
                                Value="False">
                                <Setter
                                    Property="Background"
                                    Value="LightGray" />
                            </DataTrigger>
                        </Style.Triggers>
                    </Style>
                </Border.Style>
            </Border>

            <Button
                Command="{Binding MotorToggleCommand}"
                IsEnabled="{Binding IsOnline.Value}"
                Content="{Binding Captions[LabelToggle].Value}"
                Width="90"
                Margin="10,0,0,0"
                VerticalAlignment="Center" />

            <TextBlock
                Text="{Binding APIResult.Value}"
                ToolTip="{Binding APIResultAux.Value}"
                TextAlignment="Right"
                VerticalAlignment="Center"
                Margin="10,0,20,0" />

        </DockPanel>

        <Grid
            Grid.Row="1" Grid.Column="0"
            Margin="0,10,0,0">

            <Grid.Resources>
                <Style
                    TargetType="Label">
                    <Setter
                        Property="FontSize"
                        Value="16" />
                </Style>
            </Grid.Resources>

            <Grid.RowDefinitions>
```

```
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
            </Grid.RowDefinitions>

            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="Auto" />
            </Grid.ColumnDefinitions>

            <Label
                Grid.Row="0" Grid.Column="0"
                Content="+Z"
                HorizontalAlignment="Center" />
            <Label
                Grid.Row="2" Grid.Column="0"
                Content="-Z"
                HorizontalAlignment="Center" />
            <Viewbox
                Grid.Row="1" Grid.Column="0"
                Width="200">
                <local:Stick
                    IsEnabled="{Binding IsMotorOn.Value}"
                    Position="{Binding InputService.LeftStickPosition.Value}"
/>
            </Viewbox>

            <Label
                Grid.Row="1" Grid.Column="1"
                Content="-X"
                Margin="20,0,0,0"
                VerticalAlignment="Center" />
            <Label
                Grid.Row="1" Grid.Column="3"
                Content="+X"
                Margin="0,0,10,0"
                VerticalAlignment="Center" />
            <Label
                Grid.Row="0" Grid.Column="2"
                Content="+Y"
                HorizontalAlignment="Center" />
            <Label
                Grid.Row="2" Grid.Column="2"
                Content="-Y"
                HorizontalAlignment="Center" />
            <Viewbox
                Grid.Row="1" Grid.Column="2"
                Width="200">
                <local:Stick
                    IsEnabled="{Binding IsMotorOn.Value}"
                    Position="{Binding InputService.RightStickPosition.Value}"
```

```
/>
            </Viewbox>

        </Grid>

        <Label
            Grid.Row="0" Grid.Column="1"
            Content="{Binding Captions[CaptionLogHeader].Value}"
            VerticalAlignment="Center" />

        <DockPanel
            Grid.Row="1" Grid.Column="1"
            LastChildFill="True">

            <Button
                DockPanel.Dock="Bottom"
                Command="{Binding TeachCommand}"
                IsEnabled="{Binding CanTeach.Value}"
                Content="{Binding Captions[LabelTeach].Value}"
                Width="100"
                Margin="0,10,0,0"
                HorizontalAlignment="Center" />

            <ListBox
                x:Name="TeachingLog"
                ItemsSource="{Binding LogItems}"
                Width="200">
                <i:Interaction.Behaviors>
                    <local:AutoScrollBehavior />
                </i:Interaction.Behaviors>
            </ListBox>

        </DockPanel>

    </Grid>

</UserControl>
```

7. 在 DockingWindow 文件夹中创建 LogItem.cs 文件。

   - 在本 Extension 中，示教日志将显示点编号及世界坐标的 X、Y、Z 值。

```
(前略)

namespace SimpleJog.DockingWindow
{
    using static Epson.RoboticsShared.ExtensionsAPI.IRCXRobotManagerAPI;

    /// <summary>
    /// Teaching log list box item
    /// </summary>
    public class LogItem
    {
```

```
        /// <summary>
        /// Point number
        /// </summary>
        public int PointNumber { get; }

        /// <summary>
        /// Point position
        /// </summary>
        public IDictionary<RCXJogCartesianAxis, double>? WorldPosition { get;
}

        /// <inheritdoc />
        public override string ToString()
        {
            if (WorldPosition == null)
            {
                return $"P{PointNumber}";
            }
            else
            {
                var x = WorldPosition[RCXJogCartesianAxis.X];
                var y = WorldPosition[RCXJogCartesianAxis.Y];
                var z = WorldPosition[RCXJogCartesianAxis.Z];

                return $"P{PointNumber}  X: {x:f2}, Y: {y:f2}, Z: {z:f2}";
            }
        }

        (后略)
```

8. 在 DockingWindow 文件夹中创建 AutoScrollBehavior.cs 文件。
   - 仅与 WPF 相关，细节省略。

9. 编辑 DockingWindow 文件夹下的 DockingWindowContentViewModelAddition.cs 文件。

   - 在 .xaml 中添加绑定的属性和命令。

   - 与机器人控制器是否连接，参考控制器连接 API 的 IsOnline 属性。IsOnline 在连接建立时为 true，断开时为 false，其他（如正在尝试连接等中间状态）为 null。连接状态变更时会触发 PropertyChanged 事件。
     - ObserveProperty(x => x.PropName).Subscribe(...) 是监控 API 对象中名为 PropName 的属性变更的常用方法。在本 Extension 中也有应用。

   - 电机状态参考控制器 API 的 IsMotorOn 属性。IsMotorOn 可能因控制器处于错误状态等原因为 null。电机状态变更时会触发 PropertyChanged 事件。

   - 要进行微动操作，使用 Jogger 对象。获取机器人管理器 API 对象并调用 CreateJoggerAsync 方法后，可获得 Jogger 对象。Jogger 对象有 IsValid 标志，仅在为 true 时功能可执行。调用 Jogger 对象方法时，请务必确认该标志。如因控制器断开等导致 Jogger 对象无效，请重新生成 Jogger 对象。
     - 与微动相关的参数（微动移动距离、速度等）在 RC+ 全局共享。在 RC+ 主体的“微动&示教”中进行的更改，原则上也适用于 SimpleJog。本次不实现 SimpleJog 的参数更改，如有需要请结合主程序的“微动&示教”使用。
       - 也可以考虑根据摇杆位置设定微动移动距离（如小幅操作对应短距离，大幅操作对应长距离等）的扩展。

- 使用游戏手柄时，可同时操作左右摇杆。当前 API 不支持指定多个方向进行微动操作。因此，在本 Extension 中，生成 Jogger 对象的同时启动定时器，并根据定时器周期获取的摇杆位置，依次对各轴方向进行微动，采用轮询算法。禁止同时执行多个微动任务，若前一周期启动的微动未结束，则新微动启动会报错。
  - 也可实现取消正在进行的微动并启动新的微动。

- 点文件可通过点 API 对象的 PointFileDescriptors 属性获得。本 Extension 旨在示教当前机器人的位置姿态，若控制器连接了多个机器人，原则上需示教到当前机器人关联或共用的点文件。
  - 本 Extension 中，优先示教到当前机器人的默认点文件，若无法获取，则以共用点文件之一为目标。若未找到目标点文件，则将 CanTeach.Value 设为 false，禁用示教按钮和命令。
  - 当前机器人的编号可通过机器人管理器 API 的 CurrentRobotNumber 属性获取。切换当前机器人时，该属性会触发 PropertyChanged 事件，请在此时重新选择目标点文件。

```
(前略)

namespace SimpleJog.DockingWindow
{
    (中略)

    /// <summary>
    /// Extension : Docking Window (Specific Part)
    /// </summary>
    internal partial class DockingWindowContentViewModel
    {
        /// <summary>
        /// The controller is online or not
        /// </summary>
        public ReactivePropertySlim<bool> IsOnline { get; } = new(false);

        /// <summary>
        /// Motors are powered or not
        /// </summary>
        public ReactivePropertySlim<bool> IsMotorOn { get; } = new(false);

        /// <summary>
        /// Motor state expression
        /// </summary>
        public ReactivePropertySlim<string> MotorState { get; } = new("Off");

        /// <summary>
        /// Toggle motor state command
        /// </summary>
        public AsyncReactiveCommand MotorToggleCommand { get; }

        /// <summary>
        /// TeachCommand feasibility
        /// </summary>
        public ReactivePropertySlim<bool> CanTeach { get; } = new(false);

        /// <summary>
        /// Teach command
        /// </summary>
        public AsyncReactiveCommand TeachCommand { get; }
```

```
        /// <summary>
        /// Teached points information for log
        /// </summary>
        public ReactiveCollection<LogItem> LogItems { get; } = [];

        /// <summary>
        /// API result expression
        /// </summary>
        public ReactivePropertySlim<string> APIResult { get; } = new();

        /// <summary>
        /// Auxiliary information for API result (Error message etc.)
        /// </summary>
        public ReactivePropertySlim<string> APIResultAux { get; } = new();

        /// <summary>
        /// Controller connection API object
        /// </summary>
        private IRCXControllerConnectionAPI? _connectionAPI;

        /// <summary>
        /// Controller API object
        /// </summary>
        private IRCXControllerAPI? _controllerAPI;

        /// <summary>
        /// Robot manager API object
        /// </summary>
        private IRCXRobotManagerAPI? _robotManagerAPI;

        /// <summary>
        /// Point API object
        /// </summary>
        private IRCXPointAPI? _pointAPI;

        /// <summary>
        /// Jogger object
        /// </summary>
        private IRCXRobotManagerAPI.IRCXJogger? _jogger;

        /// <summary>
        /// Polling timer
        /// </summary>
        private PeriodicTimer? _pollingTimer;

        /// <summary>
        /// Polling task
        /// </summary>
        private Task? _pollingTask;

        /// <summary>
        /// Next axis to jog
        /// </summary>
```

```
        private IRCXRobotManagerAPI.RCXJogCartesianAxis _targetAxis =
IRCXRobotManagerAPI.RCXJogCartesianAxis.Z;

        /// <summary>
        /// Polling interval
        /// </summary>
        private const long _pollingMSec = 10;

        /// <summary>
        /// Target point file for teaching
        /// </summary>
        private string? _targetPointFile;

        /// <summary>
        /// Toggles the motor state
        /// </summary>
        /// <returns>Task</returns>
        private async Task OnMotorToggleAsync()
        {
            if (_controllerAPI != null)
            {
                if (_controllerAPI.IsMotorOn == true)
                {
                    var result = await _controllerAPI.MotorOffAsync();
                    APIResult.Value = result.ToString();
                    APIResultAux.Value = string.Empty;
                }
                else if (_controllerAPI.IsMotorOn == false)
                {
                    var result = await _controllerAPI.MotorOnAsync();
                    APIResult.Value = result.ToString();
                    APIResultAux.Value = string.Empty;
                }
            }
        }

        /// <summary>
        /// Jog along specified axis
        /// </summary>
        /// <param name="axis">Axis</param>
        /// <param name="position">Stick position</param>
        /// <returns>Task</returns>
        private async Task Jog(
            IRCXRobotManagerAPI.RCXJogCartesianAxis axis,
            double position
        )
        {
            if (_jogger != null && _jogger.IsValid)
            {
                var oppositeDirection = (position > 0);
                var (result, message) = await
_jogger.StartCartesianJogAsync(axis, oppositeDirection);
                APIResult.Value = result.ToString() +
```

```
(string.IsNullOrEmpty(message) ? string.Empty : " *");
                APIResultAux.Value = message;
            }
        }

        /// <summary>
        /// Check the stick positions and jog
        /// </summary>
        /// <returns>Task</returns>
        private async Task CheckStickPosition()
        {
            switch (_targetAxis)
            {
                case IRCXRobotManagerAPI.RCXJogCartesianAxis.X:
                    if (Math.Abs(RightStickPosition.Value.X) >=
_positionThreshold)
                    {
                        await Jog(_targetAxis, RightStickPosition.Value.X);
                    }
                    _targetAxis = IRCXRobotManagerAPI.RCXJogCartesianAxis.Y;
                    break;

                case IRCXRobotManagerAPI.RCXJogCartesianAxis.Y:
                    if (Math.Abs(RightStickPosition.Value.Y) >=
_positionThreshold)
                    {
                        await Jog(_targetAxis, RightStickPosition.Value.Y);
                    }
                    _targetAxis = IRCXRobotManagerAPI.RCXJogCartesianAxis.Z;
                    break;

                case IRCXRobotManagerAPI.RCXJogCartesianAxis.Z:
                    if (Math.Abs(LeftStickPosition.Value.Y) >=
_positionThreshold)
                    {
                        await Jog(_targetAxis, LeftStickPosition.Value.Y);
                    }
                    _targetAxis = IRCXRobotManagerAPI.RCXJogCartesianAxis.X;
                    break;
            }
        }

        /// <summary>
        /// Set target point file
        /// </summary>
        /// <param name="robotNumber">Robot number</param>
        private void SetTargetPointFile(
            int? robotNumber
        )
        {
            _targetPointFile = null;

            if (_pointAPI != null)
```

```
            {
                var descriptors = _pointAPI.PointFileDescriptors;

                _targetPointFile = descriptors
                    .Where(x => x.RobotNumber == robotNumber && x.IsDefault)
                    .Select(x => x.FileName)
                    .FirstOrDefault();

                if (_targetPointFile == null)
                {
                    _targetPointFile = descriptors
                        .Where(x => x.RobotNumber == null)
                        .Select(x => x.FileName)
                        .FirstOrDefault();
                }
            }

            CanTeach.Value = (IsOnline.Value &&
!string.IsNullOrEmpty(_targetPointFile));
        }

        /// <summary>
        /// Teach point
        /// </summary>
        /// <returns>Task</returns>
        private Task OnTeachAsync()
        {
            if (_pointAPI != null && _targetPointFile != null)
            {
                var (result, points) = _pointAPI.GetPoints(_targetPointFile);
                if (result == RCXResult.Success && points != null)
                {
                    var pointNumbers = points.Select(x =>
(int)x["Number"].Value).ToHashSet();
                    var pointNumberRange = Enumerable.Range(
                        _pointAPI.PointNumberMin,
                        _pointAPI.PointNumberMax - _pointAPI.PointNumberMin +
1
                    );
                    foreach (var number in pointNumberRange)
                    {
                        if (!pointNumbers.Contains(number))
                        {
                            var stamp = DateTime.Now.ToString("yyyy-MM-dd
HH:mm:ss");

                            var teachResult = _pointAPI.TeachPoint(
                                _targetPointFile,
                                number,
                                description: $"SimpleJog: {stamp}",
                                shouldSave: true
                            );
                            APIResult.Value = teachResult.ToString();
```

```
                            APIResultAux.Value = string.Empty;

                            if (teachResult == RCXResult.Success)
                            {
                                LogItems.Add(new(number,
_robotManagerAPI?.WorldPosition));
                            }
                            break;
                        }
                    }
                }
            }

            return Task.CompletedTask;
        }

        /// <summary>
        /// Constructor
        /// </summary>
        public DockingWindowContentViewModel()
        {
            MotorToggleCommand = IsOnline
            .ToAsyncReactiveCommand()
            .WithSubscribe(OnMotorToggleAsync)
            .AddTo(_disposables);

            TeachCommand = CanTeach
            .ToAsyncReactiveCommand()
            .WithSubscribe(OnTeachAsync)
            .AddTo(_disposables);
        }

        /// <inheritdoc />
        public Task WindowCreated()
        {
            _connectionAPI = Main.GetAPI<IRCXControllerConnectionAPI>();

            _connectionAPI?.ObserveProperty(x =>
x.IsOnline).Subscribe((isOnline) =>
            {
                IsOnline.Value = (isOnline == true);

                CanTeach.Value = (IsOnline.Value &&
!string.IsNullOrWhiteSpace(_targetPointFile));
            })
            .AddTo(_disposables);

            _controllerAPI = Main.GetAPI<IRCXControllerAPI>();
            _robotManagerAPI = Main.GetAPI<IRCXRobotManagerAPI>();
            _pointAPI = Main.GetAPI<IRCXPointAPI>();

            _controllerAPI?.ObserveProperty(x => x.IsMotorOn).Subscribe(async
(isMotorOn) =>
```

```
            {
                IsMotorOn.Value = (isMotorOn == true);
                MotorState.Value = (isMotorOn == true) ? "On" : "Off";

                if (_robotManagerAPI != null)
                {
                    if (isMotorOn == true)
                    {
                        _jogger = await _robotManagerAPI.CreateJoggerAsync();
                        _pollingTimer = new
PeriodicTimer(TimeSpan.FromMilliseconds(_pollingMSec));
                        _pollingTask = Task.Factory.StartNew(async () =>
                        {
                            while (await _pollingTimer.WaitForNextTickAsync())
                            {
                                await CheckStickPosition();
                            }
                        });
                    }
                    else
                    {
                        if (_jogger != null)
                        {
                            await _jogger.DisposeAsync();
                            _jogger = null;
                        }
                        _pollingTask?.Dispose();
                        _pollingTimer?.Dispose();
                    }

                }
            })
            .AddTo(_disposables);

            _robotManagerAPI?.ObserveProperty(x =>
x.CurrentRobotNumber).Subscribe((robotNumber) =>
            {
                SetTargetPointFile(robotNumber);
            })
            .AddTo(_disposables);

            return Task.CompletedTask;
        }
    }
}
```

10. 编辑 MainMenuItem.cs 文件。

- 微动操作以与机器人控制器（虚拟或实机）连接已建立为前提。因此，若通过工具栏打开窗口时未连接，则需进行控制器连接。
  - 连接控制器需使用控制器连接 API。ConnectControllerAsync 方法与 RC+ 主体的控制器连接相同，自动连接时将尝试连接上次连接的控制器。非自动连接则显示“PC 与控制器连接”画面。

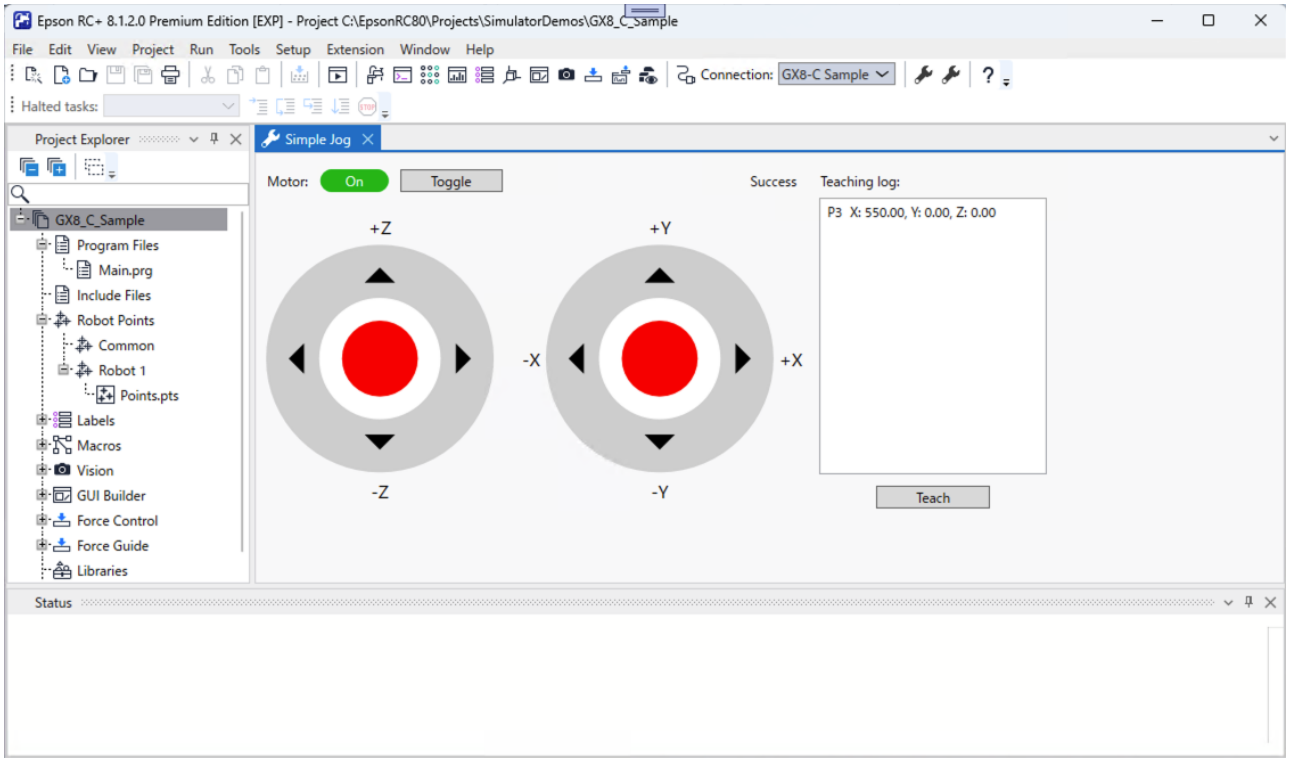
```
(前略)
        /// <inheritdoc />
        public async Task ExecuteMainMenuItemCommandAsync(
            string commandName,
            bool fromToolBar
        )
        {
            if (fromToolBar)
            {
                var controllerConnectionAPI =
Main.GetAPI<IRCXControllerConnectionAPI>();
                if (controllerConnectionAPI?.IsOnline == false)
                {
                    _ = await
controllerConnectionAPI.ConnectControllerAsync().ConfigureAwait(true);
                }
            }

            await DockingWindowContentViewModel.Show();
        }
(后略)
```

11. 编辑 Captions.xlsx 文件。

-

	A	B	C	D	E	F	G	H
1	ID	description	symbol	DEL	English	Japanese	German	French
2					English_en	Japanese_ja	German_de	French_fr
3								
4	0	Exttension Name	ExtensionName		Simple Jog	簡易ジョグ	Einfacher Jogginglauf	Jogging simple
5	1	Main Menu & ToolBar Item	MainMenu		Simple Jog	簡易ジョグ	Einfacher Jogginglauf	Jogging simple
6								
7	400	Docking Window Title	WindowTitle		Simple Jog	簡易ジョグ	Einfacher Jogginglauf	Jogging simple
8	401	"Toggle" Label	LabelToggle		Toggle	切換	Umschalten	Basculer
9	402	"Teaching Log:" Caption	CaptionLogHeader		Teaching log:	ティーチログ:	Unterrichtstagebuch:	Enregistrer d'enseignement:
10	403	"Teach" Label	LabelTeach		Teach	ティーチ	Teachen	Enseigner

12. 进行构建和调试。
    - 请打开 Extension 画面、机器人管理器的“微动&示教”，以及“模拟器”画面，测试机器人是否可移动。
      - 根据机器人的位置姿态，有时无法沿直角坐标进行微动操作。此时请先通过其他方式更改机器人位置姿态后再操作。



■ 中级篇

在中级篇中，将实现作为输入设备使用游戏手柄。(已在 Xbox Wireless Controller 上确认可用。)

1. 在 Visual Studio 的解决方案资源管理器中双击 SimpleJog 项目，进行如下更改。
   - 将 TargetFramework 修改为 net8.0-windows10.0.19041.0。
     - 由此可使用 Windows 运行时 (WinRT) 中的 Windows.Gaming.Input API，便于处理游戏手柄。

2. 编辑 install.json 文件。
   - 该文件用于指定如下内容。
     - 除构建输出文件夹外，Extension 使用的需另行复制的内容文件夹
     - 需与 Extension 主体一起显式加载的程序集
   - 此处内容如下。

```
{
    "Contents": [
    ],
    "Dependents": [
        "Microsoft.Windows.SDK.NET.dll",
        "WinRT.Runtime.dll"
    ]
}
```

3. 在 DockingWindow 文件夹中添加如下文件。
   - GamepadInfo.cs
     - 用于识别游戏手柄信息的 GamepadInfo 类文件。
       - Windows.Gaming.Input 的 Gamepad 类由于规范限制，无法直接获取易于理解的名称等信息。因此，在本 Extension 中，仅按发现顺序对游戏手柄进行识别。

```
(前略)

namespace SimpleJog.DockingWindow
{
    using Windows.Gaming.Input;

    /// <summary>
    /// Gamepad information
    /// </summary>
    public class GamepadInfo
    {
        /// <summary>
        /// Gamepad object
        /// </summary>
        public Gamepad Gamepad { get; }

        /// <summary>
        /// Gamepad number
        /// </summary>
        public int Number { get; }

        /// <summary>
        /// Gamepad name
        /// </summary>
        public string Name => $"Gamepad #{Number}";

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="gamepad">Gamepad object</param>
        /// <param name="number">Gamepad number</param>
        public GamepadInfo(
            Gamepad gamepad,
            int number
        )
        {
            Gamepad = gamepad;
            Number = number;
        }
    }
}
```

- IGamepadInputService.cs
  - 用于本 Extension 中游戏手柄输入接口 IGamepadInputService 的描述文件。

```
(前略)

namespace SimpleJog.DockingWindow
{
    using Reactive.Bindings;
    using Windows.Gaming.Input;
```

```
    /// <summary>
    /// Interface of gamepad input service
    /// </summary>
    public interface IGamepadInputService
    {
        /// <summary>
        /// Property for current reading
        /// </summary>
        public IReadOnlyReactiveProperty<GamepadReading> CurrentReading {
get; }

        /// <summary>
        /// Set target gamepad
        /// </summary>
        /// <param name="gamepad">Gamepad object</param>
        public void SetGamepad(
            Gamepad gamepad
        );

        /// <summary>
        /// Start service
        /// </summary>
        public void Start();

        /// <summary>
        /// Stop service
        /// </summary>
        public void Stop();
    }
}
```

- GamepadInputService.cs
  - 用于描述实现 IGamepadInputService 接口的 GamepadInputService 类的文件。
    - 通过使用定时器进行轮询，更新输入状态。但是，当鼠标按钮被按下时，会跳过更新。由于 DispatcherTimer 的 Tick 会在 UI 线程中调用，因此可以访问 System.Windows.Input 的 Mouse 实例。

```
(前略)

namespace SimpleJog.DockingWindow
{
    using Reactive.Bindings;
    using System.Windows.Threading;
    using Windows.Gaming.Input;

    /// <summary>
    /// Implementation of gamepad input service
    /// </summary>
    public class GamepadInputService : IGamepadInputService
    {
        /// <inheritdoc />
        public IReadOnlyReactiveProperty<GamepadReading> CurrentReading =>
```

```
_reading;

        /// <summary>
        /// The substance of CurrentReading
        /// </summary>
        private readonly ReactivePropertySlim<GamepadReading> _reading =
new(mode: ReactivePropertyMode.None);

        /// <summary>
        /// Target gamepad
        /// </summary>
        private Gamepad? _gamepad;

        /// <summary>
        /// Timer for polling
        /// </summary>
        private DispatcherTimer _timer;

        /// <summary>
        /// Polling interval
        /// </summary>
        private const int _pollingIntervalMSec = 16;

        /// <summary>
        /// Constructor
        /// </summary>
        public GamepadInputService()
        {
            _timer = new()
            {
                Interval = TimeSpan.FromMilliseconds(_pollingIntervalMSec),
            };

            _timer.Tick += (_, _) =>
            {
                if (_gamepad != null)
                {
                    if (Mouse.LeftButton == MouseButtonState.Pressed)
                    {
                        return;
                    }

                    _reading.Value = _gamepad.GetCurrentReading();
                }
            };
        }

        /// <inheritdoc />
        public void SetGamepad(
            Gamepad? gamepad
        )
        {
            _gamepad = gamepad;
```

```
        }

        /// <inheritdoc />
        public void Start()
        {
            _timer.Start();
        }

        /// <inheritdoc />
        public void Stop()
        {
            _timer.Stop();
        }
    }
}
```

- InputService.cs
  - 用于描述将游戏手柄输入转换为本Extension专用输入的服务类 InputService 的文件。
    - Stick 的鼠标处理也有类似的描述，但在此还会进行死区和平滑处理。游戏手柄的摇杆即使处于中立位置，值也可能不是零。在特定范围内将其视为零的处理就是死区处理。此外，即使摇杆动作很急，值也会经过调整，使其变化相对平缓，这就是平滑处理。

```
(前略)

namespace SimpleJog.DockingWindow
{
    (中略)

    /// <summary>
    /// Input service
    /// </summary>
    public class InputService : IDisposable
    {
        /// <summary>
        /// State of gamepad buttons
        /// </summary>
        public ReactivePropertySlim<GamepadButtons> Buttons { get; } =
new(GamepadButtons.None);

        /// <summary>
        /// Left stick position
        /// </summary>
        public ReactivePropertySlim<Vector> LeftStickPosition { get; } =
new();

        /// <summary>
        /// Right stick position
        /// </summary>
        public ReactivePropertySlim<Vector> RightStickPosition { get; } =
new();

        /// <summary>
```

```
        /// Stores the most recently calculated smoothed position for the
left stick.
        /// </summary>
        private Vector _leftSmoothedPosition;

        /// <summary>
        /// Stores the most recently calculated smoothed position for the
right stick.
        /// </summary>
        private Vector _rightSmoothedPosition;

        /// <summary>
        /// Dead zone definition
        /// </summary>
        private const double _deadZoneFactor = 0.05;

        /// <summary>
        /// Represents the smoothing factor used in calculations that
require exponential smoothing.
        /// </summary>
        /// <remarks>This constant determines the weight given to new data
points versus historical data
        /// in smoothing algorithms. A lower value results in smoother
output but slower response to changes.</remarks>
        private const double _smoothingFactor = 0.2;

        /// <summary>
        /// Disposables
        /// </summary>
        private readonly CompositeDisposable _disposables = [];

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="gamepadInputService">Gamepad input service</param>
        public InputService(
            IGamepadInputService gamepadInputService
        )
        {
            gamepadInputService.CurrentReading.Subscribe((reading) =>
            {
                Buttons.Value = reading.Buttons;

                _leftSmoothedPosition = AdjustPosition(
                    new Vector(reading.LeftThumbstickX,
reading.LeftThumbstickY),
                    _leftSmoothedPosition
                );
                _rightSmoothedPosition = AdjustPosition(
                    new Vector(reading.RightThumbstickX,
reading.RightThumbstickY),
                    _rightSmoothedPosition
                );
```

```
                LeftStickPosition.Value = _leftSmoothedPosition;
                RightStickPosition.Value = _rightSmoothedPosition;
            })
            .AddTo(_disposables);
        }

        /// <summary>
        /// Dead zone check and smoothing
        /// </summary>
        /// <param name="currentPosition">Current stick position</param>
        /// <param name="lastPosition">Last stick position</param>
        /// <returns>Adjusted stick position</returns>
        private Vector AdjustPosition(
            Vector currentPosition,
            Vector lastPosition
        )
        {
            var distance = Math.Sqrt(
                Math.Pow(currentPosition.X, 2.0)
                + Math.Pow(currentPosition.Y, 2.0)
            );

            if (distance < _deadZoneFactor)
            {
                return new Vector();
            }
            else
            {
                return new Vector(
                    lastPosition.X * (1.0 - _smoothingFactor) +
currentPosition.X * _smoothingFactor,
                    lastPosition.Y * (1.0 - _smoothingFactor) +
currentPosition.Y * _smoothingFactor
                );
            }
        }

        /// <inheritdoc />
        public void Dispose()
        {
            _disposables.Dispose();
        }
    }
}
```

4. 编辑 DockingWindow 文件夹下的 DockingWindowContent.xaml 文件。

- 在最上层的 Grid 中添加列，并放置用于选择游戏手柄的 ComboBox。
  - ItemsSource 绑定到 Gamepads(`ReactiveCollection<GamepadInfo>`)。
  - SelectedIndex 绑定到 SelectedGamepadIndex.Value。SelectedGamepadIndex 是 `ReactivePropertySlim<int>`。

- 将绑定到 Stick 的 LeftStickPosition 和 RightStickPosition 分别更改为 InputService.LeftStickPosition 和 InputService.RightStickPosition。

```
(前略)

        <StackPanel
            Grid.Row="2" Grid.Column="0"
            Orientation="Horizontal">

            <Label
                Content="Gamepads:"
                VerticalAlignment="Center" />
            <ComboBox
                ItemsSource="{Binding Gamepads}"
                SelectedIndex="{Binding SelectedGamepadIndex.Value}"
                DisplayMemberPath="Name"
                IsReadOnly="True"
                MinWidth="100"
                VerticalAlignment="Center"
                Margin="10,0,0,0" />

        </StackPanel>

(后略)
```

5. 编辑 DockingWindow 文件夹下的 DockingWindowContentViewModelAddition.cs 文件。

- CheckStickPosition 方法中的 LeftStickPosition 等要改写为 InputService.LeftStickPosition 等。
- 在窗口显示过程中，如果游戏手柄有插拔，会触发 GamepadAdded 和 GamepadRemoved 事件。如果在窗口显示前游戏手柄已连接，则不会触发这些事件，因此需要另外检查已连接的游戏手柄 (ScanGamepads 方法)。
- 游戏手柄按钮的按下，通过监视 InputService 实例的 Buttons 属性，调用相应的命令。

```
(前略)
        /// <summary>
        /// Input service object
        /// </summary>
        public InputService InputService { get; }

        (中略)

        /// <summary>
        /// List of connected game pads
        /// </summary>
        public ReactiveCollection<GamepadInfo> Gamepads { get; } = new();

        /// <summary>
        /// Selected game pad index
        /// </summary>
        public ReactivePropertySlim<int> SelectedGamepadIndex { get; } =
new(-1);

        (中略)
```

```
        /// <summary>
        /// Gamepad input service object
        /// </summary>
        private GamepadInputService _gamepadInputService = new();

        (中略)

        /// <summary>
        /// Scans for connected gamepads
        /// </summary>
        private void ScanGamepads()
        {
            SelectedGamepadIndex.Value = -1;

            Gamepads.Clear();

            const int _waitMSec = 100;
            const int _maxRetryCount = 30;

            for (var retryCount = 0; retryCount < _maxRetryCount;
retryCount++)
            {
                if (Gamepad.Gamepads.Count <= 0)
                {
                    Thread.Sleep(_waitMSec);
                }
                else
                {
                    foreach (var (gamepad, index) in
Gamepad.Gamepads.Select((x, index) => (x, index)))
                    {
                        Gamepads.Add(new GamepadInfo(gamepad, 1 + index));
                    }
                    SelectedGamepadIndex.Value = 0;
                    break;
                }
            }
        }

        /// <summary>
        /// Constructor
        /// </summary>
        public DockingWindowContentViewModel()
        {
            InputService = new(_gamepadInputService);

            (中略)

            InputService.Buttons.Subscribe((buttons) =>
            {
                if ((buttons & GamepadButtons.LeftShoulder) != 0)
                {
                    MotorToggleCommand.Execute();
```

```
                }

                if ((buttons & GamepadButtons.A) != 0)
                {
                    TeachCommand.Execute();
                }
            })
            .AddTo(_disposables);

            SelectedGamepadIndex.Subscribe((index) =>
            {
                if (index >= 0)
                {
                    _gamepadInputService.SetGamepad(Gamepads[index].Gamepad);
                }
            })
            .AddTo(_disposables);

            Gamepad.GamepadAdded += (_, gamepad) =>
            {
                Gamepads.AddOnScheduler(new GamepadInfo(gamepad,
Gamepads.Count));
            };

            Gamepad.GamepadRemoved += (_, gamepad) =>
            {
                var target = Gamepads.FirstOrDefault(x =>
ReferenceEquals(x.Gamepad, gamepad));
                if (target != null)
                {
                    Gamepads.RemoveOnScheduler(target);
                }
            };

            ScanGamepads();

            _gamepadInputService.Start();
        }

        (后略)
```

6. 请编辑 DockingWindow 文件夹中的 DockingWindowContentViewMode.cs 文件。

   - 关闭窗口时，也要停止 GamepadInputService。

```
(前略)

/// <inheritdoc />
public Task<bool> CloseAsync()
{
    _gamepadInputService.Stop();

    return Task.FromResult(true);
```

```
}

(后略)
```

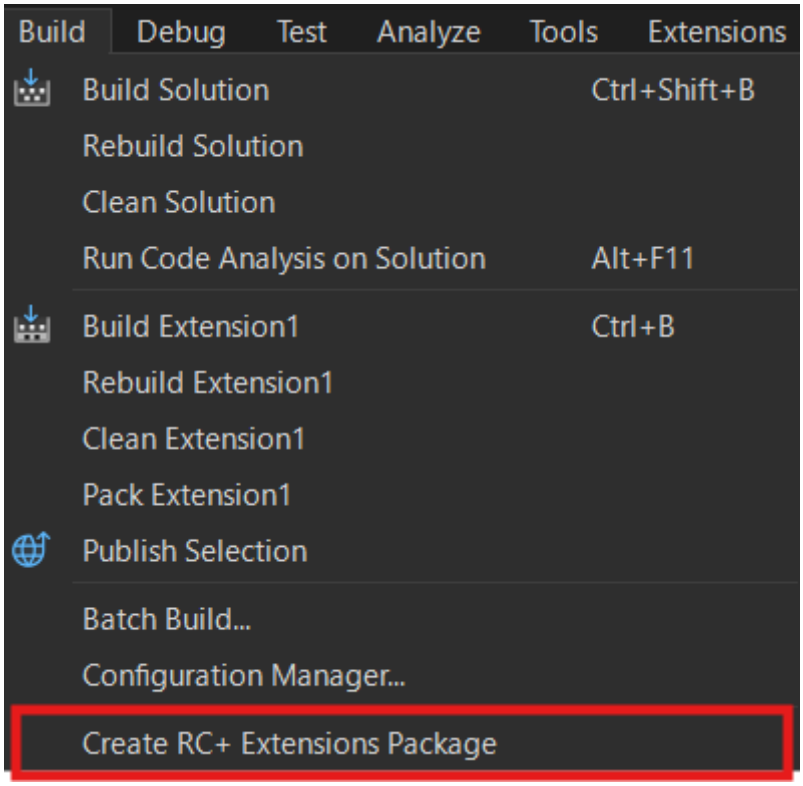
7. 进行构建和调试。
   - 与初级篇一样，打开各窗口，确认能否通过游戏手柄进行操作。
   - 本 Extension 在游戏手柄支持方面存在以下限制。
     - 如果 Extension 的窗口没有焦点，则无法捕获游戏手柄输入。
     - 特别是在通过 API 调用打开确认对话框等情况下，无法使用游戏手柄点击对话框的按钮，因此必须中断游戏手柄操作，改用 PC 的鼠标或键盘。
       - 在本 Extension 中，开启马达时的确认对话框属于此类情况。在可以判断为可以省略确认对话框显示的场合（请谨慎考虑），可以通过执行 SPEL+ 命令 "Motor On" 替代马达开启的 API，从而跳过确认。在最终代码中已实现此功能，有兴趣的用户请自行查阅。

5.5 Extension 的分发

为了在其他环境中使用开发好的 Extension，需要进行打包。在 RC+ Extensions 中，可以通过 Visual Studio 的构建菜单轻松创建包。

按照以下步骤进行打包。

1. 在 Visual Studio 中打开项目，选择菜单-[构建]-[生成解决方案]。
   - 构建成功完成后，项目根目录下的`bin`文件夹中会生成 DLL 等构建产物。
2. 选择菜单-[构建]-[Create RC+ Extensions Package]。
   - 如下图所示，Visual Studio 的构建菜单中已添加专用项。



   - 执行后，会在项目根目录下的`bin\Release`或`bin\Debug`文件夹中生成`.rcxpkg`文件。
   - `.rcxpkg`文件是可在 Epson RC+ 扩展功能管理器中安装的格式。

创建的`.rcxpkg`可按以下步骤安装到 Epson RC+。

1. 将`.rcxpkg`文件放置在 PC 上的任意文件夹或组织内的共享文件夹。

2. 启动 Epson RC+，从[扩展]菜单中选择[扩展功能管理器]。
3. 通过界面右上角的本地源设置按钮，将刚才放置的文件夹路径作为本地源添加。这样，扩展功能管理器的引用标签页中会显示已创建的 Extension，可以进行下载和安装。
   - 详情请参阅 [Extension的使用]。

5.6 API 说明

在 RC+ Extension 项目中，可以通过使用 Extensions API 调用 Epson RC+ 8.0 的功能。
主要 API 如下所示。如需更详细的信息，请参考我们在 GitHub 上提供的 API 参考文档。

https://github.com/Epson-Robots/rcplus-extensions

API	概述
IRCXProjectAPI	与 SPEL+ 项目相关的一组API。请参阅 [项目]。
IRCXPointAPI	用于操作点数据的 API 集合。请参阅 [点]。
IRCXProgramEditorAPI	与 Epson RC+ 8.0 程序编辑器操作相关的一组 API。请参阅 [程序编辑器]。
IRCXControllerConnectionAPI	与控制器连接相关的 API 集合。请参阅 [控制器连接]。
IRCXControllerAPI	用于获取控制器信息和对控制器进行设置的 API 集合。请参阅 [控制器设置]。
IRCXIOAPI	与 I/O 操作相关的 API 集合。请参阅 [I/O]。
IRCXRobotManagerAPI	与机器人操作相关的 API 集合。请参阅 [机器人操作]。
IRCXProgramExecutionAPI	与 Epson RC+ 8.0 程序执行操作相关的一组 API。请参阅 [程序执行]。
IRCXPreferencesAPI	与 Epson RC+ 8.0 开发环境设置相关的一组 API。请参阅 [开发环境设置]。

要使用 Extensions API，请按如下方式编写代码以获取 API 实例。

```
var extensionAPI = Main.GetAPI<IRCXProjectAPI>();
```

5.6.1 项目

与项目操作相关的 API 集合。

```
// Retrieves the API instance.
IRCXProjectAPI api = Main.GetAPI<IRCXProjectAPI>()!;

// Opens the specified project.
var ret = await api.OpenAsync("C:\\EpsonRC80\\projects\\SimulatorDemos\\C4_Sample",
false);

// Opens the specified project file.
ret = await api.OpenProjectFileAsync("Main.prg");

// Builds the project.
ret = await api.BuildAsync();

// Closes the specified project file.
ret = await api.CloseProjectFileAsync("Main.prg");

// Closes the project.
ret = await api.CloseAsync();
```

5.6.2 点

用于操作点数据的 API 集合。

```
// Retrieves the API instance.
IRCXPointAPI api = Main.GetAPI<IRCXPointAPI>()!;

// Retrieves the list of point file names included in the project.
var pointFileNames = api.PointFileDescriptors.Select(x => x.FileName);
if (pointFileNames.Count() == 0) return;

var pointFileName = pointFileNames.ElementAt(0);

// Retrieves the list of point data defined in the specified point file.
var (ret, points) = api.GetPoints(pointFileName);

// Adds a point.
Dictionary<string, IRCXPointAPI.RCXPointElement> point = new() {
    ["Number"] = new(typeof(int), 10),
    ["Label"] = new(typeof(string), "MyPointLabel"),
    ["X"] = new(typeof(double), 100.0),
    ["Hand"] = new(typeof(IRCXPointAPI.RCXPointElement.Hand),
IRCXPointAPI.RCXPointElement.Hand.Lefty)
};
var addResult = api.AddPoint(pointFileName, point);

// Deletes a point.
var deleteResult = api.DeletePoint(pointFileName, 10);
```

5.6.3 程序编辑器

关于程序编辑器操作的 API 集合。

```
// Retrieves the API instance.
IRCXProgramEditorAPI api = Main.GetAPI<IRCXProgramEditorAPI>()!;

// List of program editors currently opened in Epson RC+ 8.0.
var programEditors = api.ProgramEditors;
if (programEditors == null || programEditors.Count() <= 0) return;

var editor = programEditors.ElementAt(0)!;

// Retrieves the content of the specified line number.
var (ret, content) = await editor.GetLineContentAsync(1);

// Sets a breakpoint at the specified line number.
ret = await editor.SetBreakpointAsync(1);

// Clears the breakpoint set at the specified line number.
ret = await editor.ClearBreakpointAsync(1);

// Clears all breakpoints set in the program editor.
ret = await editor.ClearAllBreakpointsAsync();
```

5.6.4 控制器连接

与控制器连接相关的 API 集合。

```
// Retrieves the API instance.
IRCXControllerConnectionAPI? api = Main.GetAPI<IRCXControllerConnectionAPI>();

// Retrieves the connection information of the controller last connected.
IRCXControllerConnectionAPI.IRCXConnection? lastConnection =
api?.GetLastConnection();
if (lastConnection == null) return;

// The controller number last connected.
int number = lastConnection.Number;

// Connects to the controller.
// Specify the controller number to connect to as the first argument.
bool connectResult = await
api?.ConnectionControllerAsync(lastConnection).ConfigureAwait(false);

if (connectResult) {
    // Process executed when the connection succeeds.
} else {
    // Process executed when the connection fails.
}

bool? isConnected = api?.IsOnline;  // The current connection state of the
controller.
if (isConnected == true) {
    // Disconnects from the controller.
    await api?.DisconnectControllerAsync().ConfigureAwait(false);
}
```

5.6.5 控制器设置

用于获取控制器信息和对控制器进行设置的 API 集合。
需要连接至控制器。关于控制器连接，请参阅［控制器连接］。

```
// Retrieves the API instance.
IRCXControllerAPI api = Main.GetAPI<IRCXControllerAPI>()!;

// Retrieves the list of controller setting categories.
var categoryNames = api.GetControllerSettingsCategoryNames();
var categoryName = categoryNames.ElementAt(1); // "Configuration" category.

// Retrieves the controller settings for the specified category.
var (result, settings) = api.GetControllerSettings(null, categoryName);

// Procedure for applying new values to the controller.
// Initiates a controller settings update session.
var (ret, id) = await api.StartSetControllerSettingsAsync();

// Updates the controller name.
settings["Name"].Value = "MyController";

// Submits the updated values for the specified category.
var result = await api.SetControllerSettingsAsync(id, categoryName, settings);

// Commits the changes and applies them to the controller.
var setResult = await api.CommitSetControllerSettingsAsync(id);
```

5.6.6 I/O

与 I/O 操作相关的 API 集合。
需要连接至控制器。关于控制器连接，请参阅［控制器连接］。

```
// Retrieves the API instance.
IRCXIOAPI api = Main.GetAPI<IRCXIOAPI>()!;

var ioLabels = api.IOLabels;  // List of I/O label objects.

// Adds a new I/O label.
_ = api.AddIOLabel("MyLabel", IRCXIOAPI.RCXIOKind.Input, typeof(bool), 0,
"MyComment");
if (ioLabels?.FirstOrDefault(i => i.Label == "MyLabel") is IRCXIOAPI.IRCXIOLabel
ioLabel) {
    // Updates the existing I/O label.
    _ = api.AddIOLabel("MyLabel", IRCXIOAPI.RCXIOKind.Input, typeof(bool), 0,
"MyComment2");
}

// Creates an object that monitors the I/O state.
var ret = api.CreateWatcher<bool>(IRCXIOAPI.RCXIOKind.Input, 0, (watcher, oldData,
newData) => {
    // Called when the I/O state changes.
});

// To stop monitoring the I/O state, dispose of the watcher object.
if (ret != null) {
    IRCXIOAPI.IRCXIOWatcher? watcher = ret.Value.Item2;
    _watcher?.Dispose();
}
```

5.6.7 机器人操作

与机器人操作相关的 API 集合。

```
// Retrieves the API instance.
IRCXRobotManagerAPI api = Main.GetAPI<IRCXRobotManagerAPI>()!;

// Retrieves the current robot number.
var currentRobotNumber = api.CurrentRobotNumber;

// Sets the current robot.
var ret = await api.SetCurrentRobotAsync(1);

// Retrieves the current joint position of the current robot.
var currentJointPosition = api.JointPosition;

// Monitors changes in the current robot’s joint position and performs processing
when it changes.
api.ObserveProperty(x => x.JointPosition).Subscribe(position => {
    // Do something
});

// Retrieves the jogger object.
var jogger = await api.CreateJoggerAsync();

// Executes a jog operation.
// When using an actual robot, ensure the emergency stop button can be pressed
```

```
before executing.
var _ = jogger.StartJointJogAsync(IRCXRobotManagerAPI.RCXJogJointAxis.J1);
```

5.6.8 程序执行

关于程序执行操作的 API 集合。

```
// Retrieves the API instance.
IRCXProgramExecutionAPI api = Main.GetAPI<IRCXProgramExecutionAPI>()!;

// Retrieves the list of user-defined functions registered in Epson RC+ 8.0.
var userFunctions = api.UserFunctions;
if (userFunctions == null || userFunctions.Count() <= 0) return;

// Executes the specified user function.
await api.StartFunctionAsync(userFunctions.ElementAt(0), false, false, Id);

// Executes a SPEL command.
await api.ExecuteSpelCommandAsync("Motor ON");
```

5.6.9 开发环境设置

与开发环境设置相关的 API 集合。

```
// Retrieves the API instance.
IRCXPreferencesAPI api = Main.GetAPI<IRCXPreferencesAPI>();

// Retrieves the list of preference category names.
var preferenceCategories = api.GetPreferencesCategoryNames();

// Retrieves the preference values for the specified category.
var (ret, preferences) = api.GetPreferences(preferenceCategories.ElementAt(0));
if (preferences == null) return;

// Modifies the development environment settings.
preferences["IsAutoSave"].Value = false;
await api.SetPreferencesAsync(preferenceCategories.ElementAt(0), preferences);
```

5.6.10 窗口

用于显示停靠窗口和消息框等的 API 集合。

```
// Retrives the API instance.
IRCXWindowAPI? api = Main.GetAPI<IRCXWindowAPI>();

// Show message box.
var response = api?.ShowMessageBox(
    new RCXCaption(Main.CommonId, Caption.ExtensionName),
    new RCXCaption("Are you OK?"),
    IRCXWindowAPI.ButtonType.Yes_No,
    IRCXWindowAPI.IconType.Question
);
if (response == IRCXWindowAPI.ResponseType.OK)
{
    // Process when the response is OK.
}
```

还提供用于托管 WPF 的 WebView2 控件的停靠窗口 API。

- 可以处理以下 URI。
  - 外部（互联网、内联网）网站
  - 部署在运行 RC+ 的 PC 本地的静态网站
    - 也可以启动专用的 HTTP 服务器，以实现 JavaScript 模块的动态导入。
  - 在运行 RC+ 的 PC 本地运行的 Web 应用程序
    - 可以指定应用程序的启动命令（如 node）以及参数。
      - 作为参数，如果传递 ${port}、${lang}，则分别会替换为端口号（自动分配）、显示语言名称。
- 可以使用 Web 技术（HTML、CSS、JavaScript 等）实现功能。
  - 窗口关闭、内容保存时执行脚本
  - 内容编辑
    - 获取选中的字符串并传送到剪贴板
    - 发送与编辑菜单对应的消息（"RCX.Clear", "RCX.Paste", "RCX.SelectAll", "RCX.Undo", "RCX.Redo"）
  - 通过F10键发送与帮助显示对应的消息（"RCX.ShowHelp"）
- 创建主机对象后，可以与 C# 端进行交互。
  - 也可以使用实现了以下函数的内置主机对象 chrome.webview.hostObjects.BuiltinBridge。
    - string GetCaption(int number): 获取指定编号的标题字符串。
    - string ReadConfiguration(bool global): 以 Base64 编码字符串形式读取 Extension 的设置数据。
    - string WriteConfiguration(bool global): 以 Base64 编码字符串形式写入 Extension 的设置数据。
      - 当 global 标志为 true 时，设置数据在 Windows 登录用户间共享；为 false 时，则按登录用户分别存储。

```
// Retrives the API instance.
IRCXWindowAPI? api = Main.GetAPI<IRCXWindowAPI>();

if (api != null)
{
    // Show "Epson Global Portal" site in a docking window
    var webVewInfo = new IRCXWindowAPI.WebViewInfo(
        (_, _, _) => new Uri("https://epson.com/"),
        Id,
        $"{Id}.External",
        new RCXCaption(CommonId, Caption.WindowTitle_External),
        Main.CommonIcon
    );

    await api.ShowDockingWebViewWindowAsync(webViewInfo).ConfigureAwait(true);
}
```

5.7 扩展点说明

在 Extension（RC+自定义）中，提供了用于将菜单项添加、项目文件管理、显示停靠窗口等功能集成到 Epson RC+ 的机制（扩展点）。
这些功能通过将 Extensions API 提供的扩展点接口，作为 .NET 内置扩展框架 Managed Extensibility Framework（MEF）的 Export 实现来运行。
在创建 Extension 项目时，可以将所需的扩展点作为 Initial Feature 进行选择，项目创建后也可以作为新项进行添加。
本节将说明各扩展点的实现方法。

5.7.1 主菜单项及工具栏按钮

Extension 可以在主菜单中添加用于 Extension 的菜单项。菜单项也可以通过子菜单进行分层。

此外，对于每个菜单项，也可以添加对应的工具栏按钮。(仅添加工具栏按钮的操作不被支持)

选择菜单项或点击工具栏按钮时，会调用 Extension 的命令。被调用的命令在菜单项和工具栏按钮之间没有区别。不过，可以区分命令是否是通过工具栏按钮调用的。

要使用此扩展点，需要创建并导出实现了接口 IRCXMainMenuItemProvider 的类。

```
[Export(typeof(IRCXMainMenuItemProvider))]
public class MainMenuItem : IRCXMainMenuItemProvider
{
    /// <inheritdoc />
    public string Id => Main.CommonId;

    /// <inheritdoc />
    public string MenuItemId => "MyExtension.MainMenuItem";

    /// <inheritdoc />
    public IRCXMainMenuItemProvider.MenuItem MainMenuRootItem
    {
        get
        {
            return new IRCXMainMenuItemProvider.MenuItem
            {
                Caption = new RCXCaption(Main.CommonId, Caption.MainMenu),
                Icon = Main.CommonIcon,
                CommandName = "Main",
                ToolTip = new RCXCaption(Main.CommonId, Caption.MainMenu),
            };
        }
    }

    /// <inheritdoc />
    public IRCXMainMenuItemProvider.TopLevelMenu TopLevel =>
IRCXMainMenuItemProvider.TopLevelMenu.Default;

    /// <inheritdoc />
    public Task ExecuteMainMenuItemCommandAsync(
        string commandName,
        bool fromToolBar
    )
    {
        // (Code here)
        return Task.CompletedTask;
    }
}
```

如果包含多个菜单项，请注意 MenuItemId 不要与其他项重复。

以下是包含子菜单的示例。

```
    public IRCXMainMenuItemProvider.MenuItem MainMenuRootItem
    {
        get
        {
            return new IRCXMainMenuItemProvider.MenuItem
            {
                Caption = new RCXCaption(Main.CommonId, Caption.MainMenu),
                Icon = Icon,
                CommandName = "Main",
```

```
                Children =
                [
                    new()
                    {
                        Caption = new RCXCaption(Main.CommonId,
Caption.MainMenu_Sub1),
                        Icon = Icon,
                        CommandName = "Sub1",
                        ToolTip = new RCXCaption(Main.CommonId,
Caption.ToolTip_Sub1),
                    },
                    new()
                    {
                        Caption = new RCXCaption(Main.CommonId,
Caption.MainMenu_Sub2),
                        Icon = Icon,
                        CommandName = "Sub2",
                        ToolTip = null,
                    },
                ]
            };
        }
    }
```

将 ToolTip 设置为 null 时，该菜单项对应的工具栏按钮不会显示。
当前规范下，没有提供动态更改菜单项显示字符串等内容的方法。

5.7.2 停靠窗口

Extension 可以通过提供作为内容的用户控件及其视图模型，来显示停靠窗口。

对话框窗口（模态或非模态）可以通过WPF的标准功能进行显示，因此 API 不提供相关支持。

用于用户控件的视图模型类必须实现接口 IRCXUserControlViewModel。

```
internal partial class DockingWindowContentViewModel : IRCUserControlViewModel
{
    /// <inheritdoc />
    public string Id => Main.CommonId;

    /// <inheritdoc />
    public string ViewModelId => $"MyExtension.DockingWindow";

    /// <inheritdoc />
    public bool KeepOpenWhenProjectClosing => false;

    /// <summary>
    /// Captions
    /// </summary>
    public static IRCXCaptionGetter Captions { get; } = Main.Captions!;

    /// <inheritdoc />
    public RCXCaption WindowCaption { get; set; } = new(Main.CommonId,
Caption.WindowTitle);

    /// <inheritdoc />
    public ImageSource? WindowIcon { get; set; } = Main.CommonIcon;

    /// <inheritdoc />
    public Task<bool> CloseAsync()
    {
        return Task.FromResult(true);
    }
```

```
    /// <inheritdoc />
    public Task<bool> SaveAsync()
    {
        return Task.FromResult(true);
    }

    /// <inheritdoc />
    public void Reload()
    {
    }

    /// <inheritdoc />
    public void Copy()
    {
    }

    /// <inheritdoc />
    public void Cut()
    {
    }

    /// <inheritdoc />
    public void Paste()
    {
    }

    /// <inheritdoc />
    public void SelectAll()
    {
    }

    /// <inheritdoc />
    public void Undo()
    {
    }

    /// <inheritdoc />
    public void Redo()
    {
    }

    /// <inheritdoc />
    public void ShowHelp()
    {
    }

    /// <summary>
    /// Show docking window
    /// </summary>
    /// <returns>Task</returns>
    public static async Task Show()
    {
        DockingWindowContent control = new();

        if (control.DataContext is DockingWindowContentViewModel controlViewModel)
        {
            await Main.GetAPI<IRCXWindowAPI>
().ShowDockingWindowAsync(controlViewModel, control);
        }
    }
}
```

编辑命令（Copy、Cut、Paste、SelectAll、Undo、Redo）通过主窗口的编辑菜单中对应项目的选择来调用。

要获取或设置各菜单项的启用 / 禁用状态时，请使用窗口 API 的 GetContentState 或 SetContentState 方法。

以下是在上述 DockingWindowContentViewModel 中启用 Undo 的示例。

```
var api = Main.GetAPI<IRCXWindowAPI>();

var state = api.GetContentState(this);
api.SetContentState(this, state | ContentState.CanUndo);
```

窗口显示期间按下 F10 键时，会调用 ShowHelp 方法。

以下是在 ShowHelp 中显示 Extension 所含 PDF 手册的示例。(在本示例中，假定 PDF 手册文件位于项目的 Resources 文件夹中。要将文件包含到 Extension 中，请在 Visual Studio 中将文件属性的“生成操作”设置为“内容”，并将“复制到输出目录”设置为“始终复制”或“如果较新则复制”。

```
public void ShowHelp()
{
    try
    {
        var helpFilePath = Path.Combine(
            Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location)!,
            "Resources",
            "Manual.pdf"
        );

        ProcessStartInfo processStartInfo = new()
        {
            FileName = helpFilePath,
            UseShellExecute = true,
        };
        Process.Start(processStartInfo);
    }
    catch (Exception)
    {
        // Error handling
    }
}
```

5.7.3 项目文件

可以向 RC+ 的项目中添加由 Extension 管理的专有文件。

要使用此扩展点，需要创建并导出实现接口 IRCXProjectFileProvider 的类。

```
[Export(typeof(IRCXProjectFileProvider))]
public partial class ProjectFileEditorProjectFile : IRCXProjectFileProvider
{
    /// <inheritdoc />
    public string Id => Main.CommonId;

    /// <inheritdoc />
    public string FileTypeName => "MyExtensionFiles";

    /// <inheritdoc />
    public string Extension => ".ext";

    /// <inheritdoc />
    public bool UseDefaultProjectExplorerItem => true;

    /// <inheritdoc />
    public RCXCaption ProjectExplorerRootItemCaption => new(Main.CommonId,
Caption.FileCategory);
```

```
    /// <inheritdoc />
    public ImageSource? ProjectExplorerRootItemIconData => Main.CommonIcon;

    /// <inheritdoc />
    public ImageSource? FileIcon => Main.CommonIcon;

    /// <inheritdoc />
    public RCXCaption FileTypeNameCaption => new(Main.CommonId,
Caption.FileTypeName);

    /// <inheritdoc />
    public async Task OpenAsync(
        string fileName
    )
    {
        // Open project file editor window
        ProjectFileEditor editorControl = new();

        if (editorControl.DataContext is ProjectFileEditorViewModel
editorViewModel)
        {
            editorViewModel.FileName = fileName;
            editorViewModel.LoadContent();
            await Main.GetAPI<IRCXWindowAPI>
().ShowDockingWindowAsync(editorViewModel, editorControl);
        }
    }

    /// <inheritdoc />
    public void WriteInitialContent(
        FileStream fileStream
    )
    {
        try
        {
            // Write initial content of the file (optional)
            using var writer = new StreamWriter(fileStream, Encoding.UTF8);
            writer.WriteLine("ProjectFile Initial Content");
        }
        catch (Exception)
        {
            // Handle Error
        }
    }
}
```

在 Extension 中，需要编写以下处理流程。

1. 文件的“打开”处理 (OpenAsync)
   - 通常，提供专有项目文件时，会同时提供用于编辑该文件的编辑器。使用停靠窗口的编辑器，其实现方法与前述停靠窗口的实现方式相同。典型的实现方式如上所述，即创建编辑器控件，将从文件读取的内容传递给其视图模型，并显示编辑器窗口。
2. 在文件创建时写入初始内容的处理 (WriteInitialContent)
   - 调用本方法时，文件会以空状态创建，并会传递与该文件关联的 FileStream。如果初始内容为空，则本方法无需执行特殊处理。

将 UseDefaultProjectExplorerItem 标志设置为 true 时，可以在项目资源管理器中添加 Extension 的树形项目。(如果不使用此功能，则必须结合下一节所述的扩展点使用)

此情况下的树形项目由以下部分构成。

- 表示 Extension 文件类型的父项目
  - 作为子菜单，具有由“新建文件...”和“已有文件...”两个项目构成的“添加新项目”上下文菜单。

- 已添加文件的文件名作为子项目显示。
  - 包含“打开”、“从项目中排除”、“删除”三个项目的上下文菜单。

5.7.4 项目资源管理器的树状项目

Extension 可以在已打开的 RC+ 项目的项目资源管理器中添加专有的树状项目。
如果树状项目与专有文件关联，请使用前述的“项目文件”扩展点。

树状项目可以添加多个，并且每个项目都可以进行分层。

双击树状项目时，

- 若为父项目，则会展开或收起子项目。
- 若为末端子项目，则会调用 Extension 的命令。

树状项目可以拥有上下文菜单。选择上下文菜单项目时，会调用 Extension 的命令。

要使用此扩展点，需要创建并导出实现接口 IRCXProjectExplorerItemProvide 的类。

```
[Export(typeof(IRCXProjectExplorerItemProvider))]
public partial class ProjectExplorerItem : IRCXProjectExplorerItemProvider
{
    /// <inheritdoc />
    public string Id => Main.CommonId;

    /// <inheritdoc />
    public IRCXProjectExplorerItemProvider.Item ProjectExplorerRootItem
    {
        get
        {
            return new()
            {
                Caption = new(Main.CommonId, Caption.PECategory),
                Icon = Main.CommonIcon,
                Children =
                [
                    new()
                    {
                        Caption = new(Main.CommonId, Caption.PEItem),
                        Icon = Main.CommonIcon,
                        CommandName = "ItemCommand",
                        CommandParameter = "ItemCommandParameter",
                        ContextMenuItems =
                        [
                            new()
                            {
                                Caption = new(Main.CommonId, Caption.PEItemMenu),
                                CommandName = "ContextMenuCommand",
                                CommandParameter = "ContextMenuCommandParameter",
                            }
                        ]
                    }
                ]
            };
        }
    }

    /// <inheritdoc />
    public Task ExecuteProjectExplorerItemCommandAsync(
        string commandName,
        object? commandParameter = null
    )
```

```
    {
        // (Code here)
        return Task.CompletedTask;
    }
}
```

5.7.5 外部函数

SPEL+ 程序可以使用 Declare 语句来执行在 DLL 中定义的外部函数。该机制自早期版本的 RC+ 就已存在，并且 DLL 必须为 32 位本地格式，这是一个限制条件。

在 Extension 的开发工具包中，提供了桥接 DLL，使 SPEL+ 程序能够调用作为 64 位程序集的 Extension 内部函数。这就是 Extension 的“外部函数”扩展点。

此“外部函数”的调用格式如下。

- 在 SPEL+ 程序中插入如下的 Declare 语句。

```
Declare CallExternal, "C:\EpsonRC80\ExternalFunctionBridge.dll", "CallExternal",
(commandLine$ As String, ByRef output$ As String) As Int32
```

  - commandLine$ 是一个字符串，用于存储以空格分隔的函数名和参数的命令行。
  - output$ 是一个字符串变量，用于存储函数的输出。
  - 返回值是函数执行的结果代码。
    - 0 表示正常结束（成功）。
    - 非 0 表示错误。
  - （示例）

```
Int32 ret
String output$
ret = CallExternal("CubeRoot 10.0", ByRef output$)
Print output$
```

    - 将在 Run 窗口输出 2.154434690031884。

要使用此扩展点，需要实现并导出委托 RCXExternalFunction，并作为元数据导出函数名。

```
[Export(typeof(RCXExternalFunction))]
[ExportMetadata("Name", "CubeRoot")]
public RCXExternalFunction CubeRoot = (command, parameters) =>
{
    if (parameters.Count == 0 || !double.TryParse(parameters[0], out var input))
    {
        return ValueTuple.Create(RCXResult.BadArgument, string.Empty);
    }

    double output = Math.Cbrt(input);

    return ValueTuple.Create(RCXResult.Success, output.ToString());
};
```

通过此扩展点，可以从 SPEL+ 程序调用 Extension 中可实现的各种函数。

此外，以下作为内置“外部函数”，在支持 Extension 的 RC+ 版本中始终可用。

1. 外部程序执行（等待结束，返回输出的第一行）

```
ret = CallExternal("Execute program [arg(s)]", output$)
```

2. 外部程序启动（不等待结束）

```
ret = CallExternal("ExecuteNoWait program [arg(s)]", output$)
```

3. 获取与 CallExternal 结果代码对应的字符串

```
ret = CallExternal("ErrorStr Str$(retCode)" output$)
```

   - 例

```
Int32 ret
String output$
ret = CallExternal("ErrorStr 0", ByRef output$)
Print output$
```

     - 将在 Run 窗口输出 Success。

(高级用户向)

此外，桥接 DLL 中还包含用于控制外部函数行为的函数。(需要与 CallExternal 相同的 Declare 语句)

1. GetTimeout(ByRef timeout As Int32) As Int32
   - 获取 CallExternal 的超时时间。单位为毫秒，初始值为 30,000。
2. SetTimeout(timeout As Int32) As Int32
   - 设置 CallExternal 的超时时间。单位为毫秒。

6. PC视觉 自定义视觉对象的开发

6.1 概述

自定义视觉对象是可作为扩展功能安装到 RC+80 Vision Guide 系统中的自定义视觉对象。
在 Epson RC+ 8.0 Ver8.1.3.0 及之后的版本中，PC 视觉功能可以使用自定义视觉对象。

通过使用自定义视觉对象，可以将专属的图像处理功能集成到 RC+80 Vision Guide 系统中。自定义视觉对象可以实现类似 ImageOp 的图像转换处理，以及类似 Geometric 的位置检测处理。可以利用 OpenCV 等外部库，集成现有对象所不具备的图像处理功能。

本章将说明从自定义视觉对象的实现到包制作的开发方法。

※关于开发后的自定义视觉对象的使用方法，请参阅［RC+ Extensions 的使用］及《Vision Guide 8.0 软件篇 自定义视觉对象》。

6.2 开始使用

扩展功能为 Windows DLL，需要使用 Visual C++ 进行开发，因此必须安装 Visual Studio。请提前安装 RC+ Extensions 通用 VSIX。详细信息请参阅［5.2 安装］。

我们在 GitHub 上提供了示例程序。有关程序的实现方法等内容，也请参考此处。

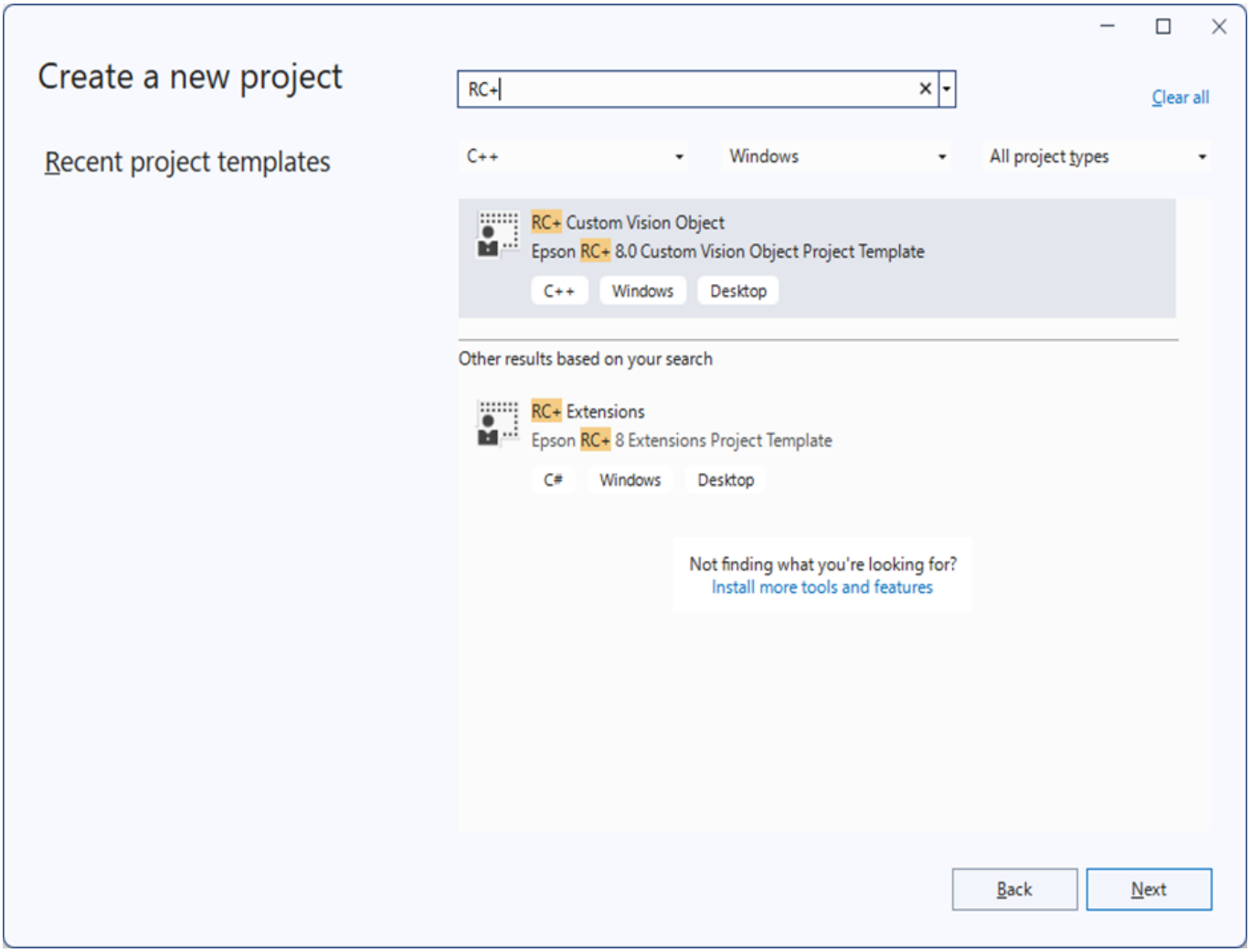
https://github.com/Epson-Robots/rcplus-extensions

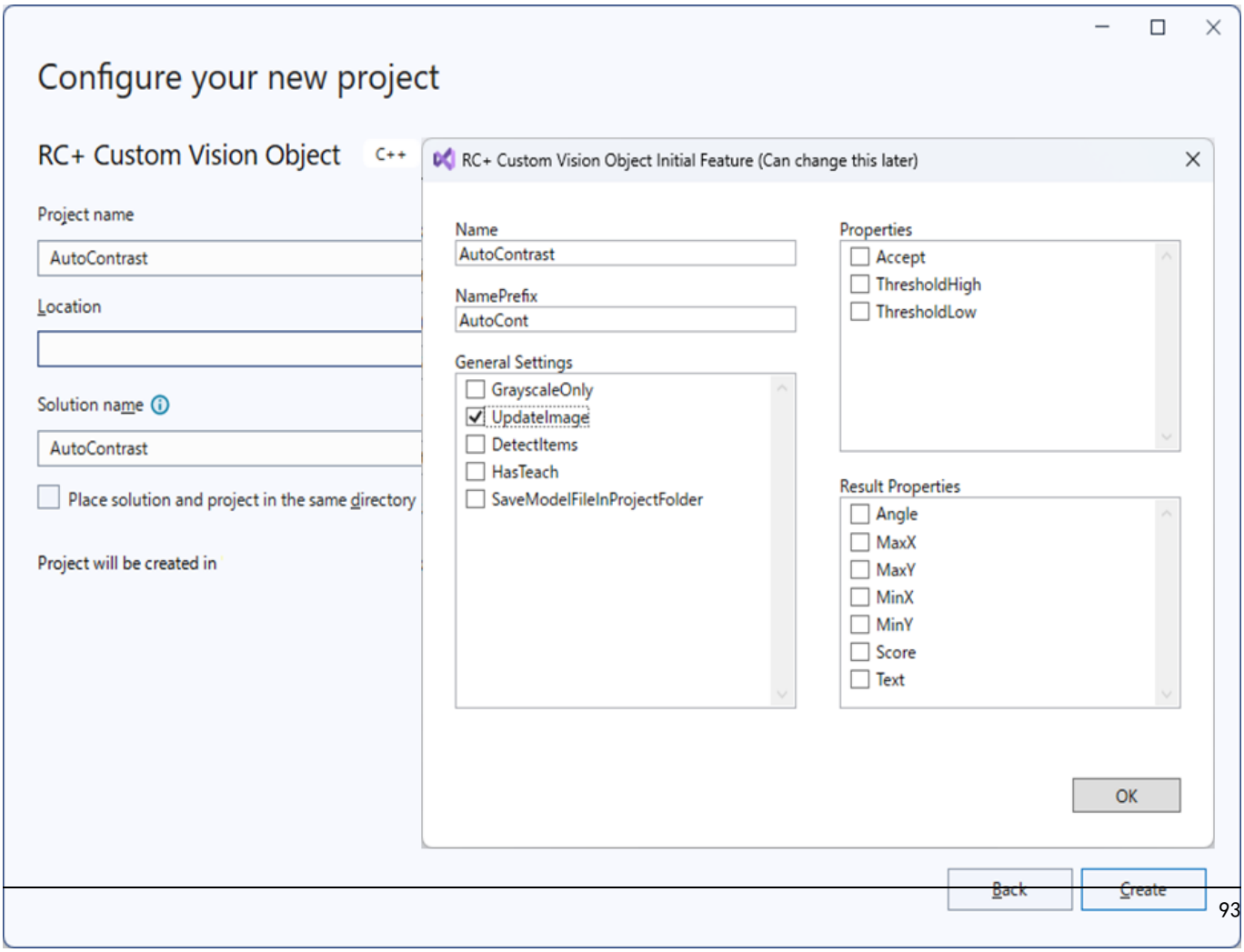
6.3 教程

通过以下步骤，可以创建自定义视觉对象。

6.3.1 项目的创建

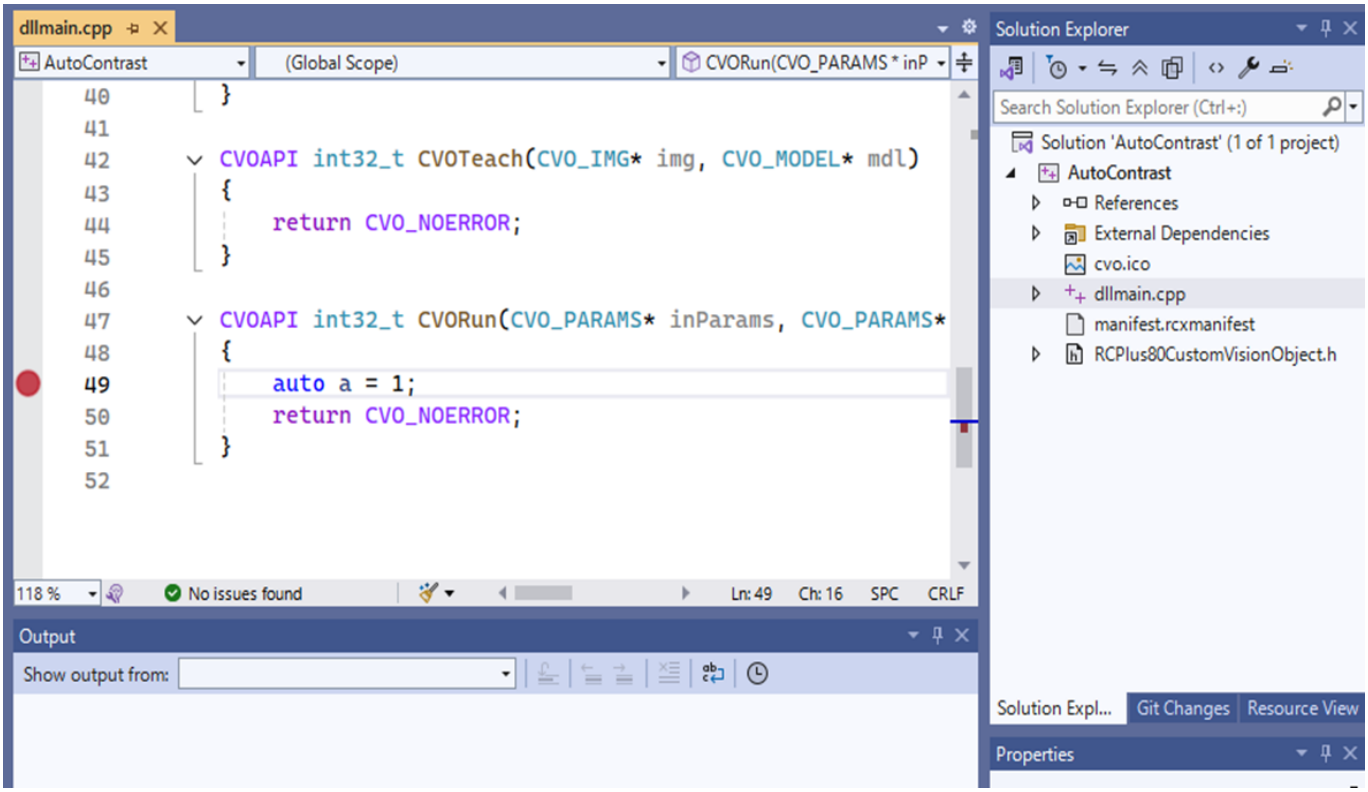
RC+ Extensions 通用的VSIX包含了用于自定义视觉对象的 Visual Studio 项目模板“RC+ Custom Vision Object”。
请使用该模板创建 Visual Studio 项目。
在项目向导创建过程中，将显示“RC+ Custom Vision Object Feature”界面，请输入您的自定义视觉对象的配置信息。
此设置可在后续的实现步骤中进行更改。


Create a new project
RC+
Clear all
Recent project templates
C++
Windows
All project types
RC+ Custom Vision Object
Epson RC+ 8.0 Custom Vision Object Project Template
C++
Windows
Desktop
Other results based on your search
RC+ Extensions
Epson RC+ 8 Extensions Project Template
C#
Windows
Desktop
Not finding what you're looking for?
Install more tools and features
Back
Next


Configure your new project
RC+ Custom Vision Object
C++
Project name
AutoContrast
Location
Solution name
AutoContrast
Place solution and project in the same directory
Project will be created in
RC+ Custom Vision Object Initial Feature (Can change this later)
Name
AutoContrast
NamePrefix
AutoCont
General Settings
GrayscaleOnly
UpdateImage
DetectItems
HasTeach
SaveModelFileInProjectFolder
Properties
Accept
ThresholdHigh
ThresholdLow
Result Properties
Angle
MaxX
MaxY
MinX
MinY
Score
Text
OK
Back
Create

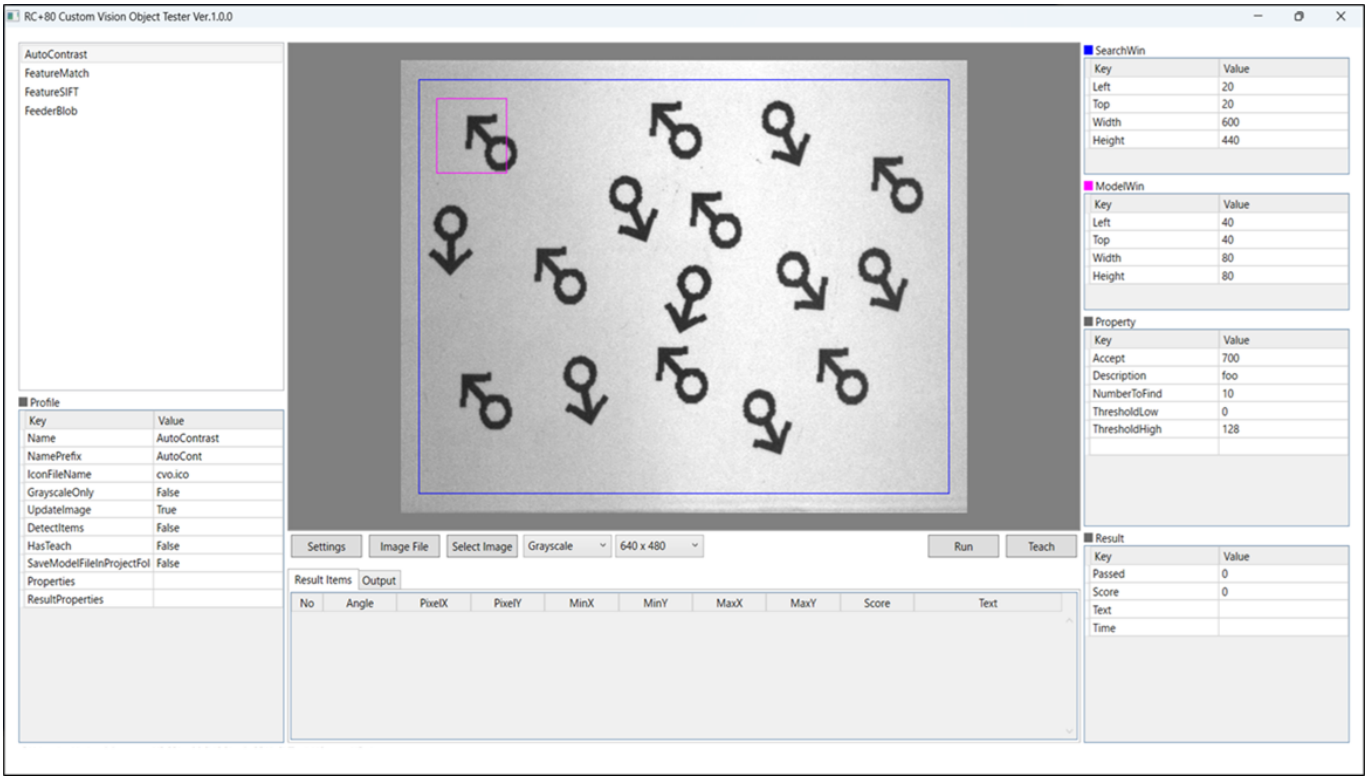
6.3.2 实现

创建项目后，在 dllmain.cpp 中进行实现。
由于该文件已预先实现了所需的 API（导出函数），因此请在函数内部进行具体实现。



6.3.3 调试

请将目标项目设置为启动项目，并通过调试方式运行。
用于自定义视觉对象开发的工具将被启动，您可以确认自定义视觉对象的运行情况。此外，通过设置断点，也可以进行调试执行。



6.3.4 包的创建

请对目标项目进行发布构建。在解决方案文件夹下会生成 Packages 文件夹，在该文件夹下会创建扩展名为 rcxpkg 的自定义视觉对象包。通过在扩展功能管理器中安装该包，您的自定义视觉对象即可在 RC+ 中使用。

6.3.5 API说明

API由四个函数组成。
请参考以下说明进行实现。
关于API的详细信息，已在 GitHub 提供了《RC+ Extension 自定义视觉对象 API 参考》。API 的详细信息已在此处记载。请一并进行确认。

https://github.com/Epson-Robots/rcplus-extensions

6.3.5.1 CVOGetAPIVersion

此函数在加载包时用于检查API的版本。
请保持此函数的实现不变，直接使用。

```
CVOAPI int32_t CVOGetAPIVersion()
{
    return CVO_API_VERSION;
}
```

6.3.5.2 CVOGetProfile

该函数在 RC+ 启动时被调用。
请将您的自定义视觉对象的配置文件信息设置到 CVO_PROFILE 结构体中并返回。
各项设置请参阅 API 参考文档。Properties 和 ResultProperties 需以逗号分隔的字符串形式指定属性。关于可设置的属性，请参阅“6.3.6 属性”章节。

```
CVOAPI int32_t CVOGetProfile(CVO_PROFILE* profile)
{
    strcpy_s(profile->Name, CVO_VALUE_MAX_LENGTH, "InvertColor");
    strcpy_s(profile->NamePrefix, CVO_VALUE_MAX_LENGTH, "Invert");
    strcpy_s(profile->IconFileName, CVO_VALUE_MAX_LENGTH, "InvertColor.ico");
    profile->GrayscaleOnly = true;
    profile->UpdateImage = true;
    profile->DetectItems = true;
    profile->HasTeach = false;
    profile->SaveModelFileInProjectFolder = false;
    strcpy_s(profile->Properties, CVO_VALUE_MAX_LENGTH, "");
    strcpy_s(profile->ResultProperties, CVO_VALUE_MAX_LENGTH, "");

    return CVO_NOERROR;
}
```

6.3.5.3 CVOTeach

此函数在对象的示教过程中被调用。
如果不需要示教，则无需实现。

```
CVOAPI int32_t CVOTeach(CVO_PARAMS* inParams, CVO_PARAMS* outParams, CVO_IMG* img,
CVO_MODEL* mdl)
{
    return CVO_NOERROR;
}
```

将模型窗口的图像信息传递给 CVO_IMG。请将其作为模型信息使用。
也可以在 RC+ 端管理已示教的模型信息。
如需在 RC+ 端进行管理，请将二进制信息返回给 CVO_MODEL。

```
CVOAPI int32_t CVOTeach(CVO_IMG* img, CVO_MODEL* mdl)
{
    int32_t width = img->Width;
    int32_t height = img->Height;
    int32_t stride = img->Stride;

    mdl->Size = 4 + 4 + 4 + stride * height;
    memcpy(mdl->pBuffer + 0, &width, 4);
    memcpy(mdl->pBuffer + 4, &height, 4);
    memcpy(mdl->pBuffer + 8, &stride, 4);
    memcpy(mdl->pBuffer + 12, img->pBuffer, stride * height);

    return CVO_NOERROR;
}
```

6.3.5.4 CVORun

该函数会在对象执行时被调用。
搜索窗口的图像信息通过 CVO_IMG 传递。
如需进行图像转换，请修改该 CVO_IMG 的信息。

```
CVOAPI int32_t CVORun(CVO_PARAMS* inParams, CVO_PARAMS* outParams,
CVO_DETECT_ITEMS* detectItems, CVO_IMG* img, CVO_MODEL* mdl)
{
    for (int y = 0; y < img->Height; y++)
    {
        for (int x = 0; x < img->Width; x++)
        {
            int idx = img->Stride * y + img->BytesPerPixel * x;

            if (img->BytesPerPixel == 3)
            {
                img->pBuffer[idx + 0] = 255 - img->pBuffer[idx + 0];
                img->pBuffer[idx + 1] = 255 - img->pBuffer[idx + 1];
                img->pBuffer[idx + 2] = 255 - img->pBuffer[idx + 2];
            }
            else if (img->BytesPerPixel == 1)
            {
                img->pBuffer[idx] = 255 - img->pBuffer[idx];
            }
        }
    }

    return CVO_NOERROR;
}
```

如需提供检测位置信息，请在 CVO_DETECT_ITEMS 中设置相关信息。

```
CVOAPI int32_t CVORun(CVO_PARAMS* inParams, CVO_PARAMS* outParams,
CVO_DETECT_ITEMS* detectItems, CVO_IMG* img, CVO_MODEL* mdl)
{
    detectItems->Count = 10;

    for (auto i = 0; i < detectItems->Count; i++)
    {
        detectItems->Items[i].PixelX = rand() % img->Width;
        detectItems->Items[i].PixelY = rand() % img->Height;
    }

    return CVO_NOERROR;
}
```

6.3.6 属性

在 Vision Guide 界面中可显示的属性会根据自定义视觉对象的配置文件信息设置情况而变化。请确认以下内容。

6.3.6.1 属性

Property	Extension Accessible	Display Always	Display DetectItems	Display HasTeach	Display by Extension Profile Settings
AbortSeqOnFail		X			
Accept	X				X
Caption		X			

Property	Extension Accessible	Display Always	Display DetectItems	Display HasTeach	Display by Extension Profile Settings
CurrentResult			X		
Description	X	X			
Enabled		X			
FailColor		X			
Graphics		X			
LabelBackColor		X			
ModelWinCenterX				X	
ModelWinCenterY				X	
ModelWinHeight				X	
ModelWinLeft				X	
ModelWinTop				X	
ModelWinWidth				X	
Name		X			
NumberToFind	X		X		
PassColor		X			
PassType			X		
SearchWinCenterX		X			
SearchWinCenterY		X			
SearchWinHeight		X			
SearchWinLeft		X			
SearchWinTop		X			
SearchWinWidth		X			
Sort			X		
ThresholdHigh	X				X
ThresholdLow	X				X

可以在配置信息的 Properties 中设置以下属性。

- Accept
- ThresholdHigh
- ThresholdLow

6.3.6.2 结果属性

Property	Extension Accessible	Display Always	Display DetectItems	Display HasTeach	Display by Extension Profile Settings
Angle	X		X *1		X

Property	Extension Accessible	Display Always	Display DetectItems	Display HasTeach	Display by Extension Profile Settings
CameraX			X		
CameraY			X		
Found			X		
MaxX	X		X *1		X
MaxY	X		X *1		X
MinX	X		X *1		X
MinY	X		X *1		X
NumberFound			X		
Passed		X			
PixelX	X		X		
PixelY	X		X		
RobotX			X		
RobotY			X		
RobotU			X		
Scale	X		X *1		X
Score	X				X
Text	X				X
Time		X			

*1 DetectItems 和 ResultProperties 均已设置时显示。

在配置文件信息中，可设置的 ResultProperties 属性信息如下。

- Angle
- MaxX/MaxY/MinX/MinY
- Scale
- Score
- Text