Kollmorgen Automation Suite

Recipes

User Manual

Document Edition: A, July 2026
Valid for KAS Software Revision: 5.00
Part Number: N/A

For safe and proper use, follow these instructions.
Keep for future use.

KOLLMORGEN
A REGAL REXNORD BRAND

Table of Contents

1 Recipes

This documentation describes how to use Recipes in KAS.

It is intended for:

- Machine builders (OEMs).
- System integrators and distributors.
- Controls Engineers.
- End users and operators.

Recipes enable efficient handling of machine configuration data for different products, improving flexibility, consistency, and traceability.

1.1 What is a Recipe?

In industrial automation, a recipe is a named collection of parameter values that together define the configuration for a particular manufacturing process or product variant, sometimes referred to as a SKU.

Rather than hard-coding process parameters into the control program itself, recipes allow operators and engineers to separate what the machine does (the program logic) from how it is configured for a given job (the data).

For a PLCopen-based controller, a recipe file is:

- A structured data set that provides motion-related parameters (e.g., process values and kinematic parameters).
- Loaded at runtime and applied by PLCopen motion function blocks.

An application control logic can remain unchanged while a recipe dynamically configures how machine behavior and motion profiles are executed for a specific product or operation.

Recipes are a foundational concept in batch and discrete manufacturing and are formally described in the ISA-88 (IEC 61512) standard for batch process control.

See ISA-88 Series of Standards.

1.2 What Problems Do Recipes Solve?

Modern manufacturing lines frequently produce multiple product variants on the same equipment. Without recipe management, engineers face a difficult choice:

- Hard-code all variants into the program - resulting in complex, branching logic that is difficult to maintain and error-prone to modify.
- Manually adjust parameters at the HMI or in the source code - a slow, undocumented, and risky process that depends entirely on operator knowledge.

Recipe management eliminates both of these problems by providing a structured, systematic mechanism for storing and applying process parameters.

Problem	How Recipes Help
Product changeovers are slow.	A single operator action loads a complete parameter set instantly.
Parameter changes are undocumented.	Recipe files are named, versioned, and stored in a traceable format.
Operators can enter invalid values.	Recipe variables carry type information and can enforce validation limits.
Process knowledge is not documented.	Parameters are captured in files that can be backed up, shared, and audited.
Scaling to new variants requires code changes.	A new recipe file is created with no changes to the control program.

1.2.1 Example Use Cases

Example Industry	Use Case
Food and Beverage	Switching between product formulations (e.g., flavors, concentrations).
Pharmaceuticals	Batch manufacturing with strict parameter traceability requirements.
Labeling	Store label offset, print speed, and dispense timing for different container sizes.
Agricultural Automation	Define dryer temperature, airflow, and target moisture content for different grain types (e.g., corn, soybeans, wheat).
Laser Notching / Cutting	Define laser power, pulse frequency, cutting speed, and focus offset for different parts.

2 Core Concepts

The recipe system uses these terms:

Term	Meaning	Example
"Recipe Definition" (→ p. 6)	A group of recipe variables that belong together.	Processing parameters or bottling parameters.
"Recipe Files" (→ p. 6)	The files that stores definitions, instances, or both.	processing.xml or masterrecipes.toml.
"Recipe Instance" (→ p. 6)	The actual values used for one product, batch, or setup.	FizzNova settings or BubbleRush settings.
"Recipe Variable" (→ p. 7)	A single parameter mapped to a PLC program variable.	MixSpeedRPM, ProductName, FillPressure.

2.1 Recipe Definition

A recipe definition groups related variables together that collectively describe one configurable aspect of the machine.

Multiple recipe sets can exist in a project, each mapped to different parts of the program.

2.2 Recipe Files

Recipe files may contain:

- A recipe definition.
- Recipe instances.
- User-defined areas for non-recipe data or audit tracking fields.

Recipe files are designed to be:

- Saved and loaded at runtime.
- Transferred between machines.
- Archived for audit or compliance purposes.
- Edited offline.

2.3 Recipe Instance

A recipe instance is a set of values for all recipe variables associated with a recipe definition.

A recipe instance typically defines values for a single part, formulation, or other product.

2.4 Recipe Variable

A recipe is composed of recipe variables - individual parameters that are mapped to variables in the PLC program.

Each recipe variable has:

- A name that matches a variable in a PLC program.
- A data type (e.g., BOOL, INT, REAL, STRING).
  - See Data Types.
- A value used by the PLC program.
- Optional: Traits that define minimum/maximum limits and clipping actions.

IMPORTANT

- Elements of arrays and structures can be part a recipe definition.
- To set all values of an array or structure, each element must be set individually.

Trait	Description
HasMax	The recipe variable has a maximum value.
HasMin	The recipe variable has a minimum value.

OnOutOfRange_Clip	When a recipe instance is loaded and the value is out of range, the value is clipped.
	• The definition of clip is:
	  • If the value is less than the minimum, the minimum value is set.
	  • If the value is greater than the maximum, the maximum value is set.
	• See numpy.clip.

OnOutOfRange_Error	When a recipe instance is loaded and the value is out of range, an error is generated.
OnOutOfRange_Ignore	When a recipe instance is loaded and the value is out of range, the variable is skipped.

2.4.1 Data Types - Supported

- Bitfields
- BOOL
- BYTE
- DINT
- DWORD
- Enumerations
- INT
- LINT
- LREAL
- LWORD
- REAL
- SINT
- STRING
- TIME
- UDINT
- UINT
- ULINT
- USINT
- WORD

3 Quick Start: Basic Recipe Workflow

This is a very basic, overall recipe workflow procedure.

Procedure

1. Create or edit a recipe file in a supported format (e.g., XML or TOML).
2. Verify the recipe variable names match the PLC program variables.
3. Transfer the recipe file to the controller User Data area.
4. In the KAS project application code:
   a. Use the appropriate recipe function block to open, load, apply, save, or close the recipe at runtime.
   b. Monitor Done, Busy, Error, and ErrorID outputs to confirm successful execution or diagnose issues.

4 Recipe File Formats and Examples

4.1 Reserved Recipe Sections

IMPORTANT

- Do not use “Recipe” as a section name prefix.
- This designation is reserved for Kollmorgen’s KAS development team for feature enhancement purposes.
- Examples:
  - **NOT Acceptable**: RecipeOptions
  - **Acceptable**: Options_Recipes, OptionsRecipes, Options-Recipes

4.2 Choosing XML or TOML

KAS supports XML and TOML file formats for recipe files.

- Use XML when the workflow already relies on XML tools, schema validation, or existing automation.
- Use TOML when readability and simple hand editing are more important.

IMPORTANT

- In either format, keep variable names, data types, and instance names consistent with the PLC application.

4.3 Example: Recipe Definition File

NOTE

- The TOML format requires single keys with periods or brackets to be quoted.

XML

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Recipe>
  <RecipeDefinition>
    <Variable name="MyGlobalINT" type="INT"
      traits="HasMin|HasMax|OnOutOfRange_Clip"
      min="-100" max="100" />
    <Variable name="MyProgram.MySTRING" type="STRING" />
    <Variable name="MyProgram.Complex.Var[1]" type="DINT" />
  </RecipeDefinition>
</Recipe>
```

TOML

TOML

```
[recipe_definition]
[[recipe_definition.variable]]
name = 'MyGlobalINT'
type = 'INT'
traits = ['HasMin', 'HasMax', 'OnOutOfRange_Clip']
min = -100
max = 100

[[recipe_definition.variable]]
name = 'MyProgram.MySTRING'
type = 'STRING'

[[recipe_definition.variable]]
name = 'MyProgram.Complex.Var[1]'
type = 'DINT'
```

4.4 Example: Recipe Definition and Multiple Instance File

NOTE

- In the TOML format, each [[recipe_instance]] entry uses a [recipe_instance.meta] sub-table for instance metadata (e.g., name) and a [recipe_instance.data] sub-table for recipe variable values.
  - This separation ensures that metadata keys can never collide with recipe variable names.
- The [recipe_instance.meta] sub-table is optional; omitting it produces an unnamed instance.
- The TOML format requires single keys with periods or brackets to be quoted.

XML

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Recipe>
  <RecipeDefinition>
    <Variable name="StampProgram.PartCount"
      type="INT" traits="HasMin|HasMax|OnOutOfRange_Clip"
      min="1" max="100" />
    <Variable name="LabelProgram.PartName" type="STRING" />
  </RecipeDefinition>
  <RecipeInstance>
    <!-- A RecipeInstance without a name is
         assigned the name "Instance" -->
    <Variable name="StampProgram.PartCount">0</Variable>
    <Variable name="LabelProgram.PartName">Budget Boff</Variable>
  </RecipeInstance>
  <RecipeInstance name="Cogs">
    <Variable name="StampProgram.PartCount">10</Variable>
    <Variable name="LabelProgram.PartName">Classy Cog</Variable>
  </RecipeInstance>
  <RecipeInstance name="Widgets">
    <Variable name="StampProgram.PartCount">20</Variable>
    <Variable name="LabelProgram.PartName">Wonderful Widget</Variable>
  </RecipeInstance>
</Recipe>
```

TOML

TOML

```
[recipe_definition]
[[recipe_definition.variable]]
name = 'StampProgram.PartCount'
type = 'INT'
traits = ['HasMin', 'HasMax', 'OnOutOfRange_Clip']
min = 1
max = 100

[[recipe_definition.variable]]
name = 'LabelProgram.PartName'
type = 'STRING'

[[recipe_instance]]
# A recipe instance without a [recipe_instance.meta]
# sub-table is given the name 'Instance'
[recipe_instance.data]
'StampProgram.PartCount' = 0
'LabelProgram.PartName' = 'Budget Boff'

[[recipe_instance]]
[recipe_instance.meta]
name = 'Cogs'
[recipe_instance.data]
'StampProgram.PartCount' = 10
'LabelProgram.PartName' = 'Classy Cog'

[[recipe_instance]]
[recipe_instance.meta]
name = 'Widgets'
[recipe_instance.data]
'StampProgram.PartCount' = 20
'LabelProgram.PartName' = 'Wonderful Widget'
```

4.5 Example: Recipe Definition and Single Instance File

NOTE

- If a recipe instance is not given a name, the instance is assigned the name "Instance".
- This example does not assign a name.
- The TOML format requires single keys with periods or brackets to be quoted.

XML

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Recipe>
  <RecipeDefinition>
    <Variable type="INT" name="StampProgram.PartCount"
      traits="HasMin|HasMax"
      min="1" max="100" />
    <Variable type="STRING" name="LabelProgram.PartName" />
  </RecipeDefinition>
  <RecipeInstance>
    <Variable name="StampProgram.PartCount">0</Variable>
    <Variable name="LabelProgram.PartName">Budget Boff</Variable>
  </RecipeInstance>
</Recipe>
```

TOML

TOML

```
[recipe_definition]
[[recipe_definition.variable]]
name = 'StampProgram.PartCount'
type = 'INT'
traits = ['HasMin', 'HasMax']
min = 1
max = 100

[[recipe_definition.variable]]
name = 'LabelProgram.PartName'
type = 'STRING'

[[recipe_instance]]
[recipe_instance.data]
'StampProgram.PartCount' = 0
'LabelProgram.PartName' = 'Budget Boff'
```

4.6 Example: Recipe Definition and User Audit Entries

NOTE

- Recipes may include custom tags that can be used for auditing purposes.
- The user is responsible for adding audit-related tags to the recipe file.
- The TOML format requires single keys with periods or brackets to be quoted.

XML

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Recipe>
  <!-- Audit Information -->
  <Version>
    <Revision>4</Revision>
    <ChangeLog>Changed temperature</ChangeLog>
  </Version>

  <CreatedBy>
    <Username>j.smith</Username>
    <EmployeeID>EMP-04471</EmployeeID>
    <Role>Process Engineer</Role>
  </CreatedBy>

  <!-- Paint Process Information -->
  <RecipeDefinition>
    <Variable name="CuringProgram.Temperature"
      type="INT" traits="HasMin|HasMax"
      min="-10" max="250" />
    <Variable name="PaintProgram.ColorName"
      type="STRING" />
  </RecipeDefinition>

  <RecipeInstance name="Panels">
    <Variable name="CuringProgram.Temperature">85</Variable>
    <Variable name="PaintProgram.ColorName">Blue</Variable>
  </RecipeInstance>
</Recipe>
```

TOML

TOML

```
#Audit Information
[version]
revision = 4
change_log = 'Changed temperature.'

[author.created_by]
username = 'j.smith'
employee_id = 'EMP-04471'
role = 'Process Engineer'

#Paint Process Information
[recipe_definition]
[[recipe_definition.variable]]
name = 'CuringProgram.Temperature'
type = 'INT'
traits = ['HasMin', 'HasMax']
min = -10
max = 250

[[recipe_definition.variable]]
name = 'PaintProgram.ColorName'
type = 'STRING'

[[recipe_instance]]
[recipe_instance.meta]
name = 'Panels'
[recipe_instance.data]
'CuringProgram.Temperature' = 85
'PaintProgram.ColorName' = 'Blue'
```

4.7 Example: Recipe Instance File

NOTE

- Recipe instance files are only used with RecipeQuickLoad.
- The TOML format requires single keys with periods or brackets to be quoted.

XML

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Recipe>
  <RecipeInstance>
    <Variable name="MyProgram.MyINT">42</Variable>
    <Variable name="MyGlobalREAL">75.5</Variable>
    <Variable name="Test.Message">Testing</Variable>
    <Variable name="Test.Time">TIME#10.5s</Variable>
    <Variable name="MyProgram.Complex.Var[1]">17</Variable>
  </RecipeInstance>
</Recipe>
```

TOML

TOML

```
[[recipe_instance]]
[recipe_instance.data]
'MyProgram.MyINT' = 42
'MyGlobalREAL' = 75.5
'Test.Message' = 'Testing'
'Test.Time' = 'TIME#10.5s'
'MyProgram.Complex.Var[1]' = 17
```

5 Recipe Management

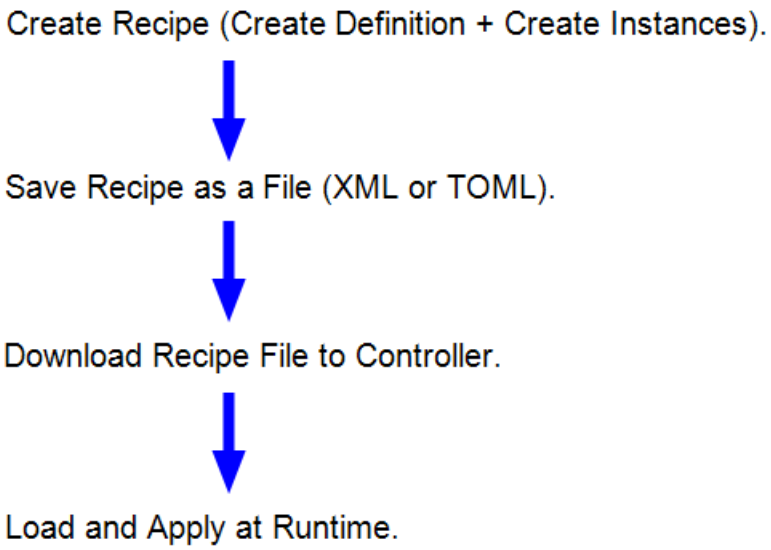
Recipe Management enables the creation, storage, transfer, and application of recipes in a controlled and repeatable way.

Recipe Management typically consists of:

- Engineering Workflow - Design time configuration and setup.
- Operational Workflow - runtime selection and execution.

5.0.1 Recipe Workflow

The typical lifecycle of a recipe is:



5.1 Create and Edit Recipe Files

Recipe files are used to define and store parameter sets outside the application logic.

This allows flexibility without modifying PLC code.

Procedure

NOTE

- Recipe Files can be created using any text editor (e.g., Notepad++, XmlNotepad, XMLBlueprint, TOML Editor for Visual Studio, etc.)
- It is recommended to use an editor that supports the file format you intend to use (E.g., XML or TOML).

1. Create a Recipe Definition:
   a. Define all required variables.
   b. Assign names, data types, and optional validation rules.
2. Add Recipe Instances:
   a. Provide values for each variable.
   b. Assign a name for the instance (recommended for traceability).
3. Save the file in a supported format.

Example Recipe File Structure

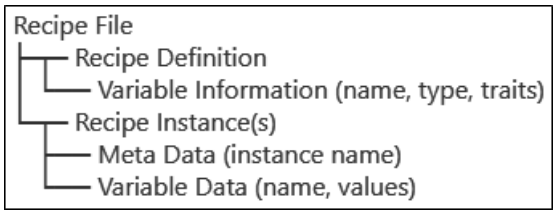


Figure 5-1: Recipe File Structure

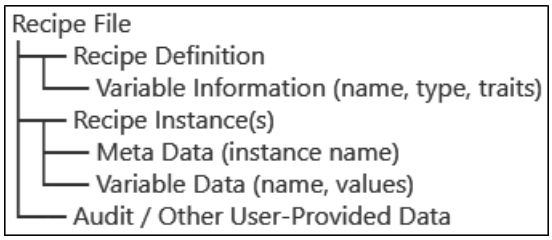


Figure 5-2: Recipe File Structure with Audit Data

- With a recipe set defined, individual recipe files are created in the editor.
- Existing recipe files can be duplicated as a starting point for new variants.
  - This is useful when recipes share most of their parameters and differ only in a few values.

5.2 Download Recipes to the Controller

Recipe files must be transferred to the controller before they can be used at runtime.

- Recipe files are stored in the controller's persistent file system (Internal Memory or USB flash drive) and are retained across power cycles.

Methods of file transfer:

- "Transfer via Controller Web Server" (→ p. 20).
- "Transfer via HTTP API" (→ p. 23).

5.2.1 Transfer via Controller Web Server

The User Data tab provides access to the User Data Storage area on the Controller's internal flash memory. (Figure 5-3)

- This area is intended for user-managed file storage to support PLC application needs.
- The system does not enforce structure or usage – it is entirely up to the user to organize and maintain the content.

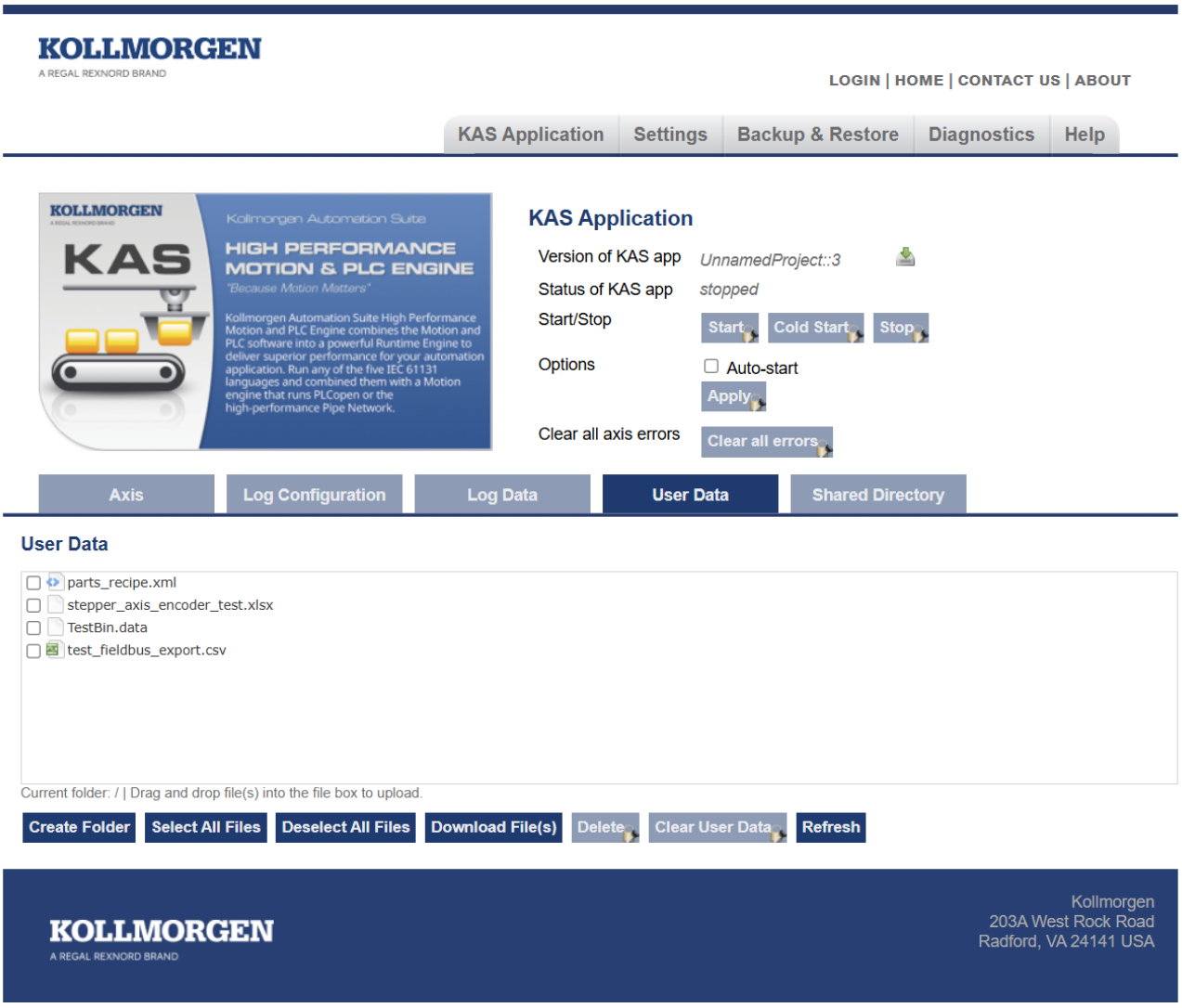


Figure 5-3: User Data tab

The User Data area is flexible, user-controlled storage space where you can maintain folders and files required for application management.

Typical use cases include:

- Storing and updating recipe files.
- Maintaining lookup tables or datasets.
- Exchanging data between external systems and the controller.

User Data File Explorer

- Display all folders and files currently stored in the User Data area on the controller flash.
- Each folder and file has a check box for selecting an action.

IMPORTANT

- Folders can only be deleted.

5.2.1.1 View File Contents from the User Data File Explorer

1. Double-click any file in the list.
   The file contents open in a separate tab. (Figure 5-4)

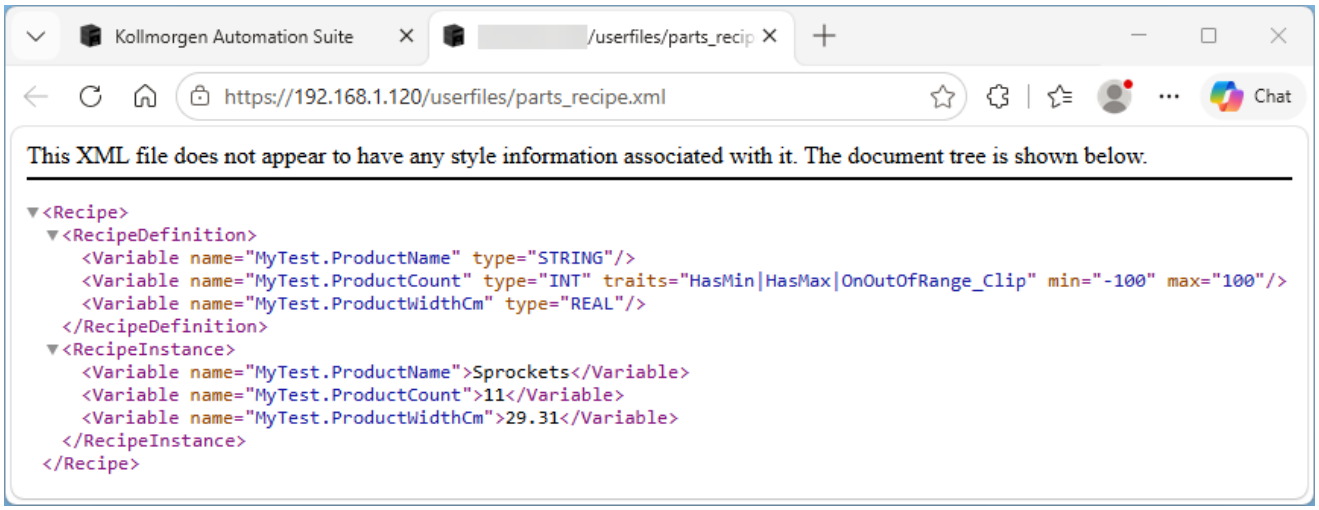


Figure 5-4: User Data tab - View File Contents

This new tab:

- Provides quick access for inspecting file data without downloading.
- Is useful for verifying configuration or recipe files.

5.2.1.2 Upload Folders and Files to the Controller

Folders and files can be uploaded to the PCMM2G controller flash using drag-and-drop.

1. On your computer, open Windows Explorer.
2. Locate and select the applicable folders and/or files.
3. Drag and drop the selected items into the User Data area.

   - Folders and files are stored immediately in the controller's User Data area.
   - Multiple folders and files can be uploaded at once.
   - There is no upload button – drag-and-drop is the only available method.

5.2.1.3 File Management Actions

All actions apply directly to the folders and files stored on the controller flash:

5.2.1.3.1 Create Folder

- Create a new folder in the current directory.

5.2.1.3.2 Select All Files

- Selects all files in the list.
- Primarily used to support bulk operations such as:
  - Downloading multiple files.
  - Deleting multiple files.

5.2.1.3.3 Deselect All Files

- Deselects all selected files.

5.2.1.3.4 Download Files

- Downloads selected files from the controller to your local computer.

IMPORTANT

- Downloading folders is not supported.

5.2.1.3.5 Delete

- Deletes the selected folders and files from the controller.

WARNING

- Before deleting folders and files, confirm they are no longer required by the application and create a backup if needed.
- Deleting folders and files cannot be undone.

5.2.1.3.6 Clear User Data

- Deletes all folders and files in the User Data area.

WARNING

- Use with caution - this removes all user-managed content.
- This action cannot be undone.

5.2.1.3.7 Refresh

- Updates the file explorer to reflect current contents on the controller.

5.2.2 Transfer via HTTP API

This is the process to manage the user data area on the controller flash memory from an external device (e.g., an HMI) using HTTP communications.

This method uses cURL (http://curl.haxx.se), a command line tool for performing HTTP communication with a server.

IMPORTANT

- The disk space available for the user data area in the controller flash is very limited.
  - It is strongly recommended to delete the files that are no longer necessary.
- The user files in the controller flash can be deleted using either HTTP commands or from the PLC application.
  - See File Management for available file function blocks.

Procedure

1. **List directories and files** present on the controller flash's user data folder:

```
curl.exe -k https://(IP_Address)/userfiles

Example Result:
{
   "Directories" : [
      {
         "Name" : "diagnostics"
      }
   ],
   "Files" : [
      {
         "Name" : "ScanTimes.csv"
      }
   ]
}
```

2. **Upload files** onto the controller flash's user data folder.
   Two HTTP requests are sent:
   - The first request is to authenticate.
   - The second request is to upload the file.

```
curl.exe -k -X POST -d "username=administrator" -d
"password=administrator" -G https://(IP_Address)/authorize/login -c
cookie.txt

curl.exe -k -F "sendfile=@C:\tmp\recipe.xml" -F "filename=recipe.xml"
https://(IP_Address)/userfiles -b cookie.txt
```

   - The first request is a POST used to provide credentials to the webserver.
     - The cookie received from the server is stored in a temporary **cookie.txt** file.
       - This is used in the second request.
     - The username, password, and IP address must be customized for your system.
   - The second request uploads the file using a HTTP Multipart file upload request.

- The `-b cookie.txt` at the end is used to add the authentication cookie received after the first request.
- The IP address, sendfile, and filename must be customized for your system.

NOTE

The first request to provide the credentials to the webserver is not required if the previously logged-in session is still active.

3. **Deleting files** from the controller flash's user data folder:

```
curl.exe -k -X POST -d "username=administrator" -d
"password=administrator" -G https://(IP_Address)/authorize/login -c
cookie.txt

curl.exe -k -X DELETE https://(IP_Address)/userfiles/recipe.txt -b
cookie.txt
```

TIP

- cURL exists as a C library.
- It can be used in compiled software which can be used in the HMI.
- If cURL is not an option, equivalent tools can be used if they can POST and PUT HTTP requests and support cookies.

5.3 Runtime-based Recipe Management through Function Blocks

For applications that require programmatic control over recipe operations, a set of function blocks are available for recipe management.

- These function blocks give the PLC program full control over recipe operations at runtime.
- Examples:
  - Automatic changeovers triggered by a HMI selection.
  - Barcode scan.
  - Manufacturing execution system integration.
  - Sequence-driven production.

Important Notes

- Always monitor these outputs from the function blocks:
  - Done
  - Busy
  - Error
- Reset the Execute input between operations.
- Do not trigger the next function until the previous one completes.

Function Block	Description
"RecipeApplyInstance" (→ p. 29)	Copy values to from a recipe instance to the program data.
"RecipeClose" (→ p. 31)	Close a recipe object.
"RecipeOpen" (→ p. 33)	Open a recipe file and create a recipe object.
"RecipeQuickLoad" (→ p. 35)	Load a recipe file and immediately apply the items it contains.
"RecipeSave" (→ p. 37)	Save a recipe object to a file.
"RecipeSaveRetainVars" (→ p. 39)	Save all retain variables, and their values, into a new recipe file.
"RecipeStoreInstance" (→ p. 41)	Store the present values of the program data in a recipe instance.

6 Recipe Design Example

This example is a beverage production line using a PLCopen-based motion controller.

The system requires specific parameters for different soda brands, which map to physical tasks like variable-speed mixing and high-speed bottling synchronization. This example examines different approaches to designing recipe files and discusses the advantages and disadvantages of each approach.

6.1 Variable Configurations

Instead of maintaining seven separate data structures for minor process steps, parameters are grouped into these core execution phases:

- "Example - Processing Parameters (Thermal and Mixing)" (→ p. 26)
- "Example - Carbonation Parameters (Pressure Controls)" (→ p. 26)
- "Example - Motion and Bottling Parameters (Cam Profiles and Packaging)" (→ p. 26)

6.2 Example - Processing Parameters (Thermal and Mixing)

PLC Variable	Data Type	FizzNova	BubbleRush	PrismPop
sugarKg	REAL	3.0	2.5	1.8
waterL	REAL	25.0	30.0	36.0
mixSpeedRPM	UINT	60	50	70
heatTempC	REAL	1200	1000	900

6.3 Example - Carbonation Parameters (Pressure Controls)

PLC Variable	Data Type	FizzNova	BubbleRush	PrismPop
chillTempC	REAL	2.0	3.0	1.5
co2PressureBar	REAL	3.5	4.0	3.0

6.4 Example - Motion and Bottling Parameters (Cam Profiles and Packaging)

PLC Variable	Data Type	FizzNova	BubbleRush	PrismPop
containerType	STRING	'Can330ml'	'Bottle500ml'	'Bottle2000ml'
fillSpeedBPM	UINT	180	150	170
packCount	UINT	6	12	1

6.5 Architectural Approaches and Tradeoffs

When deploying these parameters to a PLCopen controller, engineers must choose between a distributed or a centralized file structure.

- "Approach A: Multiple Modular Recipe Files" (➔ p. 27)
- "Approach B: Single Master Recipe" (➔ p. 27)

6.5.1 Approach A: Multiple Modular Recipe Files

Each manufacturing stage reads its own independent recipe file (e.g., processing.xml, carbonation.xml, bottling.xml).

How it Operates

The program invokes "RecipeApplyInstance" (➔ p. 29) independently at each stage of the physical line only when a new batch arrives at that zone.

Advantages

- **Asynchronous Execution**: The bottling line can pack FizzNova while the mixing tanks are already preparing the next batch of BubbleRush.
- **Memory Efficiency**: Lowers peak PLC controller memory overhead by only loading localized variables into active memory registers.

Disadvantages

- **File Sprawl**: Managing, validating, and auditing 3 to 7 separate file updates per SKU drastically increases HMI / SCADA synchronization overhead.

6.5.2 Approach B: Single Master Recipe

A single XML configuration file (e.g., master_recipes.xml) houses all parameter tables merged into one structural dataset per product variant / SKU.

How it Operates

A single call to RecipeApplyInstance downloads all processing, pressure, and motion variables simultaneously into global variables or system data blocks at the start of a production run.

Advantages

- **Simplified Auditing**: A single master file streamlines version control, recipe modification, and regulatory data tracking.
- **Atomic Updates**: Guarantees that all machine zones stay perfectly synchronized on the exact same product variant without risking a data mismatch between stages.

Disadvantages

- **Tracking Complexity**: Because all variables overwrite instantly at the beginning of the line, the PLC application logic must internally track and pass motion and bottling variables down the line along with the physical product.

7 Recipe Function Blocks

7.1 RecipeApplyInstance

This function/function block was added in KAS 5.00.

PLCopen ✓ Pipe Network ✓

Function Block - Copy values to from a recipe instance to the program data.

- Validation occurs before writing data to the program data.
  - This avoids partial writes when an error occurs.

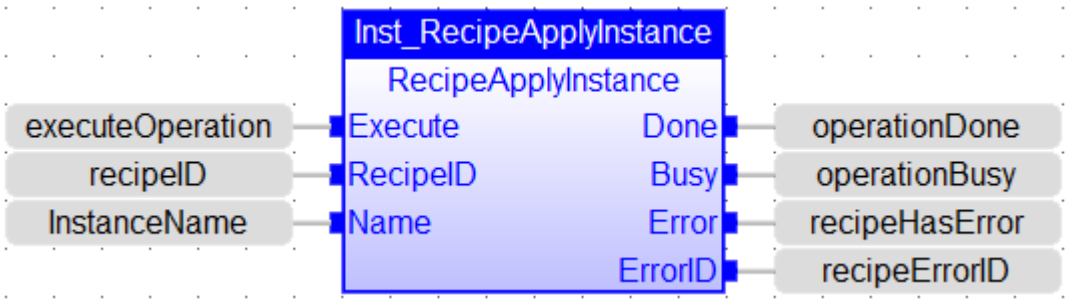
Inputs

Input	Data Type	Range	Unit	Default	Description
Execute	BOOL	FALSE, TRUE	N/A	No default	On the rising edge, copies the values of the specified instance into the program data.
RecipeID	UDINT	No range	N/A	No default	• The ID of the recipe object to use. • **RecipeID** is the handle ID returned by the "RecipeOpen" (→ p. 33) function block.
Name	STRING	No range	N/A	No default	• The name of the instance to apply. • If an empty string is supplied, the name "Instance" is used. • This is the same name applied to unnamed instances when a recipe is loaded from a file.

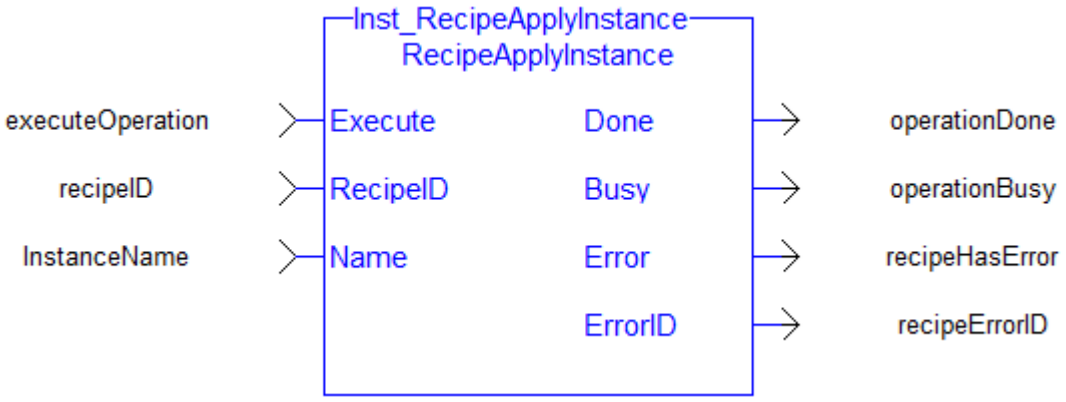
Outputs

Output	Data Type	Range	Unit	Description
Done	BOOL	FALSE, TRUE	N/A	If TRUE, the command completed successfully.
Busy	BOOL	FALSE, TRUE	N/A	If TRUE, the function block is executing.
Error	BOOL	FALSE, TRUE	N/A	If TRUE, an error has occurred.
ErrorID	DINT	Enumerated	N/A	• Indicates the error if **Error** output is TRUE. • See the table in "File, Recipe, and TCP/IP Function Block ErrorIDs" (→ p. 43).

FBD Language Example



FFLD Language Example



IL Language Example

Not available.

ST Language Example

```
Inst_RecipeApplyInstance(True, recipeID, 'Widgets');
IF Inst_RecipeApplyInstance.Error THEN
   errorMessage := MC_ErrorDescription(Inst_RecipeApplyInstance.ErrorID);
END_IF;
```

7.2 RecipeClose

This function/function block was added in KAS 5.00.

PLCopen ✓ Pipe Network ✓

Function Block - Close a recipe object.

RecipeClose unloads a recipe object from the actively running program.

- **RecipeClose** does **not** delete the recipe file used to create the recipe object.
  - To delete a recipe file, use the FileDelete function block.
- **RecipeClose** does **not** save changes made to the recipe object.
- If instances were modified via "RecipeStoreInstance" (→ p. 41), call "RecipeSave" (→ p. 37) before to persist the changes to the file system.

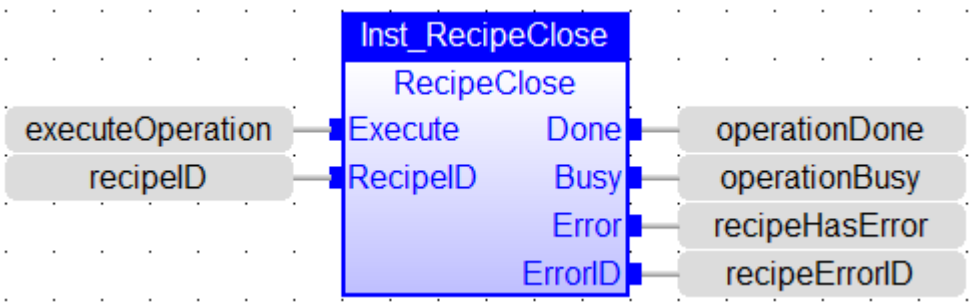
Inputs

Input	Data Type	Range	Unit	Default	Description
Execute	BOOL	FALSE, TRUE	N/A	No default	On the rising edge, close the specified recipe object.
RecipeID	UDINT	No range	N/A	No default	• The ID of the recipe object to close. • **RecipeID** is the handle ID returned by the "RecipeOpen" (→ p. 33) function block.

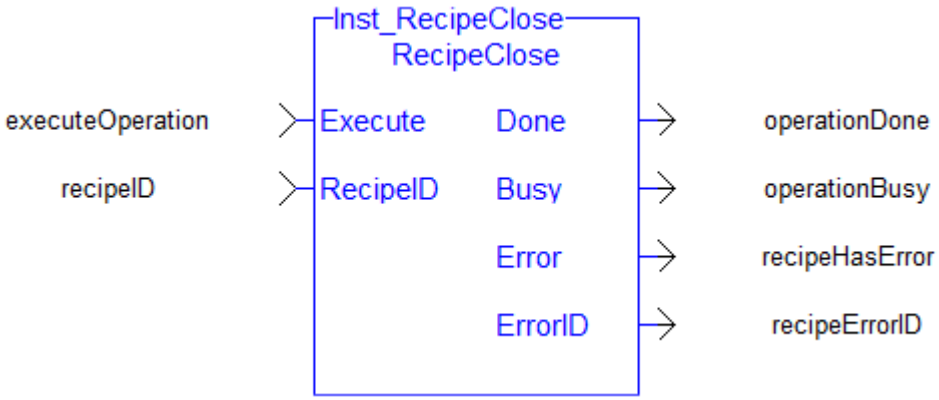
Outputs

Output	Data Type	Range	Unit	Description
Done	BOOL	FALSE, TRUE	N/A	If TRUE, the command completed successfully.
Busy	BOOL	FALSE, TRUE	N/A	If TRUE, the function block is executing.
Error	BOOL	FALSE, TRUE	N/A	If TRUE, an error has occurred.
ErrorID	DINT	Enumerated	N/A	• Indicates the error if **Error** output is TRUE. • See the table in "File, Recipe, and TCP/IP Function Block ErrorIDs" (→ p. 43).

FBD Language Example



FFLD Language Example



IL Language Example

Not available.

ST Language Example

```
Inst_RecipeClose(True, recipeID);
IF Inst_RecipeClose.Error THEN
   errorMessage := MC_ErrorDescription(Inst_RecipeClose.ErrorID);
END_IF;
```

7.3 RecipeOpen

This function/function block was added in KAS 5.00.

Function Block - Open a recipe file and create a recipe object.

Recipe files opened with RecipeOpen must include a recipe definition section.

- Instance sections are optional.
- Recipes without instance sections can be opened and used later with "RecipeStoreInstance" (→ p. 41) to create instances from active program data.

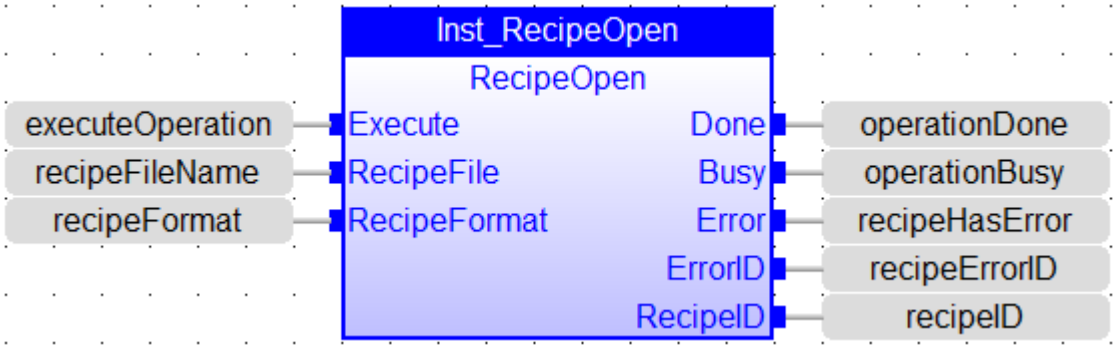
Inputs

Input	Data Type	Range	Unit	Default	Description
Execute	BOOL	FALSE, TRUE	N/A	No default	On the rising edge, load a recipe file.
RecipeFile	STRING	No range	N/A	No default	The location of the recipe file.
RecipeFormat	SINT	Enumerated	N/A	No default	• The recipe file format. • Valid values include one of these enumeration values: **Value** / **Description** RCP_FMT_XML — XML file format. RCP_FMT_TOML — TOML file format.

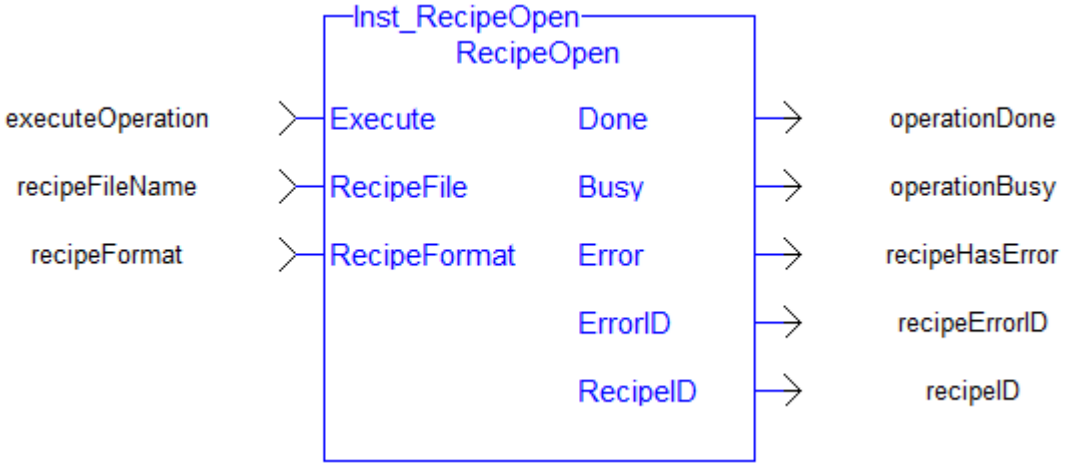
Outputs

Output	Data Type	Range	Unit	Description
Done	BOOL	FALSE, TRUE	N/A	If TRUE, the command completed successfully.
Busy	BOOL	FALSE, TRUE	N/A	If TRUE, the function block is executing.
Error	BOOL	FALSE, TRUE	N/A	If TRUE, an error has occurred.
ErrorID	DINT	Enumerated	N/A	• Indicates the error if **Error** output is TRUE. • See the table in "File, Recipe, and TCP/IP Function Block ErrorIDs" (→ p. 43).
RecipeID	UDINT	No range	N/A	The ID of the recipe object.

FBD Language Example



FFLD Language Example



IL Language Example

Not available.

ST Language Example

```
Inst_RecipeOpen(True, 'Parts.xml', RCP_FMT_XML);
IF Inst_RecipeOpen.Error THEN
   errorMessage := MC_ErrorDescription(Inst_RecipeOpen.ErrorID);
ELSE
   recipeID = Inst_RecipeOpen.RecipeID
END_IF;
//
Inst_RecipeOpen(True, '/usbflash/recipes/Parts.xml', RCP_FMT_XML);
IF Inst_RecipeOpen.Error THEN
   errorMessage := MC_ErrorDescription(Inst_RecipeOpen.ErrorID);
ELSE
   recipeID = Inst_RecipeOpen.RecipeID
END_IF;
```

7.4 RecipeQuickLoad

This function/function block was added in KAS 5.00.

Function Block - Load a recipe file and immediately apply the items it contains.

RecipeQuickLoad loads a recipe instance without requiring that a recipe definition be loaded first.

- Only the first recipe instance found in the file is applied.
- When ContinueOnError is TRUE, all validation errors encountered are added to the controller log.
  - See Log Messages tabs.

Inputs

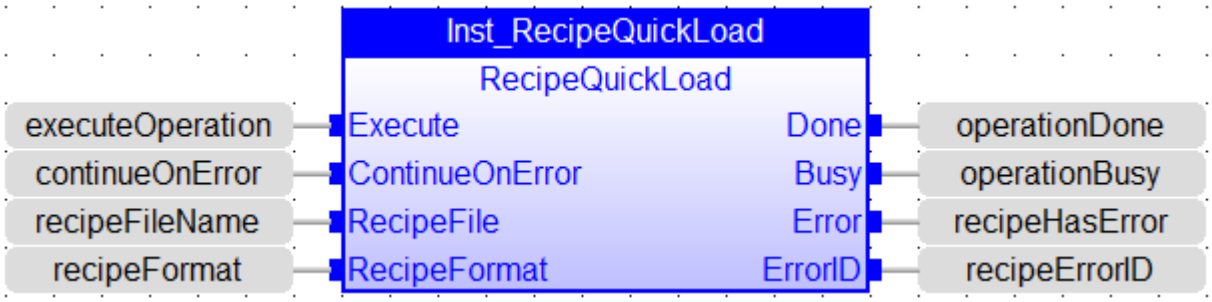
Input	Data Type	Range	Unit	Default	Description
Execute	BOOL	FALSE, TRUE	N/A	No default	On the rising edge, request loading and applying the recipe stored in the file RecipeFile.
ContinueOnError	BOOL	FALSE, TRUE	N/A	No default	• If FALSE, no recipe items are applied if an error occurs. • If TRUE, an error does not halt application of recipe items. • All recipe items that can be applied are applied.
RecipeFile	STRING	No range	N/A	No default	The location of the recipe file.
RecipeFormat	SINT	Enumerated	N/A	No default	• The recipe file format. • Valid values include one of these enumeration values: **Value** / **Description** RCP_FMT_XML — XML file format. RCP_FMT_TOML — TOML file format.

Outputs

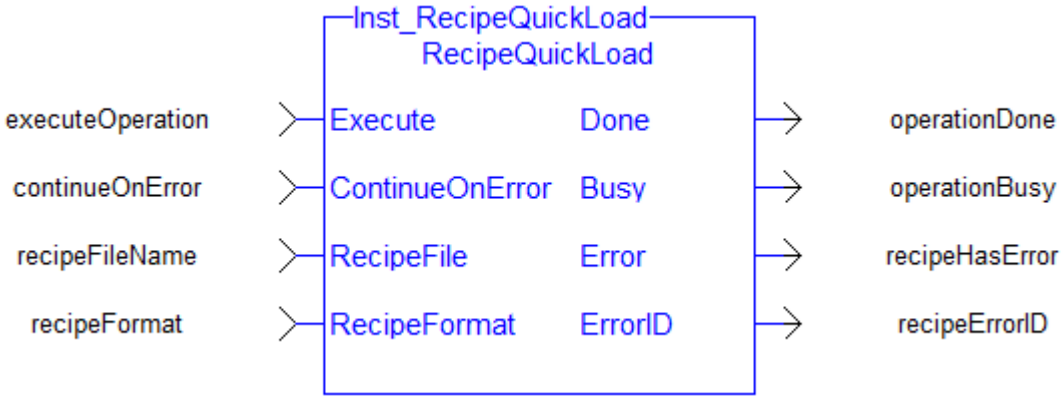
Output	Data Type	Range	Unit	Description
Done	BOOL	FALSE, TRUE	N/A	If TRUE, the command completed successfully.
Busy	BOOL	FALSE, TRUE	N/A	If TRUE, the function block is executing.
Error	BOOL	FALSE, TRUE	N/A	If TRUE, an error has occurred.

Output	Data Type	Range	Unit	Description
ErrorID	DINT	Enumerated	N/A	• Indicates the error if **Error** output is TRUE. • See the table in "File, Recipe, and TCP/IP Function Block ErrorIDs" (→ p. 43).

FBD Language Example



FFLD Language Example



IL Language Example

Not available.

ST Language Example

```
Inst_RecipeQuickLoad(True, False, 'CogsRecipeInst.xml', RCP_FMT_XML);
IF Inst_RecipeQuickLoad.Error THEN
   errorMessage := MC_ErrorDescription(Inst_RecipeQuickLoad.ErrorID);
END_IF;
```

7.5 RecipeSave

This function/function block was added in KAS 5.00.

Function Block - Save a recipe object to a file.

- **RecipeSave** writes the recipe object (definition + stored instances) to the filesystem.
- It does not read live PLC values.
- To capture PLC values before saving, call "RecipeStoreInstance" (→ p. 41) beforehand.

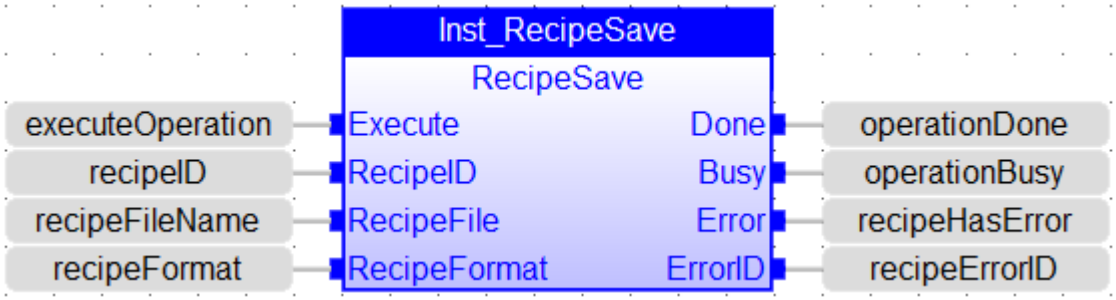
Inputs

Input	Data Type	Range	Unit	Default	Description
Execute	BOOL	FALSE, TRUE	N/A	No default	On the rising edge, save the recipe to the file RecipeFile.
RecipeID	UDINT	No range	N/A	No default	• The ID of the recipe object to use when saving a recipe. • **RecipeID** is the handle ID returned by the "RecipeOpen" (→ p. 33) function block.
RecipeFile	STRING	No range	N/A	No default	The path of the destination recipe file.
RecipeFormat	SINT	Enumerated	N/A	No default	• The recipe file format. • Valid values include one of these enumeration values: **Value** / **Description** RCP_FMT_XML — XML file format. RCP_FMT_TOML — TOML file format.

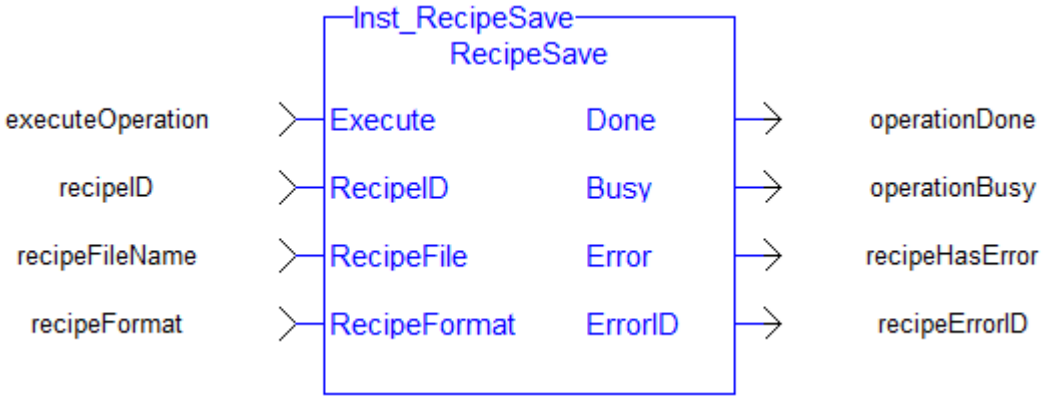
Outputs

Output	Data Type	Range	Unit	Description
Done	BOOL	FALSE, TRUE	N/A	If TRUE, the command completed successfully.
Busy	BOOL	FALSE, TRUE	N/A	If TRUE, the function block is executing.
Error	BOOL	FALSE, TRUE	N/A	If TRUE, an error has occurred.
ErrorID	DINT	Enumerated	N/A	• Indicates the error if **Error** output is TRUE. • See the table in "File, Recipe, and TCP/IP Function Block ErrorIDs" (→ p. 43).

FBD Language Example



FFLD Language Example



IL Language Example

Not available.

ST Language Example

```
Inst_RecipeSave(True, recipeID, 'Parts_Recipes.xml', RCP_FMT_XML);
IF Inst_RecipeSave.Error THEN
   errorMessage := MC_ErrorDescription(Inst_RecipeSave.ErrorID);
END_IF;
```

7.6 RecipeSaveRetainVars

This function/function block was added in KAS 5.00.

Function Block - Save all retain variables, and their values, into a new recipe file.

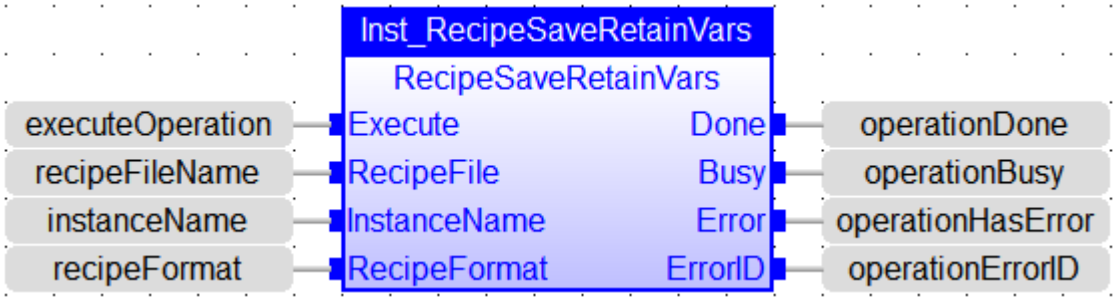
Inputs

Input	Data Type	Range	Unit	Default	Description
Execute	BOOL	FALSE, TRUE	N/A	No default	On the rising edge, save the retain variables, and their values, to a new recipe file.
RecipeFile	STRING	No range	N/A	No default	• The path of the destination recipe file. • If the file with the same name already exists, it is overwritten.
InstanceName	STRING	No range	N/A	No default	• The name of the new instance. • If InstanceName is an empty string, the instance is assigned the name **Instance**.
RecipeFormat	SINT	Enumerated	N/A	No default	• The recipe file format. • Valid values include one of these enumeration values: **Value** / **Description** RCP_FMT_XML — XML file format. RCP_FMT_TOML — TOML file format.

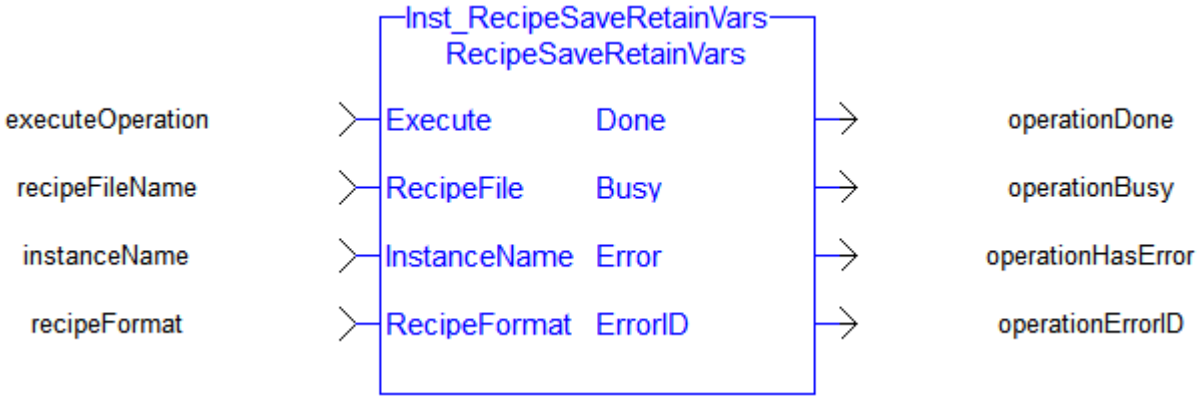
Outputs

Output	Data Type	Range	Unit	Description
Done	BOOL	FALSE, TRUE	N/A	If TRUE, the command completed successfully.
Busy	BOOL	FALSE, TRUE	N/A	If TRUE, the function block is executing.
Error	BOOL	FALSE, TRUE	N/A	If TRUE, an error has occurred.
ErrorID	DINT	Enumerated	N/A	• Indicates the error if **Error** output is TRUE. • See the table in "File, Recipe, and TCP/IP Function Block ErrorIDs" (→ p. 43).

FBD Language Example



FFLD Language Example



IL Language Example

Not available.

ST Language Example

```
Inst_RecipeSaveRetainVars(True, 'RetainVars.xml', 'Retain', RCP_FMT_XML);
IF Inst_RecipeSaveRetainVars.Error THEN
   errorMessage := MC_ErrorDescription(Inst_RecipeSaveRetainVars.ErrorID);
END_IF;
```

7.7 RecipeStoreInstance

This function/function block was added in KAS 5.00.

PLCopen ✓ Pipe Network ✓

Function Block - Store the present values of the program data in a recipe instance.

- If an instance with the specified name already exists, the values in that instance are overwritten.
- If no instance exists with the specified name, a new instance with the specified name is created.
- No range validation is performed while storing.
  - Validation is deferred until a later call to RecipeApplyInstance.

Inputs

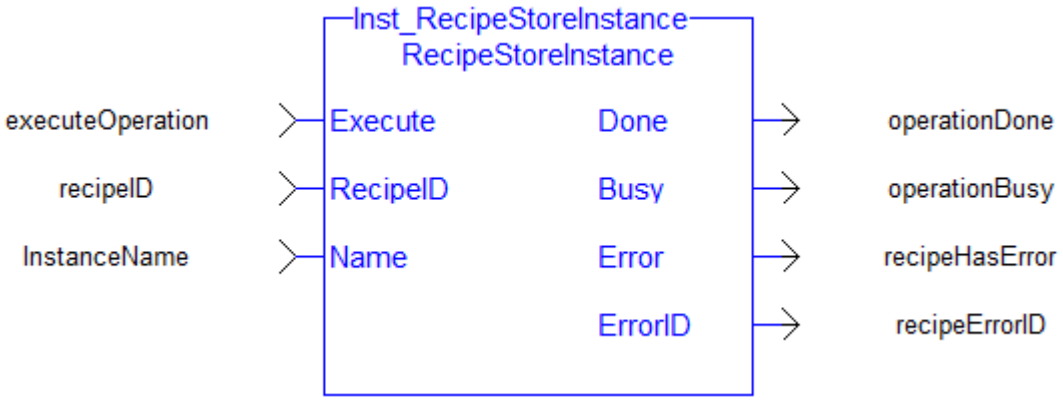
Input	Data Type	Range	Unit	Default	Description
Execute	BOOL	FALSE, TRUE	N/A	No default	On the rising edge, save the program data into the named recipe instance.
RecipeID	UDINT	No range	N/A	No default	• The ID of the recipe object to use. • **RecipeID** is the handle ID returned by the "RecipeOpen" (→ p. 33) function block.
Name	STRING	No range	N/A	No default	• The name of the new instance. • If Name is an empty string, the instance is assigned the name 'Instance'.

Outputs

Output	Data Type	Range	Unit	Description
Done	BOOL	FALSE, TRUE	N/A	If TRUE, the command completed successfully.
Busy	BOOL	FALSE, TRUE	N/A	If TRUE, the function block is executing.
Error	BOOL	FALSE, TRUE	N/A	If TRUE, an error has occurred.
ErrorID	DINT	Enumerated	N/A	• Indicates the error if **Error** output is TRUE. • See the table in "File, Recipe, and TCP/IP Function Block ErrorIDs" (→ p. 43).

FBD Language Example

FFLD Language Example



IL Language Example

Not available.

ST Language Example

```
Inst_RecipeStoreInstance(True, recipeID, 'Cogs');
IF Inst_RecipeStoreInstance.Error THEN
   errorMessage := MC_ErrorDescription(Inst_RecipeStoreInstance.ErrorID);
END_IF;
```

8 File, Recipe, and TCP/IP Function Block ErrorIDs

This is the list of possible errors that could be returned at the **ErrorID** output of the File Function Blocks, Recipe Function Blocks, and TCP/IP Function Blocks.

8.1 File ErrorIDs

ErrorID	Description
16000	Error opening file.
16001	All PLC file handles used.
16002	File ID is invalid.
16003	File ID has been closed.
16004	Error in file stream.
16005	End of file encountered.
16006	Internal file FB error.
16007	Unknown file error.
16008	No such file or directory.
16009	Bad file descriptor.
16010	File already exists.
16011	Too many open files in the system.
16012	Text file is busy.
16013	File is too large.
16014	File system is read only.
16015	File locking deadlock.
16016	File name is too long.
16017	File positioning error.
16018	Bad or corrupted file system detected .

8.2 Recipe ErrorIDs

ErrorID	Description
16008	No such file or directory.
16019	File inaccessible. (e.g., permission, locks)
16020	Error opening the file for writing.
16200	Error parsing recipe file.
16201	Invalid recipe file format.
16202	Recipe variable not found.
16203	Recipe variable type mismatch.
16204	Recipe variable value out of range.
16205	Invalid recipe file format.
16206	Recipe variable not found.
16207	Duplicate recipe instance name.
16208	Invalid recipe variable definition.
16219	Internal recipe FB error.

8.3 TCP/IP ErrorIDs

ErrorID	Description
16100	Connection error.
16101	TCP ID has been closed.
16102	All PLC TCP handles used.
16103	TCP ID is invalid.
16104	Failed to allocate TCP/IP socket.
16105	TCP operation internal error.

9 Data Types

Data types are defined within the common elements of IEC 61131-3.

Why Data Typing?

Data typing is implemented to define the type of any parameter used. This helps prevent errors early on in the programming phase (e.g., avoids dividing a Date by an Integer).

When you have defined whether the data is a string, a date, an integer or a 16-bit Boolean input, there is no longer any confusion, nor any conflict between different people using the textual representation (i.e., the name of the variable).

9.1 Different Data Types

- **Common Data Types**: Boolean, Byte, Date, Integer, Real, String, Time_of_Day, Word.
- **Derived Data Types**: Define personal data types based on the Common data types.
  - Example: Define an analog input channel as a data type and re-use it.

9.2 List of Data Types

Type	Prefix	Description	Values
BOOL		Boolean (bit)	• FALSE, TRUE • Stored in 1 byte.
BYTE		Same as "USINT" (→ p. 46).	
DINT		Signed double precision integer in 32-bits.	-2147483648 to 2147483647
DWORD		Same as "UDINT" (→ p. 45).	
INT	INT#	Signed integer in 16-bits.	-32768 to +32767
LINT	LINT#	Long signed integer in 64-bits.	
LREAL ‡	LREAL#	Double precision floating point stored in 64-bits.	• -1.7E308 to 1.7E308 • 14 to 15 significant digits of accuracy
LWORD		Same as "ULINT" (→ p. 45).	
REAL ‡	REAL# ‡‡	Single precision floating point stored in 32-bits.	• -3.4E38 to 3.4E38 • 6 to 7 significant digits of accuracy.
SINT	SINT#	Small signed integer in 8-bits.	-128 to +127
STRING		• Variable length string with declared maximum length. • Each character is store on 1 byte (i.e., on 8-bits).	Maximum length cannot exceed 255 characters.
TIME	T# or TIME#	Time data type is used to specify a time variable. • Accuracy is 1ms.	0ms - 24hr
UDINT	UDINT#	Unsigned integer in 32-bits.	0 to +4294967295
UINT	UINT#	Unsigned integer in 16-bits.	0 to 65535
ULINT	ULINT#	Long unsigned integer in 64-bits.	

Type	Prefix	Description	Values
USINT	USINT#	Small unsigned integer in 8-bits.	0 to 255
WORD		Same as "UINT" (→ p. 45).	
WSTRING		• Do not use! • The WSTRING is **not** supported.	

NOTE

- ‡ **REAL** variables are limited to 6 digits of accuracy.
  - To achieve greater accuracy, a longer mantissa may be specified by prefixing **LREAL** with #.
- Example: To achieve an accuracy of 20 digits for the value of Pi, rather than what **REAL** provides (3.14159), set the type to `LREAL#3.14159265358979238`.

IMPORTANT

‡‡ **REAL** is restrictive, but because it is the default, it is recommended to explicitly declare real constants with the **LREAL#** prefix.

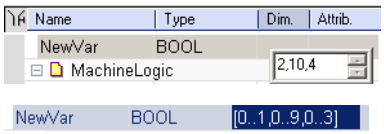
TIP

Use **2#**, **8#**, or **16#** prefixes to specify an integer in binary, octal or hexadecimal basis, respectively.

9.3 Arrays

You can declare arrays for internal variables by specifying dimension(s).

To declare an array, enter the number of elements in the **Dim.** column on the Variables tab.



For a multi-dimensional array (arrays have a maximum of three dimensions), enter the number of elements for each dimension separated by commas:

- Using **2,10,4** as the 3D array:
  - The first dimension has 2 elements.
  - The second dimension has 10 elements.
  - The third dimension has 4 elements.

Arrays are accessed by variable indices.

```
myArray[myIndex1, myIndex2]
```

IMPORTANT

- All indexes are 0 based.
- Example: In a single dimension array, the first element is always identified by `ArrayName`**[0]**.

9.4 Bitfields

A bitfield is a user-defined data type that combines multiple Boolean or integer values into a single variable.

- Bit Fields are used to define custom variable types and values.
- A bitfield packs multiple pieces of data together in one variable.
- Each field represents an independent piece of information.
  - These fields can be referenced individually within the application.
- Bitfields provide a compact and organized way to group related status and control information while maintaining easy access to individual fields.

Example

A bitfield type named AxisStatus may contain these fields:

Initialized
Enabled
Moving
Faulted

See the Bit Fields tab about creating bit fields and examples of their use.

9.5 Enumerations

An enumeration (enum) is a user-defined data type that contains a predefined set of named values.

- An enum variable can hold only one value from the defined list at a time.
- Enumerations are useful when a variable should only represent one value from a known set of states, modes, or selections.

Example

Enumeration can be used to represent machine states such as:

Idle
Homing
Running
Faulted

- Using enumerations improves program readability and maintainability by replacing numerical values with meaningful names.
- This makes application logic easier to understand, debug, and maintain.
- See the Enums tab about creating enumerations and examples of their use.

9.6 Structure

A structure is a complex data type defined as a set of members.

- Members of a structure can have various data types.
  - A member of a structure can have dimensions or can be an instance of another structure.
- When a structure is defined, it can be used like other data types to declare variables.
- Members of a structure can have an initial value.
  - Corresponding members of all declared variables having this structure type are initialized with the initial value of the member.
- To specify a member of a structured variable in PLC languages, use this notation:

```
VariableName.MemberName
```

9.6.1 Limitation

If a member of a structure is an instance of another structure, the nested structure must be declared **BEFORE** in the list.

10 Recipe Troubleshooting

- When a recipe function block fails, check the value of the function block ErrorID output against the table in "File, Recipe, and TCP/IP Function Block ErrorIDs" (➜ p. 43).
- The specific error code is used to determine the proper corrective action.

Issue	Possible Cause	Recommended Action
Recipe file does not load.	• Invalid format. • Missing file. • Incorrect file path.	• Check the function block ErrorID output. • Verify the file exists in the expected controller location. • Confirm the selected format matches the file contents.
Recipe instance is not applied.	• The instance name does not match. • The file does not contain the expected instance.	• Check the function block ErrorID output. • Verify the instance name.
Values are not written to program data.	• Variable names do not match PLC program variables. • Variable values are invalid.	• Check the function block ErrorID output. • Compare each recipe variable name against the PLC application variable name. • Verify the variable values are in the specified ranges and contain no typos.

11 Kollmorgen Documentation Legal Information

The information in these pages is only for the technical documentation (e.g., Safety Notes, User Manuals, etc.) legal information.

This information does not represent Kollmorgen company-wide legal information.

11.1 KAS - Copyrights, Trademarks, Patents, Licenses, and Disclaimers

11.2 Kollmorgen Trademarks

Regal Rexnord and Kollmorgen are trademarks of Regal Rexnord Corporation or one of its affiliated companies.

These trademarks or trade names are owned by or under the control of Regal Rexnord Corporation or any of its affiliates:

- Kollmorgen Essentials System
- Kollmorgen Essentials Cable
- Kollmorgen Essentials Drive
- Kollmorgen Essentials Motor
- AKD
- AKD2G
- AKD PDMM
- AKM
- AKMA
- AKME
- AKMH
- BLM
- Cartridge DDR
- DDL
- DDR
- Goldline
- Kollmorgen Automation Suite
- KAS
- Kollmorgen Visualization Builder
- KVB
- MKD
- Motioneering
- P8000
- PCMM
- PCMM2G
- PowerMax
- Servostar
- SafeMotion Monitor
- SMM
- TBM
- TBM2G
- WorkBench

11.3 Kollmorgen Patents

Any of these Kollmorgen patents may be used and/or identified in Kollmorgen products, firmware, and/or software.

- US Patent 2017/0211640 (method and apparatus for power saving, fail-safe control of an electromechanical brake), patent pending.
- US Patent 8,154,228 (Dynamic Braking For Electric Motors).
- US Patent 8,214,063 (Auto-tune of a Control System Based on Frequency Response).
- US Patent 8,566.415 (Safe Torque Off over network wiring).
- US Patent 10,374,468 (System and method for improved DC power line communication).
- US Patent 10,520,050 (Method and apparatus for power-saving, fail-safe control of an electromechanical brake).
- US Patent 10,840,696 (Method and apparatus for limiting the output voltages of switch mode power supplies).
- US Patent 16,247,478 (method and apparatus for limiting the output voltages of switching mode power supplies), patent pending.

Patents referring to fieldbus functions are listed in the matching fieldbus manual.

11.4 Other Trademarks

Any of these other trademarks and/or trade names may be used and/or identified in Kollmorgen products, firmware, and/or software.

These are believed to be the trademarks and/or trade names of their respective owners and are not owned or controlled by Regal Rexnord Corporation:

- BiSS: iC-Haus GmbH.
- CANopen: CAN in Automation (CiA).
- DRIVE-CLiQ: Siemens Aktiengesellschaft.
- EnDat: Dr. Johannes Heidenhain GmbH.
- EtherCAT und Safety over EtherCAT: Beckhoff Automation GmbH.
- EtherCAT: Beckhoff Automation GmbH.
- EtherNet/IP Kommunikationsstack: Rockwell Automation.
- EtherNet/IP: ODVA, Inc.
- Ghostscript: Artifex Software, Inc. (unter der AGPL Lizenz verbreitet).
- HIPERFACE DSL: SICK AG.
- HIPERFACE: SICK AG.
- HUMMEL: HUMMEL AG.
- Hysol: Hysol Huawei Electronics Co,.LTD.
- Instapak: Sealed Air Corporation.
- INTERCONTEC: INTERCONTEC Pfeiffer Industrie-Steckverbindungen GmbH.
- LOCTITE 640: Henkel AG & CO. KGAA.
- LOCTITE: Henkel AG & CO. KGAA.
- Modbus: Schneider Electric USA, Inc.
- MOLYKOTE: DDP Specialty Electronic Materials US 9, LLC.
- Mylar: DUPONT Teijin Films U.S.
- P3-topax: Henkel KGaA.
- PHOENIX CONTACT: Phoenix Contact GmbH & Co. KG.
- PLCopen: An association registered with the Chamber of Commerce in The Netherlands (Reg. 40240111).
- PROFINET: PROFIBUS Nutzerorganisation e.V.
- Scotch-Weld 2214: 3M Company.
- sercos: Sercos International E.V.
- SIMATIC: Siemens Aktiengesellschaft.
- SpeedTec: Liqui-Moly GmbH or TE Connectivity Industrial GmbH.
  If SpeedTec is used as a lubricant, it belongs to Liqui-Moly GmbH.
  If SpeedTec is used as a connector, it belongs to TE Connectivity Industrial GmbH
- Teflon: The Chemours Company FC, LLC.
- Viton: The Chemours Company FC, LLC.
- Windows: Microsoft Corporation.

11.5 3rd-party Licenses

Any of these 3rd-party licenses may be used and/or identified in Kollmorgen products, firmware, and/or software.

Library	Product	License
7-zip	https://www.7-zip.org/	7-zip - License
Adafruit core graphics library	https://github.com/adafruit	https://github.com/adafruit/Adafruit-GFX-Library/blob/master/license.txt
Adafruit display library	https://github.com/adafruit	https://github.com/adafruit/Adafruit-ST7735-Library/blob/master/README.txt
Arduino Map function	http://creativecommons.org	http://creativecommons.org/licenses/by-sa/3.0/
C++ Mathematical Expression Library	http://www.partow.net/programming/exprtk/index.html	MIT License
civetweb: Embedded C/C++ web server	https://github.com/civetweb/civetweb	MIT License
CoreSight Library	http://www.apache.org	http://www.apache.org/licenses/LICENSE-2.0
curl software library		curl license
DockPanelSuite	http://sourceforge.net/projects/dockpanelsuite/	https://opensource.org/licenses/MIT
EtherNet/IP Communication Stack		Software Distribution License
FatFs	https://github.com/abbrev/fatfs	https://github.com/stm32duino/FatFs/blob/master/Licence.md
FileHelpers	http://sourceforge.net/projects/filehelpers	https://opensource.org/licenses/MIT
Font Awesome	https://fontawesome.com	https://fontawesome.com/license/free
Ghostscript	https://ghostscript.com/index.html	AGPL
Google Fonts Roboto Condensed	http://www.apache.org	http://www.apache.org/licenses/
GSL - Guideline Support Library	https://github.com/Microsoft/GSL	https://github.com/Microsoft/GSL/blob/master/LICENSE

Library	Product	License
libsodium	https://libsodium.org	https://github.com/jedisct1/libsodium/blob/master/LICENSE
jsoncpp software		jsoncpp License
log4net	http://logging.apache.org/log4net/	http://www.apache.org/licenses/LICENSE-2.0
LVGL	https://github.com/lvgl/lvgl	https://github.com/lvgl/lvgl/blob/master/LICENCE.txt
LwIP	https://savannah.nongnu.org	https://savannah.nongnu.org/projects/lwip/
mbed Microcontroller Library	http://www.apache.org	http://www.apache.org/licenses/LICENSE-2.0
mbed TLS	http://www.apache.org	http://www.apache.org/licenses/LICENSE-2.0
Microsoft Edge WebView2	https://developer.microsoft.com/en-us/microsoft-edge/webview2	Microsoft Edge WebView2 Runtime LICENSE
MigraDoc	http://www.pdfsharp.net/migradocoverview.ashx	http://www.pdfsharp.net/MigraDoc_License.ashx
Mongoose Library	https://github.com/cesanta/mongoose/tree/	Mongoose license
MVVM Light Toolkit	https://github.com/lbugnion/mvvmlight	https://galasoft.ch/license_MIT.txt
OpENer EtherNet/IP	https://github.com/EIPStackGroup/OpENer	http://eipstackgroup.github.io/OpENer/de/d7e/license.html
PdfSharp	http://www.pdfsharp.net/migradocoverview.ashx	http://www.pdfsharp.net/PDFsharp_License.ashx
Qt Framework	https://www.qt.io/product/framework	terms of the LGPL3
Qwt project	http://qwt.sourceforge.net/	Qwt
SharpZipLib	https://github.com/icsharpcode/SharpZipLib	https://github.com/icsharpcode/SharpZipLib/blob/master/LICENSE.txt
SNMP		http://www.net-snmp.org/about/license.html
SQLite	https://system.data.sqlite.org	https://system.data.sqlite.org/index.html/doc/trunk/www/copyright.wiki
U-Boot	http://www.denx.de/wiki/U-Boot	GNU General Public License 2.0
Zed Graph	http://sourceforge.net/projects/zedgraph/	https://opensource.org/licenses/LGPL-2.0
Zlib software library	https://www.zlib.net/	Zlib License

11.6 Disclaimers

Technical changes which improve the performance of the device may be made without prior notice!

This document is the intellectual property of Kollmorgen. All rights reserved. No part of this work may be reproduced in any form (by photocopying, microfilm or any other method) or stored, processed, copied or distributed by electronic means without the written permission of Kollmorgen.

The information in this document is believed to be accurate and reliable at the time of its release. Kollmorgen assumes no responsibility for any damage or loss resulting from the use of this help, and expressly disclaims any liability or damages for loss of data, loss of use, and property damage of any kind, direct, incidental or consequential, in regard to or arising out of the performance or form of the materials presented herein or in any software programs that accompany this document.

All timing diagrams, whether produced by Kollmorgen or included by courtesy of the PLCopen organization, are provided with accuracy on a best-effort basis with no warranty, explicit or implied, by Kollmorgen. The user releases Kollmorgen from any liability arising out of the use of these timing diagrams.

Technische Änderungen, die der Verbesserung der Geräte dienen, vorbehalten!

Dieses Dokument ist geistiges Eigentum von Kollmorgen. Alle Rechte vorbehalten. Kein Teil dieses Werkes darf in irgendeiner Form (Fotokopie, Mikrofilm oder in einem anderen Verfahren) ohne schriftliche Genehmigung von Kollmorgen reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Die Genauigkeit und Verlässlichkeit der Informationen in diesem Dokument entspricht dem Kenntnisstand zum Zeitpunkt der Veröffentlichung. Kollmorgen übernimmt keine Verantwortung für Schäden oder Verluste, die sich aus der Verwendung dieser Hilfe ergeben, und lehnt ausdrücklich jegliche Haftung sowie Schadensersatzansprüche für Datenverlust, entgangene Nutzung und Sachschäden jeglicher Art ab, inklusive direkter, zufälliger oder Folgeschäden im Zusammenhang mit der Funktion oder Form des hierin oder in einem der mit diesem Dokument gelieferten Software enthaltenen Materials.

Es posible que se implementen sin aviso previo cambios técnicos que mejoran el rendimiento del dispositivo.

La propiedad intelectual de este documento pertenece a Kollmorgen. Todos los derechos reservados. Se prohíbe la reproducción de este documento por ningún medio (fotocopia, microfilm u otro método), así como su almacenamiento, procesamiento, copia o distribución por medios electrónicos, sin el consentimiento por escrito de Kollmorgen.

La información incluida en este documento se considera correcta y confiable en el momento de su publicación. Kollmorgen no asume ninguna responsabilidad por ningún daño o pérdida causados por el uso de este documento de ayuda y rechaza explícitamente toda responsabilidad o daño por pérdida de datos, pérdida de uso y daños en la propiedad de cualquier tipo, ya sea directos, incidentales o consecuentes, en relación con el rendimiento o la forma de los materiales aquí publicados, o con los programas de software que acompañan a este documento, o que surjan de estos.

Les modifications techniques qui améliorent les performances du dispositif peuvent être apportées sans avis préalable !

Ce document est la propriété intellectuelle de Kollmorgen. Tous droits réservés. Aucune partie de ce document ne peut être reproduite sous quelque forme que ce soit (photocopie, microfilm ou autre méthode) ni stockée, traitée, copiée ou distribuée de manière électronique sans l'autorisation écrite de Kollmorgen.

Les informations contenues dans ce document sont considérées exactes et fiables au moment de leur publication. Kollmorgen n'assume aucune responsabilité pour tout dommage ou perte résultant de l'utilisation de cette aide, et rejette expressément toute responsabilité ou tous dommages pour la perte de données, la perte d'utilisation et les dommages matériels de toute nature, directs, accessoires ou indirects, en ce qui concerne ou découlant des performances ou de la forme des matériaux présentés ici ou dans tout programme logiciel accompagnant ce document.

Modifiche tecniche volte a migliorare le prestazioni del dispositivo possono essere apportate senza preavviso.

Il presente documento è proprietà intellettuale di Kollmorgen. Tutti i diritti riservati. È vietata la riproduzione di qualsiasi parte di quest'opera in qualsiasi forma (fotocopia, microfilm o qualunque altro metodo) così come la memorizzazione, il trattamento, la copia o la distribuzione tramite mezzi elettronici senza l'autorizzazione scritta di Kollmorgen.

Le informazioni contenute nel presente documento sono ritenute accurate e affidabili al momento della pubblicazione. Kollmorgen non si assume alcuna responsabilità per eventuali danni o perdite risultanti dall'uso di questa guida e declina espressamente qualsiasi responsabilità o danni dovuti a perdita di dati, mancato utilizzo e danni materiali di qualunque tipo, diretti, accidentali o consequenziali, in relazione a o derivanti dal funzionamento o dalla forma dei materiali presentati qui o in eventuali programmi software che accompagnano il presente documento.

デバイス性能の向上のため、予告なく技術的な変更が実装される場合があります。

本書の知的財産権はKollmorgenに帰属しており、無断複写・転載は禁じられています。本書のいずれの部分も、Kollmorgenの書面による許可なしに、いかなる複製(写真、マイクロフィルムまたはその他の手段)、保管、改変、複写または電子的配布を行うことはできません。

本書に記載されている情報は、発行時において正確かつ信頼できるものですが、Kollmorgenはこのヘルプの使用により生じるいかなる損害や損失についても責任を負いません。また、本書の履行、資料形式または付随するあらゆるソフトウェアプログラムに関連して、あるいはこれらが原因で発生する、直接的、偶発的または結果的な、いかなるデータの損失、利用機会の喪失および財産の損害に対しても、一切の責任や損害賠償を明示的に否認します。

기기의 성능을 향상시키는 기술적 변경 사항은 사전 공지 없이 이루어질 수 있습니다!

본 문서의 지적 재산권은 Kollmorgen에 있습니다. All rights reserved. 본 저작물의 어떤 부분도 Kollmorgen의 서면 허가 없이 어떠한 형태(사진 복사, 마이크로필름 또는 기타 방법)로든 복제하거나 전자 수단으로 저장, 처리, 복사 또는 배포할 수 없습니다.

본 문서의 정보는 공개 시점을 기준으로 정확하고 신뢰할 수 있는 것으로 간주됩니다. Kollmorgen은 본 도움말의 사용으로 인한 어떠한 피해나 손실에 대해 책임을 지지 않으며, 본 문서 또는 본 문서와 함께 제공되는 소프트웨어 프로그램에 제시된 자료의 성능 또는 형태와 관련하여 또는 그로 인해 직접적, 우발적 또는 결과적으로 발생한 모든 유형의 데이터 손실, 사용 손실, 재산상의 피해에 대한 책임이나 손해를 명시적으로 부인합니다.

Alterações técnicas que melhoram o desempenho do dispositivo podem ser feitas sem aviso prévio!

Este documento é propriedade intelectual da Kollmorgen. Todos os direitos reservados. Nenhuma parte deste trabalho pode ser reproduzida sob qualquer forma (por fotocópia, microfilme ou qualquer outro método) ou armazenada, processada, copiada ou distribuída por meios eletrônicos sem a permissão escrita da Kollmorgen.

Acredita-se que as informações contidas neste documento sejam precisas e confiáveis no momento de sua divulgação. A Kollmorgen não assume nenhuma responsabilidade por quaisquer danos ou perdas resultantes do uso desta ajuda, e se isenta expressamente de qualquer responsabilidade ou danos por perda de dados, perda de uso e danos materiais de qualquer espécie, direta, incidental ou subsequente em relação a ou resultante do desempenho ou da forma dos materiais apresentados aqui ou em quaisquer programas de software que acompanhem este documento.

我们可能会进行技术更改以提高设备性能，恕不另行通知！

本文档知识产权归 Kollmorgen 所有，Kollmorgen 保留所有权利。未经 Kollmorgen书面许可，不得以任何形式（影印、缩微胶片或其他任何方式）复制本文档任何内容，亦不得以任何电子手段存储、处理、复制或分发本文档的任何内容。

据我们所知，本文档所含信息截至发布日是准确可靠的。对于因使用本帮助文档而导致的任何损害或损失，Kollmorgen概不承担任何责任。Kollmorgen特此明确声明，对于因本文档中的任何材料或随附本文档之任何软件程序的性能或形式导致的或与之相关的任何类型的数据丢失、使用中断损失或财产损害（包括直接、间接和衍生性损失或损害），Kollmorgen不承担任何法律责任或损害赔偿责任。

Support and Services

About Kollmorgen

When you need motion and automation systems for your most demanding applications and environments, count on Kollmorgen - the innovation leader for more than 100 years. We deliver the industry's highest-performing, most reliable motors, drives, AGV control solutions and automation platforms, with over a million standard and easily modifiable products to meet virtually any motion challenge. We offer manufacturing facilities, distributors and engineering expertise in all major regions around the world, so you can bring a better machine to market faster and keep it profitable for many years to come.

Kollmorgen Support Network

See the Kollmorgen Support Network for product support, product downloads, knowledge base answers, and information.

Kollmorgen Support Locations

North America
Kollmorgen Corporation
201 West Rock Road
Radford, VA 24141, USA

Web:	www.kollmorgen.com
Email:	kollmorgen.support@regalrexnord.com
Tel.:	+1-540-633-3545
Fax:	+1-540-639-4162

Europe
Kollmorgen Europe GmbH
Pempelfurtstr. 1
40880 Ratingen, Germany

Web:	www.kollmorgen.com
Email:	kollmorgen.tech.emeai@regalrexnord.com
Tel.:	+49-2102-9394-0
Fax:	+49-2102-9394-3155

South America
Altra Industrial Motion do Brasil Equipamentos Industriais Ltda.
Avenida João Paulo Ablas, 2970
Jardim da Glória, Cotia – SP
CEP 06711-250, Brazil

Web:	www.kollmorgen.com
Email:	kollmorgen.contato@regalrexnord.com
Tel.:	(+55 11) 4615-6300

China and SEA
Kollmorgen (Shanghai)
Room 1201, Building A,
Mangoo Hub, No.7 Longai Road,
Xuhui District, Shanghai, China
(Service available in English and Chinese)

Web:	www.kollmorgen.cn
Email:	kollmorgen.motion.sales.china@regalrexnord.com
Tel.:	+86-400 668 2802