Logix Designer SDK Getting Results Guide

1756 ControlLogix, 1756 GuardLogix, 1769 CompactLogix, 1769 Compact GuardLogix, 5069 CompactLogix, Emulate 5570

Rockwell Automation

Getting Results Guide

Original Instructions

Important User Information

Read this document and the documents listed in the additional resources section about installation, configuration, and operation of this equipment before you install, configure, operate, or maintain this product. Users are required to familiarize themselves with installation and wiring instructions in addition to requirements of all applicable codes, laws, and standards.

Activities including installation, adjustments, putting into service, use, assembly, disassembly, and maintenance are required to be carried out by suitably trained personnel in accordance with applicable code of practice.

If this equipment is used in a manner not specified by the manufacturer, the protection provided by the equipment may be impaired.

In no event will Rockwell Automation, Inc. be responsible or liable for indirect or consequential damages resulting from the use or application of this equipment.

The examples and diagrams in this manual are included solely for illustrative purposes. Because of the many variables and requirements associated with any particular installation, Rockwell Automation, Inc. cannot assume responsibility or liability for actual use based on the examples and diagrams.

No patent liability is assumed by Rockwell Automation, Inc. with respect to use of information, circuits, equipment, or software described in this manual.

Reproduction of the contents of this manual, in whole or in part, without written permission of Rockwell Automation, Inc., is prohibited.

Throughout this manual, when necessary, we use notes to make you aware of safety considerations.

WARNING: Identifies information about practices or circumstances that can cause an explosion in a hazardous environment, which may lead to personal injury or death, property damage, or economic loss.

ATTENTION: Identifies information about practices or circumstances that can lead to personal injury or death, property damage, or economic loss. Attentions help you identify a hazard, avoid a hazard, and recognize the consequence.

IMPORTANT: Identifies information that is critical for successful application and understanding of the product.

These labels may also be on or inside the equipment to provide specific precautions.

SHOCK HAZARD: Labels may be on or inside the equipment, for example, a drive or motor, to alert people that dangerous voltage may be present.

BURN HAZARD: Labels may be on or inside the equipment, for example, a drive or motor, to alert people that surfaces may reach dangerous temperatures.

ARC FLASH HAZARD: Labels may be on or inside the equipment, for example, a motor control center, to alert people to potential Arc Flash. Arc Flash will cause severe injury or death. Wear proper Personal Protective Equipment (PPE). Follow ALL Regulatory requirements for safe work practices and for Personal Protective Equipment (PPE).

The following icon may appear in the text of this document.

Tip: Identifies information that is useful and can help to make a process easier to do or easier to understand.

Rockwell Automation recognizes that some of the terms that are currently used in our industry and in this publication are not in alignment with the movement toward inclusive language in technology. We are proactively collaborating with industry peers to find alternatives to such terms and making changes to our products and content. Please excuse the use of such terms in our content while we implement these changes.

Summary of changes

This manual includes new and updated information. Use these reference tables to locate changed information.

Global changes

None for this release.

New or enhanced features

Subject	Reason
Use the Python client on page 9	New functionality.

Contents

Preface

This manual provides a programmer with details and examples for using the Logix Designer Software Development Kit (SDK) to manage Logix Designer programs with programmable tools and functionality.

If you design, program, or troubleshoot safety applications that use GuardLogix controllers, refer to the GuardLogix Safety Application Instruction Set Safety Reference Manual, publication 1756-RM095.

This manual is one of a set of related manuals that show common procedures for programming and operating Logix 5000 controllers.

For a complete list of common procedures manuals, refer to the Logix 5000 Controllers Common Procedures Programming Manual, publication 1756-PM001.

The term Logix 5000 controller refers to any controller based on the Logix 5000 operating system. Rockwell Automation recognizes that some of the terms that are currently used in our industry and in this publication are not in alignment with the movement toward inclusive language in technology. We are proactively collaborating with industry peers to find alternatives to such terms and making changes to our products and content. Please excuse the use of such terms in our content while we implement these changes.

Additional resources

These documents contain additional information concerning related Rockwell Automation products.

Resource	Description
Industrial Automation Wiring and Grounding Guidelines, publication, 1770-4.1	Provides general guidelines for installing a Rockwell Automation industrial system.
Rockwell Automation product certifications	Provides declarations of conformity, certificates, and other certification details.

View or download publications at https://www.rockwellautomation.com/en-us/support/documentation/literature-library.html. To order paper copies of technical documentation, contact a local Rockwell Automation distributor or sales representative.

Legal notices

Rockwell Automation publishes legal notices, such as privacy policies, license agreements, trademark disclosures, and other terms and conditions on the Legal Notices page of the Rockwell Automation website.

Software and Cloud Services Agreement

Review the Rockwell Automation Software and Cloud Services Agreement here.

Open Source Software Licenses

The software included in this product contains copyrighted software that is licensed under one or more open source licenses.

Select this link to view a full list of all open source software used in this product and their corresponding licenses:

Logix Designer SDK open-source licenses

You may obtain Corresponding Source code for open source packages included in this product from their respective project web site(s). Alternatively, you may obtain complete Corresponding Source code by contacting Rockwell Automation via the **Contact** form on the Rockwell Automation website: http://www.rockwellautomation.com/global/about-us/contact/contact.page. Please include "Open Source" as part of the request text.

Overview

The Logix Designer Software Development Kit (SDK) provides programmatic access for third-party applications to manage and interact with the Logix Designer application. Use the LogixProject class and ILogixProject interface to set up the SDK and manage Logix Designer programs with programmable tools and functionality. For a complete description of the SDK functions and requirements, see the Logix Designer SDK online help.

Considerations when using the Logix Designer SDK

Keep these considerations in mind when using the Logix Designer SDK.

- The SDK supports projects for Logix Designer version 31 and later. Projects for versions 30 and earlier are not supported, other than to convert the project to version 31 and later.
- The SDK supports FactoryTalk Linx as its communication software. FactoryTalk RSLinx is not supported.
- The controller that you configure to be used by the communication path must be previously configured or visible over the FactoryTalk Linx Network Browser.
- The Logix Designer SDK supports only a 64-bit OS.
- The maximum size for Logix Designer projects is 500MB.
- The maximum size for a single Logix Designer SDK operation data file is 30kB.
- The Logix Designer SDK server uses this specific TCP port: 53204. Make sure this port is free. You can configure the port number if you use the C# SDK client.
- Visual Studio 2022 must be installed on the computer on which you run the Logix Designer SDK.

Set up a Logix Designer SDK client

The Logix Designer SDK supports Python, C#, and C++ clients.

This table lists the versions that support each client.

Supported clients	Version
Python, C#, and C++	2.01 and later
C# and C++	1.01 and later
C++	1.00 and later

Use the C# client

The Logix Designer SDK versions 2.00 and later support the C# client. (Version 1.01 supports the C# sharp client with limited functionality.)

C# is the programming language that most directly reflects the structures in the Common Language Infrastructure (CLI).

Set up the C# environment

Follow these steps to set up the C# environment.

1. Install the Logix Designer SDK.
2. Navigate to the SDK installation and open the .sln file in your preferred Integrated Development Environment (IDE), or create a new solution of your own.
3. Create a new project and add `RockwellAutomation.LogixDesigner.CSClient` as a dependency.

NOTE: Before you build, make sure to set the architecture to that of the system that will be running your executable file.

Develop C# projects

To get started writing your C# project, you can create a new project in the examples directory (the default directory is C:\Users\Public\Documents\Studio 5000\Logix Designer SDK\dotnet\Examples) or you can create your own solution in another location after creating a local `nuget.config` file.

For more information on creating a nuget.config file, see the Microsoft documentation at this URL:

https://learn.microsoft.com/en-us/nuget/reference/nuget-config-file

Use the Python client

The Logix Designer SDK versions 2.01 and later support the Python client. Python scripts can be written and executed without a compilation step, which allows for faster iteration and testing. It is also forgiving of errors that can cause your script not to work as intended, or end before execution completes.

Tip: If Python version 3.11 or earlier is already installed on your workstation, we recommend you uninstall it before installing Logix Designer SDK version 2.01 or later. Scripts run most efficiently when only one version of Python is installed.

Set up the Python environment

Follow these steps to set up the Python environment.

Tip: Some installations of Python require you to run the command Python while other installations might require you to run py. If these instructions do not work with python, try running py. For example:

```
py -m venv ./.venv
```

1. Install Logix Designer SDK version 2.01 or later.
2. (Optional) Install FactoryTalk Logix Echo.
3. Navigate to your SDK Python folder and create or activate a virtual environment.
   For example:

   ```
   cd 'C:\Users\Public\Documents\Studio 5000\Logix Designer SDK\python\Examples'
   python -m venv ./.venv
   ..venv\Scripts\activate
   ```

4. Use Pip to install the SDK.
   For example:

   ```
   pip install -r .\requirements.txt
   ```

5. Run an example:

   ```
   python .\get_comm_path.py C:\myAcds\myAcdFile.ACD
   ```

Develop Python projects

You can create Python projects (.py files) in the examples directory (for example, C:\Users\Public\Documents\Studio 5000\Logix Designer SDK\python\Examples) or in a directory that you specify.

The contents of requirements.txt can contain a relative path or an absolute path to the SDK .whl file. For more information on the requirements file format, see the Pip documentation at this URL:

https://pip.pypa.io/en/stable/reference/requirements-file-format/

Tip: The requirements.txt file is optional. To install the .whl file without a requirements file, create and start your virtual environment (venv) and use a command like this example:

```
pip install .\path\to\logix_designer_sdk-2.0.1-py3-none-any.whl
```

Use interactive development

Python is an interpreted language, so you can interact with the Python client in real time. You can leave a project file open and enter new commands at any time, so you can make updates and test script ideas immediately.

After you install the Python client, follow these steps to use interactive development.

1. Enter asyncio Python interpreter:

```
python -m asyncio
```

2. Execute some python SDK code:

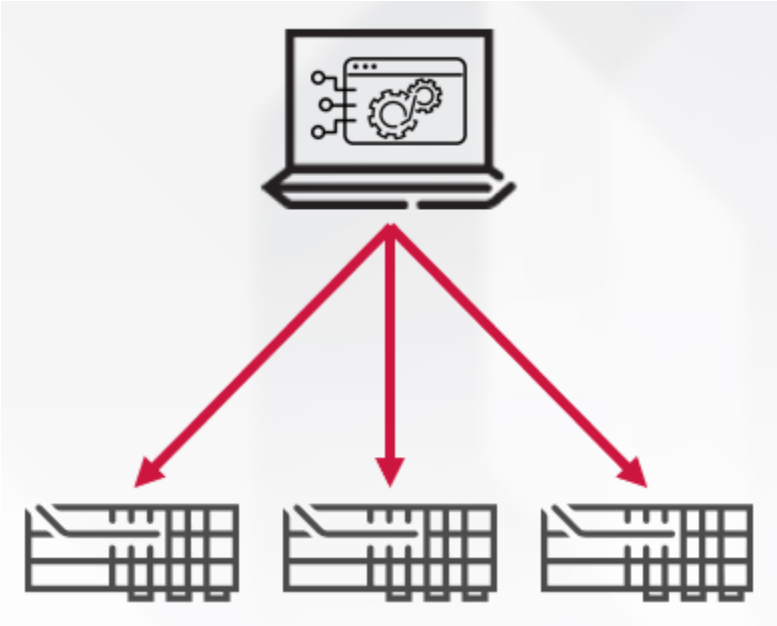
```
from logix_designer_sdk.logix_project import LogixProject

project = await LogixProject.open_logix_project(r"C:\myAcds\myAcd.ACD", StdOutEventLogger())

await project.set_communications_path("a/known/communication/path")

await project.get_communications_path()

'a/known/communication/path'
```

Download projects to controllers

Use the Logix SDK to build programs that download projects to controllers.

Download one project to multiple controllers

Use the Logix Designer SDK to create a program that downloads one project to multiple controllers. For example, if you have several identical machines set up and you need one project to run on all of them, you can create a program that completes the download steps.



The download program carries out these steps:

1. Opens the project in Logix Services.
2. Sets the communication path to the controller.
3. Switches the controller to Remote Program mode.
4. Downloads the project to the controller.
5. Without closing the project, repeats steps 2, 3, and 4 for each controller.

Example: Download one project to multiple controllers

This example code shows how to construct an application that downloads one project to multiple controllers.

C#

```
var programMode = LogixProject.RequestedControllerMode.Program;

var runMode = LogixProject.RequestedControllerMode.Run;

/*! Download project file to all controllers and set to run mode. !*/

foreach (var controller in _controllerCommPaths)
```

```
{
 LogixProject logixProject = await LogixProject.OpenLogixProjectAsync(_filePath);
 await logixProject.SetCommunicationsPathAsync(controller);
 await logixProject.ChangeControllerModeAsync(programMode);
 await logixProject.DownloadAsync();
 await logixProject.ChangeControllerModeAsync(runMode);
 /*! hold project reference for later use !*/
 logixSdkProjects.Add(controller, logixProject);
}
```

Python

```
programMode = RequestedControllerMode.PROGRAM
   runMode = RequestedControllerMode.RUN

   # Download ACD file to all controllers and set to run mode.
   for controller in DownloadCtrl.controller_comm_paths:
       logix_project = await LogixProject.open_logix_project(
           DownloadCtrl.file_path
       )
       await logix_project.set_communications_path(controller)
       await logix_project.change_controller_mode(programMode)
       await logix_project.download()
       await logix_project.change_controller_mode(runMode)

       # hold project reference for later use
       DownloadCtrl.logix_sdk_projects[controller] = logix_project
```

Download multiple projects to multiple controllers

Use the Logix Designer SDK to create a program that downloads multiple projects to multiple controllers. In this use case, each controller has its own project file to be downloaded.

The download program runs this sequence as parallel threads for many controllers:

1. Open the project.
2. Set the comm path to the controller.
3. Set the controller to Remote Program mode.
4. Download the project.

Example: Multiple projects to multiple controllers

This example code shows how to construct an application that downloads multiple projects to multiple controllers.

C#

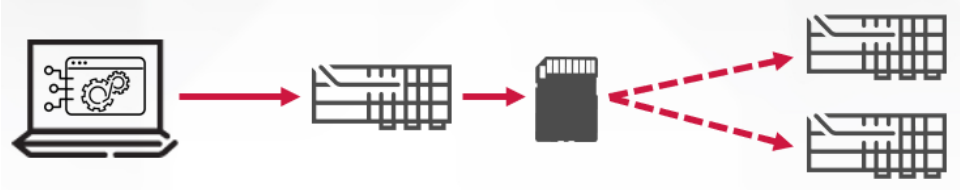
```
private static Dictionary<string, string> _controllers = new Dictionary<string, string>()
 {
  { CTRL1, FILE1 },
  { CTRL2, FILE2 },
  { CTRL3, FILE3 },
  { CTRL4, FILE4 },
  { CTRL5, FILE5 },
 };


 foreach( var (commPath, acdFile) in _controllers){
  /*! Each comm path/acdFile pair has a task to run the following lambda function added
  to the list of tasks created above. !*/
  taskList.Add(Task.Run(async ()=>
  {
   /*! Open project, set comm path, change mode to program, download, change mode to
   run. !*/
   var logixProject = await LogixProject.OpenLogixProjectAsync(acdFile);
   await logixProject.SetCommunicationsPathAsync(commPath);
   await
   logixProject.ChangeControllerModeAsync(LogixProject.RequestedControllerMode.Program);
   await logixProject.DownloadAsync();
   await
   logixProject.ChangeControllerModeAsync(LogixProject.RequestedControllerMode.Run);
   Console.WriteLine($"Finished downloading {acdFile} to ${commPath}");
  }));
 }
 return Task.WhenAll(taskList); /*! Return when all threads are complete !*
```

Python

```
 controllers = {
       ctrl1: file1,
       ctrl2: file2,
       ctrl3: file3,
       ctrl4: file4,
       ctrl5: file5,
   }
   """
   This function loops over each item in "controllers" above and
   queues all the necessary steps to download to a controller for each item.
   """


   tasks = []
   for comm_path, file_path in controllers.items():
       tasks.append(DownloadManyAcds.prepare_controller(comm_path, file_path))
   await asyncio.gather(*tasks)
```

Deploy a project and save it to an SD Card

Use the Logix Designer SDK to create a program that downloads a project to a controller and then saves the project to an SD card. Users can carry the SD card to other controllers and run the program on startup.



The program carries out these steps:

1. Opens the project in Logix Services.
2. Sets the communication path to the controller.
3. Switches the controller to Remote Program mode.
4. Downloads the project to the controller.
5. Saves the project to the SD card.

Example: Download a project and save to an SD card

This example code shows how to construct an application that downloads a project to a controller and saves the project to an SD card.

C#

```
using RockwellAutomation.LogixDesigner;



namespace CreateDeploymentSdCard
{
    /// >summary>
    /// Class which opens a >see cref="LogixProject"/>, goes online with a given controller,
 downloads the program,
    /// and then exposes the >see cref="LogixProject.StoreImageOnSDCardAsync"/> to the user. Uses the
 initializer pattern
    /// because the initial steps are asynchronous.
    /// >/summary>
    public class SDCardFactory
    {
        /// >summary>
        /// Reference to the >see cref="LogixProject"/> to be referenced later.
        /// >/summary>
        private LogixProject _project;


        /// >summary>
        /// Instantiates the SDCardFactory class.
```

```
        /// >/summary>
        /// >param name="project">The >see cref="LogixProject"/> to load to SD card.>/param>
        private SDCardFactory(LogixProject project)
        {
            _project = project;
        }

        /// >summary>
        /// A pass through method to >see cref="LogixProject.StoreImageOnSDCardAsync"/>
        /// >/summary>
        /// >returns>A task which resolves when the SD card write is complete.>/returns>
        public async Task LoadToSdCard()
        {
            await _project.StoreImageOnSDCardAsync();
        }

        /// >summary>
        /// Static initializer which opens the project and does all the one time
        /// setup tasks, then returns a new instance of >see cref="SDCardFactory"/>.
        /// >/summary>
        /// >param name="acdPath">The path to the ACD file.>/param>
        /// >param name="commPath">The comm path to the controller.>/param>
        /// >returns>An instance of >see cref="SDCardFactory"/>>/returns>
        public static async Task>SDCardFactory> Init(string acdPath, string commPath)
        {
            var project = await LogixProject.OpenLogixProjectAsync(acdPath); /*! Open the project !*/
            await project.SetCommunicationsPathAsync(commPath); /*! Set the comm path in the project
 to the one given by the user. !*/
            await
 project.ChangeControllerModeAsync(LogixProject.RequestedControllerMode.Program); /*! Set controller
 to program mode !*/
            await project.DownloadAsync(); /*! Download the program. !*/

            return new SDCardFactory(project);
        }
    }
}
```

Python

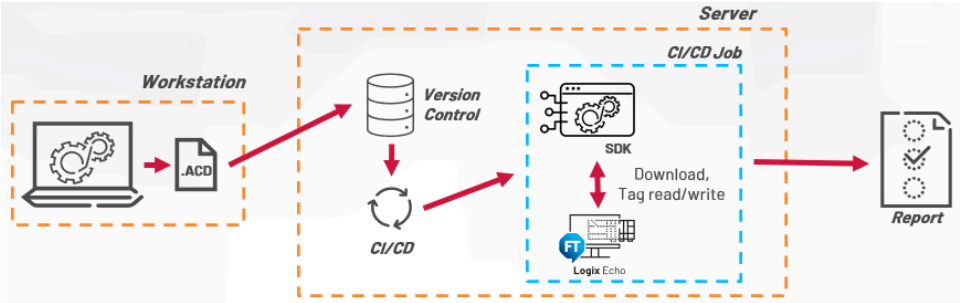
```
class SDCardFactory:
    """
    Class which opens a "LogixProject", goes online with a given controller, downloads the program,
    and then exposes the "LogixProject.store_image_on_sd_card" method to the user. Uses the
 initializer pattern
    because the initial steps are asynchronous.
```

```
    """


def __init__(self, project: LogixProject) -> None:
    """
    Instantiates an instance of the "SDCardFactory" class.
    """
    self.project = project


@staticmethod
async def init(file_path: str, comm_path: str):
    """
    Static initializer which opens the project and does all the one time
    setup tasks, then returns a new instance of "SDCardFactory".
    """
    # Open the project
    project = await LogixProject.open_logix_project(file_path, StdOutEventLogger())
    # Set the comm path in the project to the one given by the user.
    await project.set_communications_path(comm_path)
    # Set controller to program mode
    await project.change_controller_mode(RequestedControllerMode.PROGRAM)
    # Download the program
    await project.download()

    return SDCardFactory(project)


async def load_to_sd_card(self) -> None:
    """
    A pass through method to "LogixProject.store_image_on_sd_card"
    """
    await self.project.store_image_on_sd_card()
```

Download a project for automated testing

Use the Logix Designer SDK to create a program that downloads a project and runs it on an emulated controller.

- Commit a project file to version control.
- A CI/CD system, such as Jenkins, invokes a script that uses the Logix Designer SDK to download and run the project on an emulated controller.
- The script writes and reads tag values to verify expected file behavior, providing automated feedback as part of a CI/CD tool chain.



Example: Provision and validate

This example code shows how to construct an application that downloads a project and validates it by running it on an emulated or physical controller.

```
   /// <summary>
   /// This example shows a very basic demo of how to use LogixDesignerSDK to provision a controller and test for expected values.
   /// This could be used, for example, to retrieve a project file (.ACD, .L5X, or .L5K) from version control
   /// and then programatically test on a physical or emulated controller.
   /// You could (and likely should) also use libraries like NUnit, XUnit, or MSTest along with the demo helper class for an even more robust solution.
   ///
   /// This program compiles to an executable that takes one argument: The file path, and has a hard coded comm path.
   /// For examples on taking more than one argument, see CreateDeploymentSdCard.
   /// </summary>
   internal class HowToUse
   {
       public static async Task<int> Main(string[] args)
       {
           int failedTests = 0;
           // in this example, you could pass the string in as a command line parameter to an EXE you compile.
           string path = args[0];
           // Let's pretend we're using FatoryTalk Echo and running this on a ci device to test.
```

```
            string commPath = "EmulateEthernet\\127.0.0.7";
            var projectToTest = await ProvisionAndValidate.Init(path, commPath);

            // These helper methods are declared below. In a larger solution, you may want to move
 them to their own class and expand on them.
            string stringTagPath = CreateTagPathFromName("myString");
            string dintTagPath = CreateTagPathFromName("myDint");
            try /*! The ProvisionAndValidate class throws an error if the values do not match. !*/
            {
                await projectToTest.StringValueMatches(stringTagPath, "expectedValue");
            }
            catch { failedTests++;} /*! In this instance we only count the test as failed. You may
 wish to report more information. !*/

            try
            {
                await projectToTest.DINTValueMatches(dintTagPath, 42);
            }
            catch { failedTests++;}

            if(failedTests == 0)
            {
                Console.WriteLine("All tests passed!!!");
                return 0;
            }
            else
            {
                Console.WriteLine($"There were {failedTests} failed tests.");
                return 1; // Some tests failed. End program with error.
            }
        }

        private static string CreateTagPathFromName(string name)
        {
            const string tagPathPrefix = "Controller/Tags/Tag[@Name='";
            const string tagPathSuffix = "']";
            return $"{tagPathPrefix}{name}{tagPathSuffix}";
        }
    }

}
```

Open and save or export a project

Use the Logix Designer SDK to create a program that opens and saves or exports a project.

Your program can open, save, or export a project to these file types:

- ACD
- L5K
- L5X

This parameter is required:

- Destination path for the saved or exported project file.

These parameters are optional:

- A boolean flag to force overwriting an existing file; this flag is false by default.
- A cancellation token that can cancel the operation.
- An option for including extra information in the exported project file; this option is false by default.

Example: Open and save or export a project

This example code shows how to construct an application that saves or exports a project.

```
using LogixProject project = await LogixProject.OpenLogixProjectAsync(projectFilePath, new
 StdOutEventLogger());


       await project.SaveAsAsync(saveProjectPath, true, detailedL5x);


       var fileBytes = new FileInfo(saveProjectPath).Length;
       if (!(fileBytes > 0))
       {
           Console.WriteLine("Unable to save project: An unknown error has occured");
           return 1;
       }


       Console.WriteLine($"Project has been saved to: {saveProjectPath}.");
```

Export a project component

Export a component of a project so you can reuse it. When you export a selected component from a project to an .L5X file, the file contains the definition of that component and everything else from the project that is referenced by that component, such as tags, User-defined Data Types, and Add-On Instructions. You can import these components into the same project or a different project.

Keep these considerations in mind when exporting components:

- When you export components from safety projects that contain safety signatures, SafetySignaturesHmac is not exported. The exported file can be imported, but because SafetySignaturesHmac is not present, all safety signatures are removed.
- You cannot export Controller/DataTypes/DataType[@Class='*'] for Logix Designer versions 32 and earlier.
-

You can export these components:

- Groups of rungs
- Routines
- Programs and equipment phases
- User-defined Data Types (UDT)
- Add-On Instructions

Example: Export project components

This example code shows how to export specific project components to an .L5X file.

```
LogixProject project;
try
{
    project = await LogixProject.OpenLogixProjectAsync(projectPath);
}
catch (LogixSdkException ex)
{
    Console.WriteLine($"Unable to open project at {projectPath}");
    Console.WriteLine(ex.Message);
    return 1;
}

try
{
    await project.PartialExportToXmlFileAsync(xPath, filePath);
}
catch (LogixSdkException ex)
{
    Console.WriteLine($"Unable to export {xPath} to {filePath}");
    Console.WriteLine(ex.Message);
    return 1;
}

Console.WriteLine("Partial export was successful");
```

Import a project component

Import project components into a project to reuse them. You can import components when online or offline with a controller.

Importing a component and its associated components is like doing a number of edits to your project file in one operation. You could do all these edits separately, but importing a component gives you the ability to do them all at once.

NOTE: Importing components can affect your project in unexpected ways. For a complete list of considerations to keep in mind when importing components, both online and offline, consult the Logix 5000 Controllers Import/Export Project Components Programming Manual (1756-PM019).

You can import these components:

- Groups of rungs
- Routines
- Programs and equipment phases
- User-defined Data Types (UDT)
- Add-On Instructions

For a detailed description of component structure, consult the Logix 5000 Controllers Import/Export Reference Manual.

Example: Import a project component offline

This example code shows how to import specific project components into a project while offline with a controller. You can also import a component when online with a controller.

NOTE: Importing components can affect your project in unexpected ways. For a complete list of considerations to keep in mind when importing components, both online and offline, consult the Logix 5000 Controllers Import/Export Project Components Programming Manual (1756-PM019).

```
LogixProject project;
try
{
    project = await LogixProject.OpenLogixProjectAsync(projectPath);
}
catch (LogixSdkException ex)
{
    Console.WriteLine($"Unable to open project at {projectPath}");
    Console.WriteLine(ex.Message);
    return 1;
}
```

```
       try
       {
           await project.PartialImportFromXmlFileAsync(xPath, filePath,
LogixProject.ImportCollisionOptions.OverwriteOnColl);
           await project.SaveAsync();
       }
       catch (LogixSdkException ex)
       {
           Console.WriteLine($"Unable to import {xPath} from {filePath}");
           Console.WriteLine(ex.Message);
           return 1;
       }

       Console.WriteLine("Partial import was successful");
```

Convert a project to a newer Logix Designer version

Use the Logix Designer SDK to create a program that converts an older Logix Designer ACD, L5K, or L5X project file to a newer version of Logix Designer, from v31 to the most recent release.

Your program requires these parameters:

- An ACD, L5K, or L5X project path to convert.
- The installed version of Logix Designer to which you want to convert the file.

Your program can convert a project from any older version of Logix Designer, which is not installed, to any version, from v31 to the latest release, of Logix Designer that is installed.

However, your program cannot:

- Convert from a newer version to an earlier version; for example, you cannot convert a v34 ACD file to a v33 project file.
- Convert to a version of Logix Designer that is not installed.
- Convert to version 30 and earlier. You can only convert to Logix Designer versions 31 and later.

Example: Convert a project to a newer Logix Designer version

This example code shows how to construct an application that converts an older project to a newer version of Logix Designer, from v31 to the most recent release.

```
LogixProject project;


try
{
    project = await LogixProject.ConvertAsync(projectFilePath, majorRevision);
}
catch (LogixSdkException ex)
{
    Console.WriteLine("Unable to convert project");
    Console.WriteLine(ex.Message);
    return 1;
}


try
{
    await project.SaveAsAsync(saveProjectPath, true);
}
catch (LogixSdkException ex)
{
    Console.WriteLine("Unable to save project");
```

```
            Console.WriteLine(ex.Message);
            return 1;
        }
        var fileBytes = new FileInfo(saveProjectPath).Length;
        if (!(fileBytes > 0))
        {
            Console.WriteLine("Unable to save project: An unknown error has occured");
            return 1;
        }

        Console.WriteLine($"Project has been saved to: {saveProjectPath}.");
```

Convert a controller to a compatible controller

Use the Logix Designer SDK to create a program that converts the controller in a project to a different, compatible controller.

This table lists the compatible controller conversions:

Convert from these controllers	To any of these controllers
• Compact GuardLogix 5370 • CompactLogix 5370 • ControlLogix 5570 • GuardLogix 5570	• Compact GuardLogix 5370 • CompactLogix 5370 • ControlLogix 5570 • GuardLogix 5570 • Compact GuardLogix 5380 • CompactLogix 5380 • CompactLogix 5480 • ControlLogix 5580 • GuardLogix 5580
• Compact GuardLogix 5380 • CompactLogix 5380 • CompactLogix 5480 • ControlLogix 5580 • GuardLogix 5580	• Compact GuardLogix 5380 • CompactLogix 5380 • CompactLogix 5480 • ControlLogix 5580 • GuardLogix 5580

However, you **cannot** convert to an incompatible controller. This table lists invalid controller conversions:

Cannot convert from these controllers	To these controllers
• Compact GuardLogix 5380 • CompactLogix 5380 • CompactLogix 5480 • ControlLogix 5580 • GuardLogix 5580	• Compact GuardLogix 5370 • CompactLogix 5370 • ControlLogix 5570 • GuardLogix 5570

Example: Convert a controller to a compatible controller

This example code shows how to construct an application that converts the controller in a project to a different, compatible controller.

```
       LogixProject project;


       try
       {
           processorTypes = await LogixProject.GetProcessorTypesAsync(majorRev);


           var validSelection = processorTypes.Contains(controllerType);
           if(!validSelection)
           {
               throw new ArgumentException($"Logix revision {majorRev} does not contain an entry for
controller type {controllerType}");
           }
       }
       catch(LogixSdkException ex) {
           Console.WriteLine($"Unable to get processor types dictionary for rev {majorRev}");
           Console.WriteLine(ex.Message);
           return 1;
       }


       try
       {
           project = await LogixProject.OpenLogixProjectAsync(projectPath);
       }
       catch (LogixSdkException ex)
       {
           Console.WriteLine($"Unable to open project at {projectPath}");
           Console.WriteLine(ex.Message);
           return 1;
       }


       processorTypeInfo = processorTypes[controllerType];


       try
       {
           await project.ChangeControllerTypeAsync(processorTypeInfo);
       }
       catch (LogixSdkException ex)
       {
           Console.WriteLine($"Unable to change controller to {processorTypeInfo.Name}");
           Console.WriteLine(ex.Message);
```

```
            return 1;
        }


        try
        {
            string fileDir = Path.GetDirectoryName(projectPath);
            string fileName = Path.GetFileNameWithoutExtension(projectPath);
            string fullPath = $"{fileDir}\\{fileName}_converted.acd";
            await project.SaveAsAsync(fullPath);
        }
        catch (LogixSdkException ex)
        {
            Console.WriteLine($"Unable to save project");
            Console.WriteLine(ex.Message);
            return 1;
        }



        Console.WriteLine($"Controller type was changed to {controllerType} DONE.");
```

Lock, unlock, or modify the password for Safety applications

Use the Logix Designer SDK to change the lock state of a Safety application. Locking the Safety application protects the application from unauthorized changes, and prevents editing of safety components.

The Logix Designer SDK provides options to:

- Lock the Safety application
- Unlock the Safety application
- Change the lock password
- Remove the lock password

Example: Lock, unlock, or modify the password for Safety applications

This example code shows how to lock, unlock, or modify the password for Safety applications.

Open the Logix Designer project

```
      LogixProject project;
      try
      {
          project = await LogixProject.OpenLogixProjectAsync(projectPath);
      }
      catch (LogixSdkException ex)
      {
          Console.WriteLine($"Unable to open project at {projectPath}");
          Console.WriteLine(ex.Message);
          return 1;
      }
```

Functional code for various safety lock operations

```
   private static async Task<int> CheckSafetyLockUnlockStatus(LogixProject project)
   {
       bool isSafetyLocked;
       try
       {
           isSafetyLocked = await project.IsSafetyLockedAsync();
           Console.WriteLine(String.Format("Safety lock status: {0}.", isSafetyLocked ? "Locked" :
"Unlocked"));
       }
       catch (LogixSdkException ex)
       {
           Console.WriteLine("Unable to read safety lock/unlock status.");
           Console.WriteLine(ex.Message);
```

```
            return 1;
        }

        return 0;
    }

    private static async Task<int> SafetyLock(LogixProject project, string safetyLockPassword)
    {
        try
        {
            await project.SafetyLockAsync(safetyLockPassword);
            Console.WriteLine($"Performed safety lock with password: {safetyLockPassword}.");
        }
        catch (LogixSdkException ex)
        {
            Console.WriteLine($"Unable to perform safety lock with password: {safetyLockPassword}.");
            Console.WriteLine(ex.Message);
            return 1;
        }
        return 0;
    }

    private static async Task<int> SafetyUnlock(LogixProject project, string safetyUnlockPassword)
    {
        try
        {
            await project.SafetyUnlockAsync(safetyUnlockPassword);
            Console.WriteLine($"Performed safety unlock with password: {safetyUnlockPassword}.");
        }
        catch (LogixSdkException ex)
        {
            Console.WriteLine($"Unable to perform safety unlock with password:
{safetyUnlockPassword}.");
            Console.WriteLine(ex.Message);
            return 1;
        }
        return 0;
    }

    private static async Task<int> ChangeSafetyLockPassword(LogixProject project, string oldPassword,
string newPassword)
    {
        try
        {
            await project.SetSafetyLockPasswordAsync(newPassword, oldPassword);
```

```
            Console.WriteLine($"Changed safety lock password to: {newPassword}.");
        }
        catch (LogixSdkException ex)
        {
            Console.WriteLine($"Unable to set safety lock password: {newPassword}.");
            Console.WriteLine(ex.Message);
            return 1;
        }
        return 0;
    }


    private static async Task<int> ChangeSafetyUnlockPassword(LogixProject project, string
oldPassword, string newPassword)
    {
        try
        {
            await project.SetSafetyUnlockPasswordAsync(newPassword, oldPassword);
            Console.WriteLine($"Changed safety unlock password to: {newPassword}.");
        }
        catch (LogixSdkException ex)
        {
            Console.WriteLine($"Unable to set safety unlock password: {newPassword}.");
            Console.WriteLine(ex.Message);
            return 1;
        }
        return 0;
    }
```

Rockwell Automation Support

Use these resources to access support information.

Technical Support Center	Find help with how-to videos, FAQs, chat, user forums, and product notification updates.	rok.auto/support
Knowledgebase	Access Knowledgebase articles.	rok.auto/knowledgebase
Local Technical Support Phone Numbers	Locate the telephone number for your country.	rok.auto/phonesupport
Literature Library	Find installation instructions, manuals, brochures, and technical data publications.	rok.auto/literature
Product Compatibility and Download Center (PCDC)	Get help determining how products interact, check features and capabilities, and find associated firmware.	rok.auto/pcdc

Documentation feedback

Your comments help us serve your documentation needs better. If you have any suggestions on how to improve our content, complete the form at rok.auto/docfeedback.

Waste Electrical and Electronic Equipment (WEEE)

At the end of life, this equipment should be collected separately from any unsorted municipal waste.

Rockwell Automation maintains current product environmental information on its website at rok.auto/pec.

Rockwell Otomasyon Ticaret A.Ş. Kar Plaza İş Merkezi E Blok Kat:6 34752 İçerenköy, İstanbul, Tel: +90 (216) 5698400 EEE Yönetmeliğine Uygundur

Connect with us.

rockwellautomation.com — expanding **human possibility**™

AMERICAS: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444
EUROPE/MIDDLE EAST/AFRICA: Rockwell Automation NV, Pegasus Park, De Kleetlaan 12a, 1831 Diegem, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640
ASIA PACIFIC: Rockwell Automation, Level 14, Core F, Cyberport 3, 100 Cyberport Road, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

LDSDK-GR001C-EN-P - January 2025
Supersedes Publication LDSDK-GR001B-EN-P - September 2024