mapp Framework

Table of contents

mapp Framework

To get started with the mapp Framework, proceed to the General information section.

mapp Framework - General information

This section contains information that is relevant for all of the mapp Frameworks.

Topics in this section:

mapp Framework - Introduction

mapp Technology is the overarching term for the ready-made, modular software products at B&R. This technology enables you to implement complex or tedious features (such as a recipe system) with just a few mapp function blocks and configuration settings rather than creating it from scratch with PLCopen. By using mapp Technology you can complete your application development up to three times faster, with significantly less code than if you wrote it entirely with PLCopen.

The mapp Framework takes mapp Technology one step further to provide the user with a universal starting point for mapp Technology. This even further reduces the amount of application code that must be written by the application engineer. The framework includes programming tasks and supporting configuration files with built-in best practices and application know-how. It is designed to be modular, so the user can easily add the specific parts that are relevant to the machine to an existing project.

Motivation and Goals

The motivation and overarching goals of the mapp Framework are as follows:

- Quality
  - The framework is designed with best practices in mind, which have been vetted by several experienced application engineers
  - The goal is to set each mapp user up for success
- Time Savings
  - By giving users a reliable starting point, we lower the learning curve of mapp Technology
  - New engineers can ramp up faster
  - Quicker time to market
- Simplicity
  - The mapp Framework is modular without being overly complex
  - With a standardized approach to mapp Technology, application support/hand-off and code maintenance become more straightforward
  - The framework is scalable according to the needs of the application
- Cost savings
  - Ultimately the use of the framework will result in cost savings for the machine, largely re-lated to a reduction in the required application engineering time

Availability

A mapp Framework is currently available for the following mapp Technologies:

- mapp AlarmX
- mapp Recipe
- mapp UserX
- mapp File
- mapp Backup
- mapp Axis / Cockpit

The development of additional mapp Services is in progress.

Framework Contents

Each mapp Framework contains the following:

- Logical View task(s)
- Configuration file(s)
- Supporting Help files

The supporting Help pages are individualized for each mapp Framework. These pages will identify what is included in the framework and any changes that are necessary to properly embed the framework to an existing application. It is important to note that the documentation focuses on the Framework itself, and not the fundamentals of mapp Services / mapp Motion. For details on the fundamentals of mapp Technology, refer to the respective sections in the Help ("Services" → "mapp Services", or "Motion control" → "mapp Motion").

IMPORTANT: Every mapp Framework Help section has a page titled "Required Modifications". The steps on this page must be executed in order to get the Framework into a functional state!

mapp Framework - Prerequisites

It is expected that the user has a base knowledge of Automation Studio and mapp Technology before utilizing the mapp Framework. Therefore, the following training courses are recommended as prerequisites:

- SEM210 – Automation Studio
- SEM270 – mapp Services
- SEM611 – mapp View
- SEM415 – mapp Axis

The mapp Framework is designed and intended to be used by B&R/partner application engineers and customer application engineers alike. In other words, any mapp Technology user can utilize the mapp Framework.

mapp Framework - Version information

This section describes the new features in each version of the mapp Framework.

Topics in this section:

mapp Framework - Version 0.1.x.x

Framework	Description
mapp AlarmX	Prototype version (0.1.x.x)
mapp Axis / Cockpit	Prototype version (0.1.x.x)
mapp Backup	Prototype version (0.1.x.x)
mapp File	Prototype version (0.1.x.x)
mapp Recipe	Prototype version (0.1.x.x)
mapp UserX	Prototype version (0.1.x.x)

mapp AlarmX

This section describes the mapp AlarmX Framework.

IMPORTANT: The steps on the "Required Modifications" page must be executed in order to get the Framework into a functional state!

Topics in this section:

mapp AlarmX Framework - Features

The following features and functionality are included in the mapp AlarmX Framework:

- 100 ready-made discrete value monitoring alarms and a boolean array to trigger them
- Localizable text is set up for each alarm
- Alarm mapping by severity with reactions
- A query along with the supporting state machine to be able to query large amounts of data
- A Python script to be able to easily import alarms into the AlarmX configuration from Excel {not available in prototype version}
- A mapp View content to display the current alarms, alarm history, and the alarm query
- The ability to acknowledge and export the alarms from the HMI

The following examples are embedded into the Framework:

- Examples for each type of monitoring alarm. Details are in the comments in the action AlarmSamples.st (starting on line 6).
    - Alarm names: LevelMonitoringExample, DeviationMonitoringExample, RateOfChangeExample
- An example for incorporating a snippet into the alarm. This example is available so you can easily

copy/paste the syntax for referencing a snippet in the mapp AlarmX configuration.
  - Alarm name: SnippetExample
- An example of using MpAlarmXAlarmControl() to manually set/reset an alarm from code
  - Alarm name: MpAlarmXControlExample
  - The supporting code is shown in the AlarmSamples.st action file (lines 18-22)

mapp AlarmX Framework - Task Overview

The AlarmMgt task contains all of the provided Framework code for mapp AlarmX. The chart below provides a description for the main task and each action file.

Filename	Type	Description
AlarmMgt.st	Main task code	General mapp function block handling. Handles alarm acknowledgment. Handles alarm history export. Checks if any reactions are active. Calls all actions.
AlarmHandling.st	Action	Defines the conditions which trigger each alarm. The AlarmMgt task contains a Boolean array called “Alarms”. Each index of the array corresponds to the Monitored PV for each of the 100 predefined alarms in the AlarmX configuration. This is also the designated place to define the conditions under which to inhibit alarms if inhibiting is needed in the application.
HMIActions.st	Action	Shows the alarm backtrace information on the HMI. Typically the backtrace action (GetBacktraceInformation) does not need to be modified.
ExecuteQuery.st	Action	Executes a query. Includes the supporting state machine to query large amounts of data.
AlarmSamples.st	Action	Calls the variables for the examples built into the Framework. Contains comments to explain each example.

mapp AlarmX Framework - Required Modifications

The Framework by default provides a solid foundation, but in order to integrate it fully into your application there are a few modifications that must be made.
The following list of modifications are required to get the Framework in a functional state within the application.

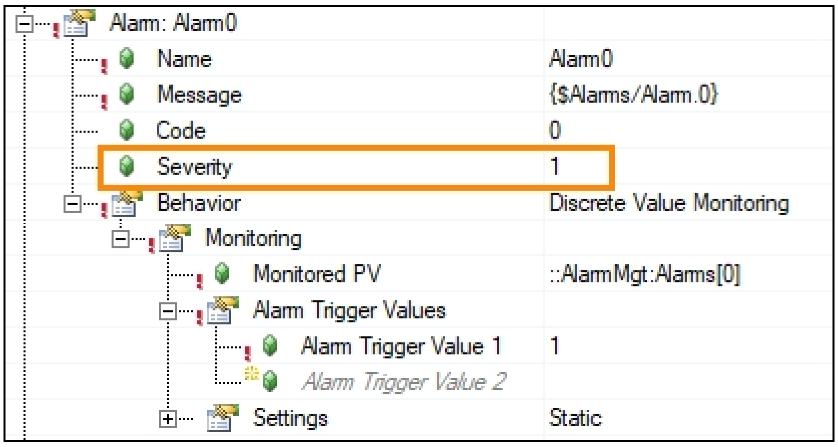
IMPORTANT: The steps on this page must be executed in order to get the Framework into a functional state!

1. Define the condition to **trigger** each alarm
   - The AlarmMgt task contains a Boolean array called “Alarms”. Each index of the array corresponds to the Monitored PV for each of the 100 predefined alarms in the AlarmX configuration.
   - For each of these 100 alarms, set the Alarms[] bit equal to the alarm condition that is relevant to the application. This is done in the AlarmHandling.st action file.

- For example, if Alarms[0] should trigger when Estop1 is pressed, then:
  `Alarms[0] := EStop1;`

2. Define the unique **alarm text** for each alarm in the Alarms.tmx file
   - Alarms.tmx is located in the Logical View within the Infrastructure → AlarmX package
   - Text ID Alarm.0 holds the alarm text for Alarm0, Text ID Alarm.1 holds the alarm text for Alarm1, etc
   - Remember to define the text for all languages that are relevant to the application

3. Assign an appropriate **severity** to each alarm in the Alarm List according to the alarm mapping
   - This is done in the AlarmX.mpalarmxcore configuration file
   - By default, the severity of all alarms is "1", which corresponds to the "Info" reaction
   - The severity you select will affect what alarm reaction is triggered when the alarm is triggered



Alarm: Alarm0	
Name	Alarm0
Message	{$Alarms/Alarm.0}
Code	0
Severity	1
Behavior	Discrete Value Monitoring
Monitoring	
Monitored PV	::AlarmMgt:Alarms[0]
Alarm Trigger Values	
Alarm Trigger Value 1	1
Alarm Trigger Value 2	
Settings	Static

4. Define how the application should **respond to each alarm reaction**
   - This is done via the MpAlarmXCheckReaction() function calls in AlarmMgt.st (starting on line 56)
   - Within each condition of the IF statement, program how the machine should respond to the reaction
   - For example, if the "Error" reaction is true, then stop all axes; if the "Warning" reaction is true, stop the machine after the next cycle; if the "Info" reaction is true, show a pop-up on the HMI with the information.
   - For more details on optional changes regarding alarm reactions / alarm mapping, see here.

5. If you imported the mapp View front end with the Framework, assign the provided **mapp View content** (content ID = AlarmX_content) to an area on a page within your visualization.

mapp AlarmX Framework - Optional Modifications

The Framework can be adjusted as needed for the application in any way necessary. This section summarizes some optional modifications that are commonly done to the Framework.

- Adjust the provided query (ActiveAlarms) source conditions ("SELECT" and "WHERE") in the AlarmX configuration.
- In the CPU configuration, modify the "mappAlarmX" file device to the desired storage medium. By default, this corresponds to the User partition (F:\AlarmX).
  - If you do, update or delete lines 11-17 of the AlarmMgt.st INIT program, which creates the directory F:\AlarmX if it does not already exist.
- Decide how you want to handle the situation where the query result has more than 20 alarms in it. See comments in ExecuteQuery.st starting on line 31.

- Add more alarms / delete unused alarms (see here for more details)
- Adjust the Alarm Mapping (see here for more details)
- Inhibit certain alarms under specific conditions (see here for more details)
- Add additional queries (see here for more details)

Topics in this section:

mapp AlarmX Framework - Add or Delete Alarms

Adding Alarms

The Framework comes with 100 discrete value monitoring alarms. If you need more discrete value monitoring alarms or any other type of alarm, add them accordingly. To do so:

1. Increase the size of the Alarms[] array in the AlarmMgt task according to how many alarms you need to add.
2. Add the new alarms to the AlarmList in the AlarmX.mpalarmxcore configuration file. Set the Monitored PV(s) to the newly added elements of the Alarms[] array from step 1.
3. Define the condition to trigger each new alarm in the AlarmHandling.st action file.

Note that it is permissible to mix and match different types of alarms (e.g. monitoring alarms with edge/persistent alarms that are triggered via MpAlarmXAlarmControl() or MpAlarmXSet()).

Deleting Alarms

The Framework comes with 100 discrete value monitoring alarms. If you don’t need all 100 of these alarms, delete them as needed. To do so:

1. Optionally decrease the size of the Alarms[] array in the AlarmMgt task according to how many alarms you want to remove.
2. Delete the unwanted alarms from the Alarm List in the AlarmX.mpalarmxcore configuration file.
3. Delete the alarm assignments for the unwanted alarms in the AlarmHandling.st action file. Note that these may be commented out from the original Framework import.

Deleting the Example Alarms

The Framework also comes with a number of example alarms. If you don't need some (or all) of these examples, then do the following:

1. Remove members from the AlarmExamples_typ or completely delete the AlarmExamples variable in the AlarmMgt task.
2. Delete the example alarm(s) from the Alarm List in the AlarmX.mpalarmxcore configuration file.
3. Edit or completely delete the AlarmSamples.st file, depending on the alarms you no longer need. If you delete this file, then also delete line 54 of AlarmMgt.st where the "AlarmSamples" action is called.

mapp AlarmX Framework - Alarm Mapping

The Framework establishes three alarm reactions by default within the alarm mapping:

1. Info (severity 1 – 9)
2. Warning (severity 10 – 19)
3. Error (severity 20 – 29)

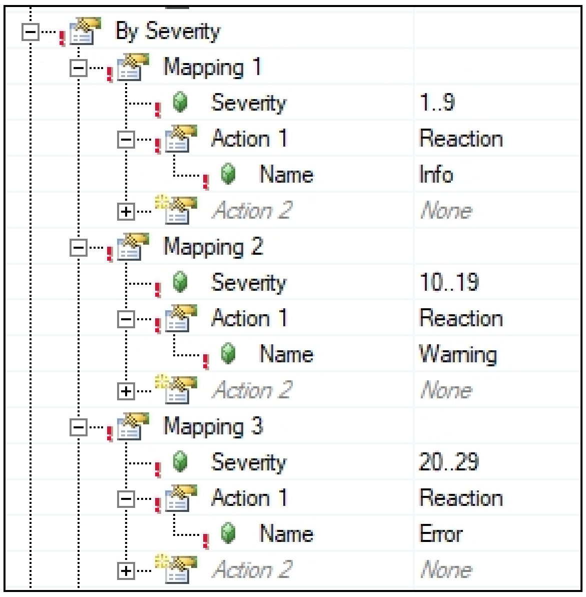
The MpAlarmXCheckReaction() function is called for each reaction within the AlarmMgt task.

The following optional changes should be considered to align the Framework with the application requirements:

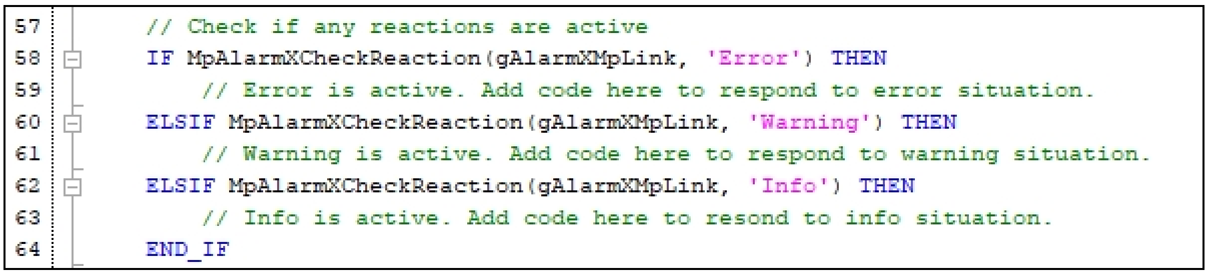
- Adjust the severity ranges

- This is done in the AlarmX.mpalarmxcore configuration file

- Rename the reactions
  - This is done in the AlarmX.mpalarmxcore configuration file
  - If you do this, remember to change the name in the MpAlarmXCheckReaction() function calls in AlarmMgt.st to the new name (starting at line 56).
- Add / remove reactions
  - This is done in the AlarmX.mpalarmxcore configuration file
  - If you do this, remember to add / remove function calls of MpAlarmXCheckReaction() in AlarmMgt.st accordingly (starting at line 56).
- Check for the reactions elsewhere in code
  - Typically the MpAlarmXCheckReaction() function is called from other tasks within the application. For example, the axis control task might check for the “Error” reaction to determine whether to send a stop command to the axes. Therefore, copy / paste the IF statements containing MpAlarmXCheckReaction() from AlarmMgt.st as needed around the application.



```
By Severity
    Mapping 1
        Severity        1..9
        Action 1        Reaction
            Name        Info
        Action 2        None
    Mapping 2
        Severity        10..19
        Action 1        Reaction
            Name        Warning
        Action 2        None
    Mapping 3
        Severity        20..29
        Action 1        Reaction
            Name        Error
        Action 2        None
```



```
57      // Check if any reactions are active
58      IF MpAlarmXCheckReaction(gAlarmXMpLink, 'Error') THEN
59          // Error is active. Add code here to respond to error situation.
60      ELSIF MpAlarmXCheckReaction(gAlarmXMpLink, 'Warning') THEN
61          // Warning is active. Add code here to respond to warning situation.
62      ELSIF MpAlarmXCheckReaction(gAlarmXMpLink, 'Info') THEN
63          // Info is active. Add code here to resond to info situation.
64      END_IF
```

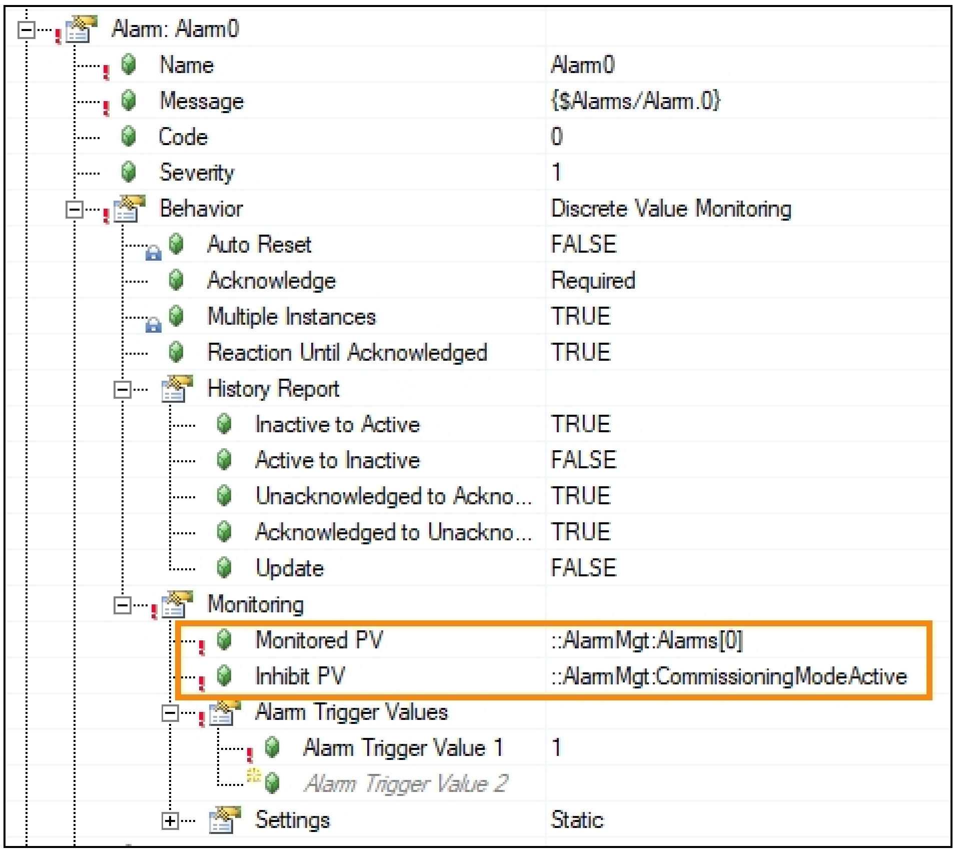
mapp AlarmX Framework - Inhibit Alarms

Decide whether each monitoring alarm should be inhibited at any point. If so:

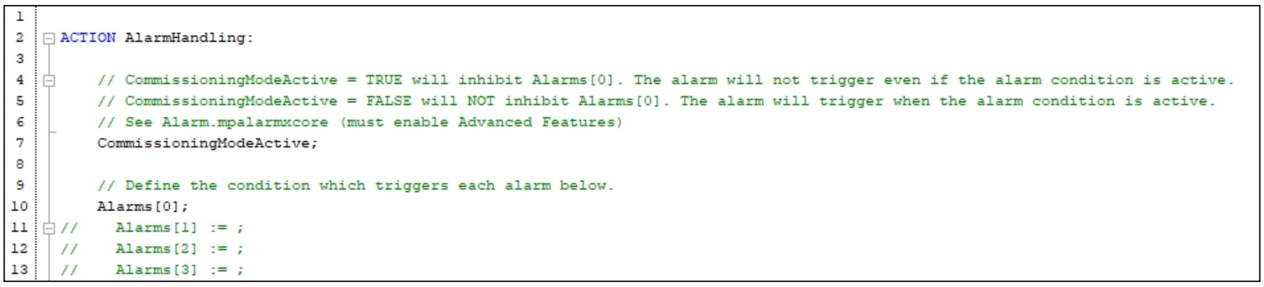
1. Assign an Inhibit PV to the alarm within the Alarm List of the mapp AlarmX configuration.
   - This is an advanced parameter, so remember to click the icon.
2. In the AlarmHandling.st action, write an IF statement to set the Inhibit PV according to a specific condition.
   - It is common to use the same Inhibit PV to inhibit multiple alarms. For example, you could have a “maintenance mode” bit which if True would simultaneously inhibit any alarms that should not be monitored during machine maintenance.

An example of using the Inhibit PV is built into Alarm0. The condition to set CommissioningModeActive to

True is not yet defined (this needs to be based on the application).



Alarm: Alarm0	
Name	Alarm0
Message	{$Alarms/Alarm.0}
Code	0
Severity	1
Behavior	Discrete Value Monitoring
Auto Reset	FALSE
Acknowledge	Required
Multiple Instances	TRUE
Reaction Until Acknowledged	TRUE
History Report	
Inactive to Active	TRUE
Active to Inactive	FALSE
Unacknowledged to Ackno...	TRUE
Acknowledged to Unackno...	TRUE
Update	FALSE
Monitoring	
Monitored PV	::AlarmMgt:Alarms[0]
Inhibit PV	::AlarmMgt:CommissioningModeActive
Alarm Trigger Values	
Alarm Trigger Value 1	1
Alarm Trigger Value 2	
Settings	Static



```

ACTION AlarmHandling:

    // CommissioningModeActive = TRUE will inhibit Alarms[0]. The alarm will not trigger even if the alarm condition is active.
    // CommissioningModeActive = FALSE will NOT inhibit Alarms[0]. The alarm will trigger when the alarm condition is active.
    // See Alarm.mpalarmxcore (must enable Advanced Features)
    CommissioningModeActive;

    // Define the condition which triggers each alarm below.
    Alarms[0];
//    Alarms[1] := ;
//    Alarms[2] := ;
//    Alarms[3] := ;
```

mapp AlarmX Framework - Add Queries

The ExecuteQuery.st action file contains the programming required to execute the query that comes with the Framework (the "ActivateAlarms" query).
It also contains the supporting state machine that is used to query large amounts of data.

If you'd like to add an additional query, the following steps are required:

1. Define the new query in the "Data Queries" section of the AlarmX.mpalarmxcore configuration file. Give it a unique name.
2. In the AlarmMgt task, create a variable declaration for a new MpAlarmXQuery() function block. (Each query needs a dedicated function block instance.)
3. Create a new variable of type "AlarmQueryType".
4. Copy/paste lines 4-41 of ExecuteQuery.st to duplicate that code. Then within the copied code:
    - Replace the query name assignment with the name of your new query from step 1.
        - `QueryActiveAlarms.Name := ADR('QueryNameFromStep1');`
    - Replace every instance of "QueryActiveAlarms" with your new function block name from step 2.
    - Replace every instance of "AlarmQuery" with your new variable name from step 3.

Repeat these steps for all additional queries.

mapp AlarmX Framework - Script

The provided Python Script can be used to easily import a list of alarms from Excel into the mapp AlarmX configuration file. This allows the user the flexibility to modify the alarm list in Excel if desired.

Here are the steps to use the Python script:
1. {Not available in prototype version}

mapp Axis Framework - mapp Axis / Cockpit

This section describes the mapp Axis Framework.

IMPORTANT: The steps on the "Required Modifications" page must be executed in order to get the Framework into a functional state!

Topics in this section:

mapp Axis Framework - Features

The following features and functionality are included in the mapp Axis Framework:

- A single-axis implementation which can be repeated for multiple axes. Includes basic commands, manual/automatic modes, and an overarching state machine for axis control
- Cyclic data exchange for current, lag error, and motor temperature
- Detailed alarm integration with mapp AlarmX
- Automatic setup of mapp Cockpit
- A mapp View content to show the axis faceplate

The mapp Axis framework is designed in such a way that the main axis control code within the AxisTemplate package is reused for all axes that you add. This makes it easy to update the overall process for all axes simultaneously. Any modifications made to the AxisControl.var, AxisStateMachine.st, ChangeConfiguration.st, or Recipe.st files within this package will apply to all axes. The Framework comes with this AxisTemplate package as well as one application axis set up for you in the AppAxis_1 package. To add additional axes, see here.

Axis-specific code is written within a dedicated package for that axis. For more details, see here.

mapp Axis Framework - Task Overview

The AxisControl task contains all of the provided Framework code for mapp Axis. The chart below provides a description for the main task and each action file.

Filename	Type	Reference vs Unique	Description
AxisControl.st	Main task code	Unique / dedicated file per axis	General mapp function block handling. Calls all actions.
AxisStateMachine.st	Action	Referenced file within the AxisTemplate package	Generic state machine used for all axes. Includes states like power-on, homing, manual/automatic operation, stopping, etc.
AxisControlModes.st	Action	Unique / dedicated file per axis	Programming that is specific to the axis. Contains manual and automatic mode state machines Manual mode is set up for basic jogging. Automatic mode is by default empty (to be filled in pe the application).
Recipe.st	Action	Referenced file within the AxisTemplate package	Registers axis variables to the recipe system. Optionally add additional variables as needed.
SimulationControl.st	Action	Unique / dedicated file per axis	Supporting code for simulation. Modify as needed for simulation requirements.
ManualCommand.st	Function	Unique / dedicated file per axis	Returns True/False depending on whether a manual mode command has been issued. Used in AxisStateMachine.st to handle the transition between manual and automatic mode. Change the ManualCommand assignment as needed. This functio allows the AxisStateMachine to remain generic.
AutomaticCommand.st	Function	Unique / dedicated file per axis	Returns True/False depending on whether a automatic mode command has been issued. Used in AxisStateMachine.st to handle the transition between manual and automatic mode. Change the AutomaticCommand assignment as needed. This function allows the AxisStateMachine to remain generic.
ChangeConfiguration.st	Action	Referenced file within the AxisTemplate package	Sets up the ability to change configuration settings at runtime. Modifications to this action are typically not necessary.

mapp Axis Framework - Required Modifications

The Framework by default provides a solid foundation, but in order to integrate it fully into your application there are a few modifications that must be made.
The following list of modifications are required to get the Framework in a functional state within the application.

IMPORTANT: The steps on this page must be executed in order to get the Framework into a functional state!

1. In the Configuration view, **add a single Axis configuration file.** Assign a unique MpLink.

2. In the Physical View, **add the drive** which will control this new axis. Then in the drive configuration, set the axis reference to the MpLink from step 1.

3. Edit the SimulationControl.st, AxisControl.st, ManualCommand.st, and AutomaticCommand.st files according to the **application requirements of your axis**. For details about what these files are intended for, see here.

4. If you imported the mapp View front end with the Framework, assign the **mapp View content** (content ID = Axis_content) to an area in a page within your visualization.

mapp Axis Framework - Optional Modifications

The Framework can be adjusted as needed for the application in any way necessary. This section summarizes some optional modifications that are commonly done to the Framework.

- The Framework comes with the axis template files and one axis already set up. To add an additional axis, see here.

mapp Axis Framework - Add Additional Axis

Here are the steps to add a new axis to the project using the provided axis template:

1. Copy/paste the AxisTemplate package in the Logical View to duplicate it. Rename the copied package. (Note: for the rest of these steps, the copied package will be referred to as the "NewAxis" package.)
2. Delete the following files from the NewAxis package:
    - AxisControl.var
    - AxisStateMachine.st
    - ChangeConfiguration.st
    - Recipe.st
3. Add the four files from step 2 above back into the NewAxis package as a reference to the files within the AxisTemplate package. In other words: from the Toolbox, add a “Referenced File” four times (one for each file in step 2). Set the reference to point back to the corresponding file in the AxisTemplate package.
4. In ModuleInfo.tmx of the NewAxis package:
    1. Change the namespace to something unique
    2. Modify the value of the text ID “Name” to describe the axis
5. In ModuleAlarms.tmx of the NewAxis package:
    1. Change the namespace to something unique
    2. Add entries to correspond to alarm messages, if desired
6. In the Configuration View, copy/paste ConvAxis.axis file in the mappMotion package. Re-name the copied file. Then within the copied file:
    1. Re-name the MpLink as desired
    2. Update the namespace references in the Alarms section to the namespaces you chose in steps 4 and 5. Alternatively, you can open the .axis file in a text editor and do a find-and-replace. The namespace is identified with the ‘$’ within the first set of curly brackets. For example: {$Namespace/TextID}
7. In the Logical View, in the AxisControl task of your copied package (NewAxis), update the MpLink reference on lines 15, 23 and 24 to the name you chose in step 6.
8. In the Physical View, add the drive which will control this new axis. Then in the drive configuration, set the axis reference to the MpLink you named in step 6.

Note that in a future version of the Framework, this process will be automated.

mapp Backup Framework - mapp Backup

This section describes the mapp Backup Framework.

IMPORTANT: The steps on the "Required Modifications" page must be executed in order to get the Framework into a functional state!

Topics in this section:

mapp Backup Framework - Features

The following features and functionality are included in the mapp Backup Framework:

- Easily create and restore a backup from the HMI
- View a list of all available backups from the HMI
- Choose whether to automatically backup the project at a certain interval

mapp Backup Framework - Task Overview

The BackupMgt task contains all of the provided Framework code for mapp Backup. The chart below provides a description for the main task and each action file.

Filename	Type	Description
BackupMgt.st	Main task code	General mapp function block handling. This task contains all of the necessary code to create, restore, and delete a backup of the program.
HMIActions.st	Action	Supporting code for HMI functionality. Loads and saves the backup configuration. File manager handling.
ChangeConfiguration.st	Action	Contains the programming to modify the backup configuration at runtime. Modifications to this action are typically not necessary.

mapp Backup Framework - Required Modifications

The Framework by default provides a solid foundation, but in order to integrate it fully into your application there are a few modifications that must be made.
The following list of modifications are required to get the Framework in a functional state within the application.

IMPORTANT: The steps on this page must be executed in order to get the Framework into a functional state!

1. If you imported the mapp View front end with the Framework, assign the **mapp View content** (content ID = Backup_content) to an area in a page within your visualization.

mapp Backup Framework - Optional Modifications

The Framework can be adjusted as needed for the application in any way necessary. This section summarizes some optional modifications that are commonly done to the Framework.

- In the CPU configuration, modify the “mappBackup” file device to the desired storage medium. By default, this corresponds to the User partition (F:\Backup).
    - If you do, modify or delete lines 10-16 of the BackupMgt.st INIT program, which creates the directory F:\Backup if it does not already exist.

mapp File Framework - mapp File

This section describes the mapp File Framework.

IMPORTANT: The steps on the "Required Modifications" page must be executed in order to get the Framework into a functional state!

Topics in this section:

- Features
- Task Overview
- Required Modifications
- Optional Modifications
- FIFO

mapp File Framework - Features

The following features and functionality are included in the mapp File Framework:

- General file explorer functionality on the HMI
    - Create / delete / sort / rename / search / copy / etc
- The ability to enable a FIFO for one file device. For more details, see here.

mapp File Framework - Task Overview

The FileMgt task contains all of the provided Framework code for mapp File. The chart below provides a description for the main task and each action file.

Filename	Type	Description
FileMgt.st	Main task code	General mapp function block handling. Calls all actions.

HMIActions.st	Action	Handles the file explorer operations for the HMI. Note that use of this file explorer is exclusive to administrative users.
FIFOOperations.st	Action	Handles the implementation of the FIFO (first-in-first-out) for the selected file device.

mapp File Framework - Required Modifications

The Framework by default provides a solid foundation, but in order to integrate it fully into your application there are a few modifications that must be made.
The following list of modifications are required to get the Framework in a functional state within the application.

IMPORTANT: The steps on this page must be executed in order to get the Framework into a functional state!

1. If you imported the mapp View front end with the Framework, assign the mapp View content (content ID = File_content) to an area in a page within your visualization.

mapp File Framework - Optional Modifications

The Framework can be adjusted as needed for the application in any way necessary. This section summarizes some optional modifications that are commonly done to the Framework.

- N/A

mapp File Framework - FIFO

The mapp File Framework comes with the option of enabling a FIFO (first-in-first-out) for a file device of your choosing. There are a few key details regarding this feature:

- The FIFO deletes the oldest files once the file device starts filling up. There are two configuration options for you to define what "filling up" means for your application. A dialog box on the HMI allows you to choose between the following:
    1. Define a **maximum memory size** and delete the oldest file once the total file device contents exceeds this size
    2. Define a **maximum number of files** and delete the oldest file once the number of files on the file device exceeds this value
- The user must select one file device for the FIFO to be active on. [note: in the prototype version, the FIFO will check the currently selected file device in the file explorer]
- Once activated, the FIFO will check this file device periodically for size/number of files. [note: in the prototype version, the check is completed when you click to view a file device in the file explorer]
- The FIFO will monitor all files in the root directory of the selected file device. Files contained within any sub-folders will not be checked!

mapp Recipe Framework - mapp Recipe

This section describes the mapp Recipe Framework.

IMPORTANT: The steps on the "Required Modifications" page must be executed in order to get the Framework into a functional state!

Topics in this section:

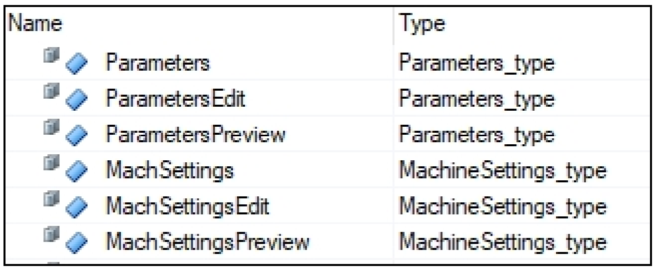
mapp Recipe Framework - Features

The following features and functionality are included in the mapp Recipe Framework:

- The infrastructure to set up two distinct recipe files, with a structure variable registered to each
- The ability to save recipes to a USB drive
- The ability to preview and edit a recipe on the HMI before loading it

mapp Recipe Framework - Recipe System Design

- There are two recipe categories in this recipe system: "Product" and "Machine".
- Each category gets saved to its own recipe file.
- One structure variable is registered to each category.
  - Parameters_type holds the product data.
  - MachineSettings_type holds the machine data.
- Three variables of each datatype are used in order to accomplish the preview functionality and the ability to edit a recipe without formally loading it (see below). For example, let's focus on the parameters structure:
  - The variable Parameters is the actual variable structure that holds the active recipe, which should be used around the application. This variable is not directly registered to the recipe.
  - The variable ParametersPreview is the only variable that is registered to the Product category of the recipe. This allows you to load and preview recipes without actually activating them in the application. If a recipe should be loaded to the application, then the Parameters variable structure gets set equal to the ParametersPreview structure by the framework.
  - The variable ParametersEdit is used as an intermediary structure to be able to edit the recipe without loading it to the application. It also acts as a buffer between the registered variable so that you can easily discard changes while editing if you need to.



Name	Type
Parameters	Parameters_type
ParametersEdit	Parameters_type
ParametersPreview	Parameters_type
MachSettings	MachineSettings_type
MachSettingsEdit	MachineSettings_type
MachSettingsPreview	MachineSettings_type

mapp Recipe Framework - Task Overview

The RecipeMgt task contains all of the provided Framework code for mapp Recipe. The chart below provides a description for the main task and each action file.

Filename	Type	Description
RecipeMgt.st	Main task code	General mapp function block handling. Registers structure variables to the two recipes. Loads the two default recipes. Calls all actions.
HMIActions.st	Action	Supporting code for HMI functionality, such as the recipe preview feature. Handles all recipe commands coming from the HMI (load, save, etc).
FileOperations.st	Action	File handling for copying recipes between the User partition and the USB stick.

mapp Recipe Framework - Required Modifications

The Framework by default provides a solid foundation, but in order to integrate it fully into your application there are a few modifications that must be made.
The following list of modifications are <u>required</u> to get the Framework in a functional state within the application.

IMPORTANT: The steps on this page must be executed in order to get the Framework into a functional state!

1. Two default recipes ("Machine.mcfg" for machine settings and "Default.par" for product data) must exist in the “mappRecipe” file device at startup. Initial versions of these files are provided in the framework within the CF package in the Logical View (CF → Recipe).

2. Modify the structure type within RecipeMgt.typ that is used for each registered variable to suit the needs of the application.
   - Parameters_type is used to hold the product data
   - MachineSettings_type is sued to hold the machine settings / commissioning data

   Further information about the design of the recipe system is provided here.

3. Register additional variables if needed. To do so:
   - Create a new function block declaration for MpRecipeRegPar()
   - Register the variable in the init program of RecipeMgt.st. Refer to lines 48-66 for reference, and replicate this code.

4. On RecipePreviewPars.content and RecipePreviewMachConfig.content, change the text on the labels and the bindings on the NumericOutputs/TextOutputs according to the parameters you want to display from your recipe.
   - These contents get loaded into the Preview window on Recipe.content, depending on the category selection
   - The texts are located in RecipePageTexts.tmx

  - Repeat on ContentEditRecpe.content and ContentCreateNewRecipe.content

5. If you imported the mapp View front end with the Framework, assign the **mapp View content** (content ID = Recipe_content) to an area in a page within your visualization.

mapp Recipe Framework - Optional Modifications

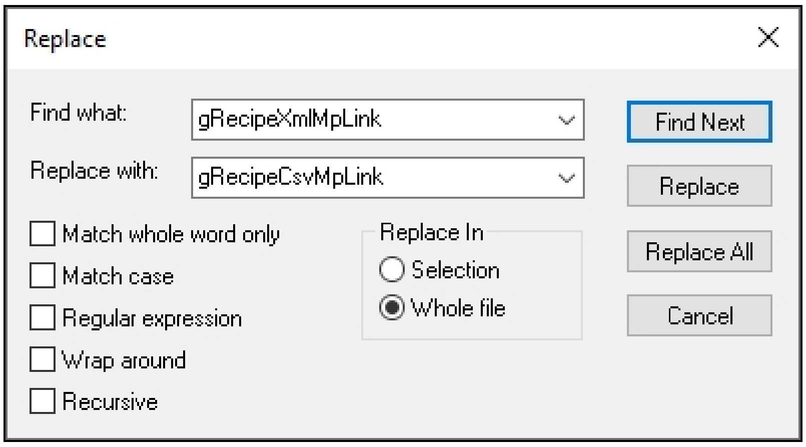
The Framework can be adjusted as needed for the application in any way necessary. This section summarizes some optional modifications that are commonly done to the Framework.

- Register your variables within their relevant tasks instead of all within the recipe task, if desired
- Change to CSV format (see here)
- In the CPU configuration, modify the “mappRecipe” file device to the desired storage medium. By default, this corresponds to the User partition (F:\Recipe).
  - If you do, modify or delete lines 10-15 of the RecipeMgt.st INIT program, which creates the directory F:\Recipe if it does not already exist.

mapp Recipe Framework - Change Recipe Format

By default the Framework sets up the recipe system in the XML format. The framework includes recipe configurations for both the XML and CSV formats to simplify changeover. If you'd like to switch to CSV, follow the steps below:

1. In RecipeMgt.var:
   1. Change the datatype of variable **MpRecipe_Preview** to **MpRecipeCsv**
   2. Change the datatype of variable **MpRecipe_0** to **MpRecipeCsv**
   3. Change the datatype of variable **Header** to **MpRecipeCsvHeaderType**
2. In RecipeMgt.st:
   1. Go to Edit → Find and Replace → Replace
   2. Replace all instances of **gRecipeXmlMpLink** with **gRecipeCsvMpLink**


Replace

Find what: gRecipeXmlMpLink

Replace with: gRecipeCsvMpLink

☐ Match whole word only
☐ Match case
☐ Regular expression
☐ Wrap around
☐ Recursive

Replace In
○ Selection
[✓ Whole file]

3. Delete any existing .mcfg and .par files from the Recipe file device (by default this is F:\Recipe), since these will be the XML format.
4. Copy the default recipe files that are provided in the CSV format (Logical View → CF → Recipe → CSVformat) to the root directory of the Recipe file device. Now the recipe system will load the CSV format versions of the files by default.

mapp Recipe Framework - Add Registered Variables in Other Tasks

xxx

In Recipe_content.eventbinding, scroll down to the Recipe_Eventbinding_Edit event.
Copy/Paste the first action (opcUa SetValueBool on ()::RecipeMgt:hmiRecipe.Parameters.EditDialogOpened).
On the copied action, change the refId to the EditDialogOpened variable you created in your non-recipe task.

In Recipe_content.eventbinding, scroll down to the Recipe_Eventbinding_Create event.
Copy/Paste the first action (opcUa SetValueBool on ()::RecipeMgt:hmiRecipe.Parameters.CreateDialogOpened).
On the copied action, change the refId to the CreateDialogOpened variable you created in your non-recipe task.

In RecipeDialog_Edit_content.eventbinding, scroll down to the Recipe_Eventbinding_EditConfirm event.
Copy/Paste the first action (opcUa SetValueBool on ()::RecipeMgt:hmiRecipe.Commands.SaveSelectedRecipe).
On the copied action, change the refId to the SaveRecipe variable you created in your non-recipe task.

In RecipeDialog_New_content.eventbinding, scroll down to the Recipe_Eventbinding_ConfirmCreate event.
Copy/Paste the action opcUa SetValueBool on ()::RecipeMgt:hmiRecipe.Commands.CreateRecipe.
On the copied action, change the refId to the CreateRecipe variable you created in your non-recipe task.

In Recipe_content.eventbinding, scroll down to the Recipe_Eventbinding_Load event.
Within the event handler for "Execute if: result = 1", copy/paste the opcUa action to SetValueBool on ()::RecipeMgt:hmiRecipe.Commands.LoadRecipe.
On the copied action, change the refId to the LoadRecipe variable you created in your non-recipe task.

Add an opcUa action. Set the refId to the XXX variable you created in your non-recipe task. Set the method to SetValueBool, and the static value to true.

mapp UserX Framework - mapp UserX

This section describes the mapp UserX Framework.

IMPORTANT: The steps on the "Required Modifications" page must be executed in order to get the Framework into a functional state!

Topics in this section:

mapp UserX Framework - Features

The following features and functionality are included in the mapp UserX Framework:

- Local user management (as opposed to Active Directory)
- Ability to import/export the user information via the HMI
- The following predefined roles: Everyone, Operator, Service, Admin
- The following predefined users: Admin, Operator, ServiceTech, Anonymous

mapp UserX Framework - Task Overview

The UserXMgt task contains all of the provided Framework code for mapp UserX. The chart below provides a description for the main task and each action file.

Filename	Type	Description
UserX.st	Main task code	Handles user interface interaction.

mapp UserX Framework - Required Modifications

The Framework by default provides a solid foundation, but in order to integrate it fully into your application there are a few modifications that must be made.
The following list of modifications are required to get the Framework in a functional state within the application.

IMPORTANT: The steps on this page must be executed in order to get the Framework into a functional state!

1. If you imported the mapp View front end with the Framework, assign the mapp View content (content ID = UserX_content) to an area in a page within your visualization.

mapp UserX Framework - Optional Modifications

The Framework can be adjusted as needed for the application in any way necessary. This section summarizes some optional modifications that are commonly done to the Framework.

- Modify the available roles and users if desired.
    - This is done in the Role.role and User.user files the Configuration View
    - If you make changes, be sure to update the UserX.mpuserx configuration file as well
        - Note that users only need to be added to the UserX.mpuserx configuration file if you want to utilize the "Language", "Measurement system", or "Additional Data" properties for that user. mapp UserX automatically has access to all users listed in User.user.
        - Similarly, note that roles only need to be added to the UserX.mpuserx configuration file if you want to specify administrative or access rights. mapp UserX automatically has access to all roles listed in Role.role.
- In the User.user file, set new passwords for Admin, Operator and ServiceTech. By default all three passwords are set to 123ABc. Do not set a password for the Anonymous user.
- In the CPU configuration, modify the “mappUserX” file device to the desired storage medium. By default,

this corresponds to the User partition (F:\UserX).

- If you do, modify or delete lines 10-15 of the UserX.st INIT program, which creates the directory F:\UserX if it does not already exist.