

mapp

FRAMEWORK



Table of contents

mapp Framework	4
General Information	4
Introduction	4
System Requirements	6
Prerequisites	6
Launching the Importer	6
User Partition Handling	7
Visualization Considerations	8
Version Information	8
Version 1.0.x	9
mapp AlarmX	9
Features	10
Task Overview	10
Required Modifications	11
Optional Modifications	12
Add or Delete Alarms	12
Alarm Mapping	13
Inhibit Alarms	14
Add Queries	15
mapp Axis / Cockpit	16
Features	16
Task Overview	16
Required Modifications	17
Optional Modifications	18
Add Additional Axis	18
Rename AppAxis_1	19
mapp Backup	20
Features	20
Task Overview	20
Required Modifications	21
Optional Modifications	21
mapp File	21
Features	21
Task Overview	22
Required Modifications	22
Optional Modifications	22
FIFO	22
mapp Recipe	23
Features	23
Recipe System Design	23
Task Overview	24
Required Modifications	24
Optional Modifications	25
Change Recipe Format	25
mapp UserX	26
Features	26
Task Overview	27

Required Modifications 27

Optional Modifications 27

mapp Framework

To get started with the mapp Framework, proceed to the [General information](#) section.



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 - General Information

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

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!

Topics in this section:

- [Introduction](#)
- [System Requirements](#)
- [Prerequisites](#)
- [Launching the Importer](#)
- [User Partition Handling](#)
- [Visualization Considerations](#)
- [Version Information](#)

mapp Framework - Introduction

Overview

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 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
 - Use of the Framework will result in cost savings for the machine, due 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)
- 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 - System Requirements

The mapp Framework is supported in the following versions:

mapp Technologies	Automation Studio	Automation Runtime
5.16 or later	4.8.2 or later	M4.34, F4.45, C4.53, C4.63, A4.72 or later

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

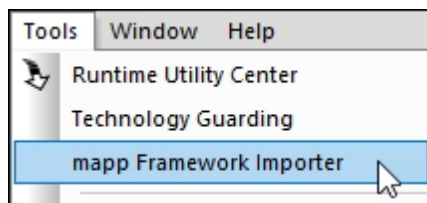
The mapp Framework is designed and intended to be used by B&R/partner application engineers and customer application engineers alike. In other words, any mapp Technology user can utilize the mapp Framework.

mapp Framework - Launching the Importer

To launch the mapp Framework import tool within Automation Studio, first make sure to set a version for the Framework in Project -> Change Runtime Versions:

Component	Preferred	In use	Scope
Automation Runtime	B4.92	B4.92	
Visual Components	not defined	not defined	
mapp View	5.18.0	5.18.0	
mapp Services	5.18.0	5.18.0	
mapp Cockpit	5.18.0	5.18.0	
mapp Motion	5.18.0	5.18.0	
ACP10 ARNC0 (Motion)	not defined	not defined	
mapp Control	not defined	not defined	
mapp Vision	not defined	not defined	
mapp Safety	not defined	not defined	
Safety Release	not defined	not defined	
Framework Importer	1.0.0	1.0.0	

Then navigate to Tools → mapp Framework Importer:



Notes:

- If you don't see this menu option after installing the Framework, then please close and re-open Automation Studio.
- If this menu option is grayed out, then the version number has not yet been set in Change Runtime Versions.

mapp Framework - User Partition Handling

Default User Partition Files

The Framework contains a few default files (e.g. recipe files) that need to be transferred to the user partition via an offline install or FTP access. These files are located in the Logical View within the "UserPartition" package.

(If the user changed the mapp Framework file devices to correspond to a location other than the user partition, then these files need to be placed in that new location instead.)

Switching between ARsim and Hardware

By default, all file devices in the Framework are pointing to the user partition (F:\). For easy transitions between ARsim and a physical PLC, do the following:

1. Create a new file with the extension ".bat" (batch file)
2. Add the following line to the batch file: **subst F: C:\UserPartition**
3. Replace **C:\UserPartition** with the location on your development PC's C:\ drive where you would like to hold the user partition files in simulation
4. Run the batch file prior to running the project in simulation. Alternatively, you can use the Windows Task

Scheduler to automatically run the batch file each time you boot your PC.

This will create a virtual drive on your PC called F: which functionally uses your user partition folder on the C drive.

This way, the file device definition can always use F:\ no matter if you are running in simulation or on real hardware.

mapp Framework - Visualization Considerations

Interface to the HMI

Each mapp Framework has a structure variable for commands, parameters, and status information from the HMI. This variable name always starts with "Hmi" followed by the mapp Technology (e.g. HmiRecipe).

Similarly, each mapp Framework has an action file called HMIActions.st, which contains all of the programming related to the HMI interface.

If you import the mapp View front end to the Framework, then the connections to this Hmi structure are already in place for you. If you are using VC4 or you prefer to design your mapp View HMI yourself, then you will need to connect the variables within this structure to your HMI yourself. The structure elements are explained in the Description column each .typ file so that you know how to connect them to your HMI. Note, however, that not all of the variables/structures are compatible with a VC4 setup (so additional re-work may be required).

Demo Page

A mapp View demo page is included in the mapp Framework so that you can quickly and easily navigate through all the imported contents in Chrome. This page is intended for demonstration purposes only. It is not intended to be used in the final application.

If you do not yet have a mapp View visualization when you import the Framework, then this Demo page will be assigned as the start page. If you have an existing mapp View application that the Framework merges into, then the Demo page will be added to the pages list and you have to add navigation to this page yourself.

Administrative Rights

The following functionality on the mapp View front end is restricted to administrative access only:

- Backup
 - The ability to restore or delete a backup, and change the automatic backup settings
- File
 - The ability to cut/delete a file and configure the FIFO
- Recipe
 - The ability to create/delete/load/edit a Machine Settings recipe.
- UserX
 - The ability to view and edit/add/delete/import/export the users in the UserList widget

mapp Framework - Version Information

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

The mapp Framework versioning system follows the versioning scheme of mapp Technology:

Major.Minor.Bugfix

Topics in this section:

- [Version 1.0.x](#)

mapp Framework - Version 1.0.x

v1.0.1

Framework	Description
mapp AlarmX	No changes
mapp Axis / Cockpit	Bugfixes: <ul style="list-style-type: none">• Fixed manual mode set speed and stop deceleration connections• Fixed bug regarding manual mode stopping behavior• Fixed State variable capitalization• Added units to the datapoints
mapp Backup	No changes
mapp File	No changes
mapp Recipe	No changes
mapp UserX	No changes
Import Tool	New behavior: <ul style="list-style-type: none">• The Framework no longer sets a version number automatically for the Framework Importer in Change Runtime Versions. This must be set manually. Bugfix: <ul style="list-style-type: none">• Fixed the situation where the importer wouldn't launch if AS was not installed to C:\BrAutomation\

v1.0.0

Framework	Description
mapp AlarmX	First release
mapp Axis / Cockpit	First release
mapp Backup	First release
mapp File	First release
mapp Recipe	First release
mapp UserX	First release
Import Tool	First release

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:

- [Features](#)
- [Task Overview](#)

- [Required Modifications](#)
- [Optional Modifications](#)

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 each alarm
- Localizable text 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 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 file AlarmSamples.st (starting on line 5).
 - 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 24-32)

mapp AlarmX Framework - Task Overview

The AlarmMgr task contains all of the provided Framework code for mapp AlarmX. This task is located in the Logical View within the Infrastructure package. The chart below provides a description for the main task and each action file.

Filename	Type	Description
AlarmMgr.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 AlarmMgr 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. Also sets up the table configuration for the query table.
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 AlarmMgr 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 the light curtain is interrupted and you are not in maintenance mode, then:

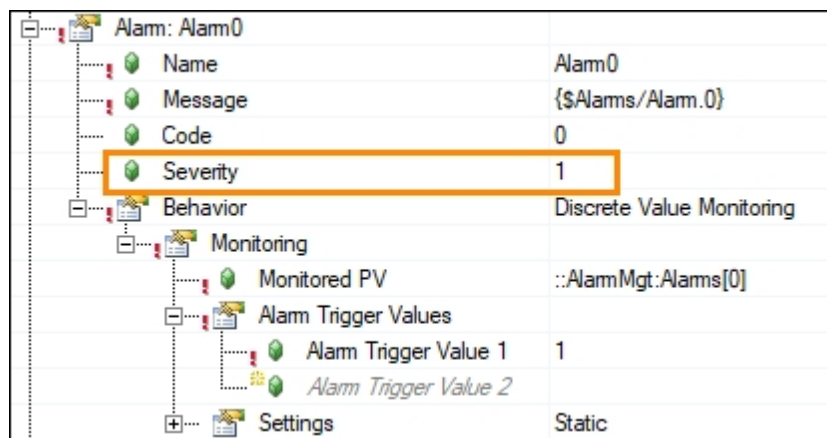
```
Alarms[0] := LightCurtainInterrupted AND NOT MaintenanceMode;
```

2. Define the unique **alarm text** for each alarm in the Alarms.tmx file

- Alarms.tmx is located in the Logical View → Infrastructure package → AlarmX package
- Text ID Alarm.0 holds the alarm text for Alarm0 (Alarms[0]), Text ID Alarm.1 holds the alarm text for Alarm1 (Alarms[1]), 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 AlarmXCfg.mpalarmxcore configuration file, which is located in the Configuration View → {CPU name} package → mapp Services package → AlarmX package
- By default, the severity of all alarms is "1", which corresponds to the "Info" reaction
- The severity you select will affect which alarm reaction is triggered when the alarm is triggered



4. Define how the application should **respond to each alarm reaction**

- This is done via the MpAlarmXCheckReaction() function calls in AlarmMgr.st (starting on line 62)

- Within each condition of the IF statement, add code to define 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. If you did not import the mapp View front end, then connect the HmiAlarmX structure elements to your visualization accordingly (see [here](#) for more details).

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 AlarmXCfg.mpalarmxcore.
- 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 38.
- If you are triggering alarms via MpAlarmXControl() or MpAlarmXSet(), consider setting/resetting the alarms throughout the application and not solely within the AlarmMgr task.
 - For example, if you have an error in the recipe system, you can trigger it directly in the recipe task.
 - Decentralization in this way is a key benefit of mapp AlarmX.
- Currently the provided query runs only when a "Run query" button is clicked on the HMI. If you'd rather that the query runs anytime there is new data is available, then refer to the comments in the ACTIVE_ALARM_WAIT state of the ExecuteQuery action.
- In the CPU configuration, modify the "mappAlarmXFiles" file device to the desired storage medium. By default, this corresponds to the User partition (F:\AlarmX).
 - If you do, update or delete lines 10-16 of the AlarmMgr.st INIT program, which creates the directory F:\AlarmX if it does not already exist.
- 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:

- [Add or Delete Alarms](#)
- [Alarm Mapping](#)
- [Inhibit Alarms](#)
- [Add Queries](#)

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 AlarmMgr.var according to how many alarms you need to add.
2. Add the new alarms to the AlarmList in the AlarmXCfg.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 AlarmMgr.var according to how many alarms you want to remove.
2. Delete the unused alarms from the Alarm List in the AlarmXCfg.mpalarmxcore configuration file.
3. Delete the alarm assignments for the unused alarms in the AlarmHandling.st action file. Note that these may be commented out from the initial Framework import.

Deleting the Example Alarms

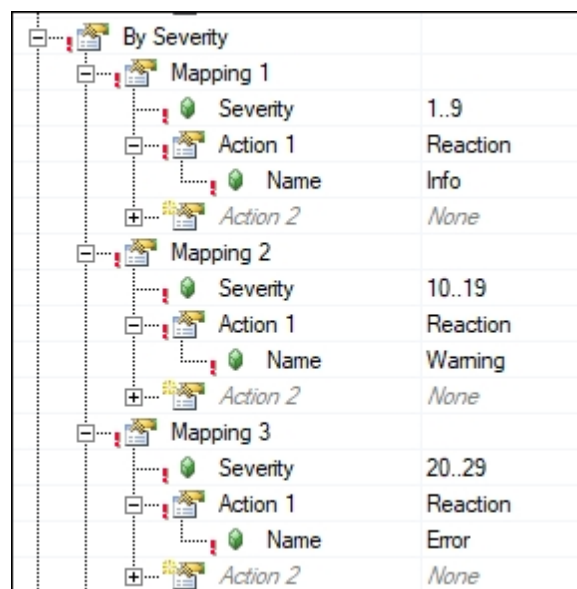
The Framework also comes with a number of example alarms. If you don't need some (or all) of these examples, then do the following:

1. Remove members from the AlarmExamples_typ or completely delete the AlarmExamples variable in AlarmMgr.var.
2. Delete the example alarm(s) from the top of the Alarm List in the AlarmXCfg.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 lines 19 & 60 of AlarmMgr.st where the AlarmSampleInit and AlarmSampleFub actions are 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 AlarmMgr task.

```

62 // Check if any reactions are active.
63 // Typically the MpAlarmXCheckReaction() function is called from other tasks within the application.
64 // For example, the axis control task might check for the 'Error' reaction to determine whether to send a stop command to the axes.
65 // Therefore, copy/paste these IF statements to other places in the application as needed.
66 IF MpAlarmXCheckReaction(gMplinkAlarmXCore, 'Error') THEN
67     // Error is active. Add code here to respond to error situation.
68 ELSEIF MpAlarmXCheckReaction(gMplinkAlarmXCore, 'Warning') THEN
69     // Warning is active. Add code here to respond to warning situation.
70 ELSEIF MpAlarmXCheckReaction(gMplinkAlarmXCore, 'Info') THEN
71     // Info is active. Add code here to respond to info situation.
72 END_IF


```

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

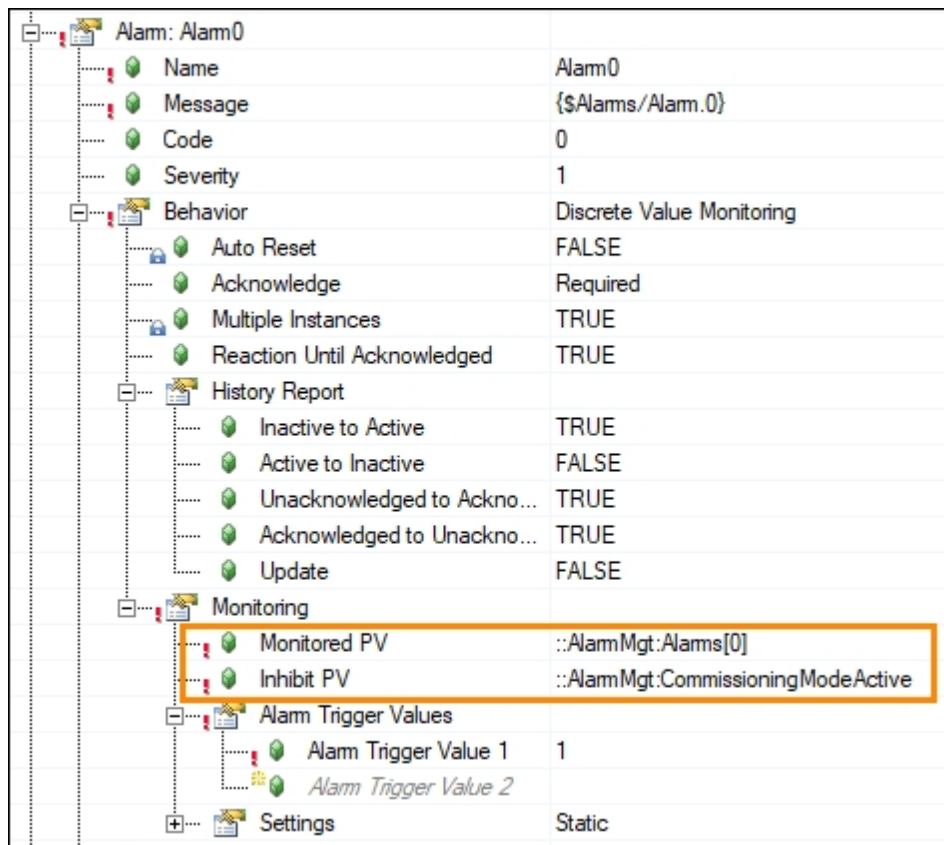
- Adjust the severity ranges
 - This is done in the AlarmXCfg.mpalarmxcore configuration file
- Rename the reactions
 - This is done in the AlarmXCfg.mpalarmxcore configuration file
 - If you do this, remember to change the name in the MpAlarmXCheckReaction() function calls in AlarmMgr.st to the new name (starting at line 66).
 - Examples of alternative reaction names:
 - SlowDownConveyor
 - HydraulicMotorOff
 - StopAllMotion
 - YellowLamp
- Add / remove reactions
 - This is done in the AlarmXCfg.mpalarmxcore configuration file
 - If you do this, remember to add / remove function calls of MpAlarmXCheckReaction() in AlarmMgr.st accordingly (starting at line 66).
- 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 AlarmMgr.st as needed around the application.

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 AlarmXCfg.mpalarmxcore configuration file.
 - 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 in AlarmHandling.st (this needs to be based on the application).



```

1  |
2  | ACTION AlarmHandling:
3  |
4  |     // CommissioningModeActive = TRUE will inhibit Alarms[0]. The alarm will not trigger even if the alarm condition is active.
5  |     // CommissioningModeActive = FALSE will NOT inhibit Alarms[0]. The alarm will trigger when the alarm condition is active.
6  |     // See Alarm.mpalarmxcore (must enable Advanced Features)
7  |     CommissioningModeActive;
8  |
9  |     // Define the condition which triggers each alarm below.
10 |     Alarms[0];
11 |     // Alarms[1] := ;
12 |     // Alarms[2] := ;
13 |     // 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. In AlarmMgr.var:
 1. Copy/paste the variable declaration for QueryActiveAlarms_0 and give it a unique name. (Each query needs a dedicated function block instance.)
 2. Copy/paste the variable declaration for AlarmQuery and give it a unique name.
2. In the AlarmXCfg.mpalarmxcore configuration file:
 1. Define the new query in the "Data Queries" section at the bottom. Give it a unique name.
 2. Use the new query variable you created in step 1.2 for the process variable connections and update count.
3. Copy/paste lines 3-47 of ExecuteQuery.st to duplicate that code. Then within the copied code:
 1. Replace every instance of "QueryActiveAlarms_0" with your new function block name from step 1.1.

2. Replace the query name assignment with the name of your new query from step 2.1.
 - `NewQueryFUB_0.Name := ADR('NewQueryNameFromStep2.1');`
3. Replace every instance of "AlarmQuery" with your new variable name from step 1.2.
4. If you'd like the new query to only run based on a button press, then add a new boolean within HmiAlarmX.Commands for the new trigger. Then update the IF statement accordingly within the ACTIVE_ALARM_WAIT state.

Repeat these steps for all additional queries.

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:

- [Features](#)
- [Task Overview](#)
- [Required Modifications](#)
- [Optional Modifications](#)

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 AxisMgr.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 AxisMgr task contains all of the provided Framework code for mapp Axis. This task is located in the Logical View within the MachineControl package. The chart below provides a description for the main task and each action file.

Filename	Type	Reference vs Unique	Description
AxisMgr.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.
SimulationControl.st	Action	Unique / dedicated file per axis	Supporting code for simulation. Modify as needed for simulation requirements.
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 per the application).
Recipe.st	Action	Referenced file within the AxisTemplate package	Registers axis variables to the recipe system. Optionally add additional variables as needed.
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 function 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, AxisMgr.st, ManualCommand.st, and AutomaticCommand.st files according to the unique **application requirements of your axis**. For details about what these files are

intended for, see [here](#).

4. The axis task(s) should run in cyclic 1, and the maximum allowed cycle time is 20ms. Adjust the **TC1 cycle time** accordingly.
5. 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. If you did not import the mapp View front end, then connect the HmiAxis structure elements to your visualization accordingly (see [here](#) for more details).

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 (AppAxis_1). To add an additional axis, see [here](#).
- To rename AppAxis_1 to something that more specifically reflects its purpose, see [here](#).

Topics in this section:

- [Add Additional Axis](#)
- [Rename AppAxis_1](#)

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 the AxisTemplate package that is located in the Infrastructure package of the Logical View. Paste it into the MachineControl package.
2. Rename the copied package. (Note: for the rest of these steps, the copied package will be referred to as the "NewAxis" package.)
3. Rename the contained AxisMgr task to match the new package name. Deploy this task to the Software Configuration.
4. Delete the following files from the NewAxis package:
 - AxisMgr.var
 - AxisStateMachine.st
 - ChangeConfiguration.st
 - Recipe.st
5. Add the 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.
6. In AxisInfo.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.
7. In AxisAlarms.tmx of the NewAxis package:
 1. Change the namespace to something unique.
 2. Delete all existing entries. Generic axis alarms are handled by the AxisTemplate package.
 3. Add entries to correspond to alarm messages as needed.

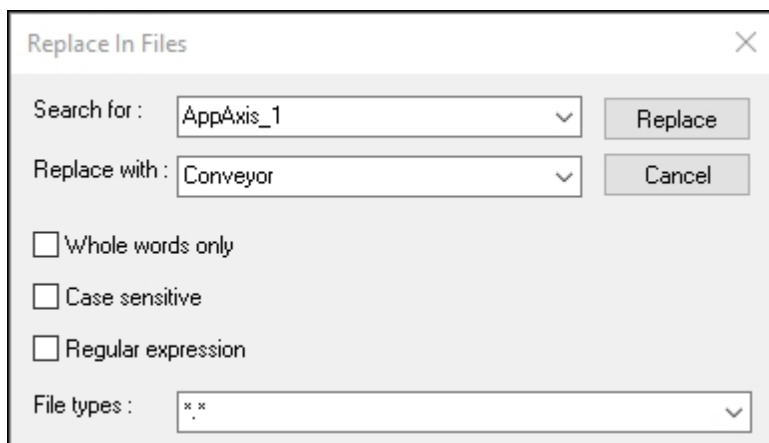
8. In the Configuration View → TextSystem → TC.textconfig, add the two text files from step 6 and 7 to the "Tmx files for target" list.
9. In the Configuration View, copy/paste AppAxis_1.axis file in the mappMotion package. Re-name the copied file. (Note that the renamed file cannot exactly match the name of the corresponding task in the Logical View.) 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 6.1 and 7.1. 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}
10. Optional: If you are planning to run the axis in simulation, then also copy/paste the VAppAxis1.purevaxcfg file from within the AppAxis_1 package in the Configuration View. In the pasted file:
 1. Rename the MpLink
 2. Assign the axis reference to the MpLink from step 9.1.
11. In the Logical View → MachineControl package → NewAxis package → AxisMgr.st, update the MpLink reference on lines 27 & 28 to the name you chose in step 9.1.
12. In the Configuration View → mappServices package, copy/paste the AppAxis_1 package. Rename the copied package. Then regarding the two contained files:
 1. Rename the .mpalarmxcore file accordingly. Within this file, re-name the MpLink. Then in AxisMgr.st on line 37, paste this new MpLink.
 2. Rename the .mpcomgroup file accordingly. Within this file:
 1. Rename the MpLink of the group accordingly.
 2. Change the Child 1 element to match the new MpLink from step 9.1.
 3. Change the Child 2 element to match the new MpLink name you created step 12.1.
13. In the Physical View, within the drive configuration that corresponds to this axis, set the axis reference to the MpLink you named in step 9.1.

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

mapp Axis Framework - Rename AppAxis_1

Here are the steps to rename the provided AppAxis_1 to something more specific to the application:

1. Choose a name that is 10 characters or less.
2. Do a replace-in-files (Edit→ Find and Replace → Replace in Files) to replace all instances of AppAxis_1 with your new name. Make sure the box for "Whole words only" is NOT checked, and the file types filter is set to *.*.



3. Manually rename the following files/packages in the Logical View → MachineControl package to your new name:

1. AppAxis_1 package
2. AppAxis_1 task
3. AppAxis_1_Info.tmx
4. AppAxis_1_Alarms.tmx
4. In the Configuration View → Connectivity → OpcUA, open up Axis.uad in a text editor. On line 25, replace AppAxis_1 with your new name.
5. In the Configuration View → TextSystem, open up TC.textconfig in a text editor. Replace all instances of AppAxis_1 with your new name (4 instances total).

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:

- [Features](#)
- [Task Overview](#)
- [Required Modifications](#)
- [Optional Modifications](#)

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 BackupMgr task contains all of the provided Framework code for mapp Backup. This task is located in the Logical View within the Infrastructure package. The chart below provides a description for the main task and each action file.

Filename	Type	Description
BackupMgr.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. If you did not import the mapp View front end, then connect the HmiBackup structure elements to your visualization accordingly (see [here](#) for more details).

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 “mappBackupFiles” 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 BackupMgr.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 FileMgr task contains all of the provided Framework code for mapp File. This task is located in the Logical View within the Infrastructure package. The chart below provides a description for the main task and each action file.

Filename	Type	Description
FileMgr.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. If you did not import the mapp View front end, then connect the HmiFile structure elements to your visualization accordingly (see [here](#) for more details).

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.
- Once activated, the FIFO will check this file device periodically for size/number of files.
 - Note that the FIFO will not automatically delete files if the File_content is active, because you could be viewing a different file device at the time and the FIFO would interrupt your viewing.
- 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:

- [Features](#)
- [Recipe System Design](#)
- [Task Overview](#)
- [Required Modifications](#)
- [Optional Modifications](#)

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
- The recipe system is set up in the XML format. (To convert to a CSV recipe system, see [here](#).)
 - Both the RecipeXML.mprecipexml and the RecipeCSV.mprecipecsv configuration files are already included in the Framework to enable the ability to easily switch back and forth. Only one of these configuration files is used at a time (i.e. the Mplink from only one of these files is referenced in the application).

mapp Recipe Framework - Recipe System Design

- There are two recipe categories in this recipe system: "Parameters" and "Machine Configuration".

- 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. 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

Note that the Recipe Framework has some While loops in the Initialization program, whereas the other Frameworks do not. The reason for this is because the Framework loads the default recipes in the initialization program. As a result, the supporting function blocks must be active before the load command can be successfully triggered. The default recipes are loaded in the Initialization program rather than the Cyclic program to avoid the need for a global variable across the Frameworks to indicate when the recipes are finished loading.

mapp Recipe Framework - Task Overview

The RecipeMgr task contains all of the provided Framework code for mapp Recipe. This task is located in the Logical View within the Infrastructure package. The chart below provides a description for the main task and each action file.

Filename	Type	Description
RecipeMgr.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 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. 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 for reference in the Logical View (UserPartition → Recipe → CSVformat and UserPartition → Recipe → XMLformat). Only the files that are directly in the UserPartition → Recipe package will be loaded by the recipe system (which by default is the XML file format).
2. Modify the **structure types** within RecipeMgr.typ for each registered variable to suit the needs of the application.
 - Parameters_type is used to hold the product data
 - MachineSettings_type is used to hold the machine settings / commissioning dataFurther information about the design of the recipe system is provided [here](#).
3. 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.
 - 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 ContentEditRecipe.content and ContentCreateNewRecipe.content
4. If you did not import the mapp View front end, then connect the HmiRecipe structure elements to your visualization accordingly (see [here](#) for more details).

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.

- Change to CSV format (see [here](#))
- In the CPU configuration, modify the "mappRecipeFiles" file device to the desired storage medium. By default, this corresponds to the User partition (F:\Recipe).
 - If you do, modify or delete lines 10-16 of the RecipeMgr.st INIT program, which creates the directory F:\Recipe if it does not already exist.

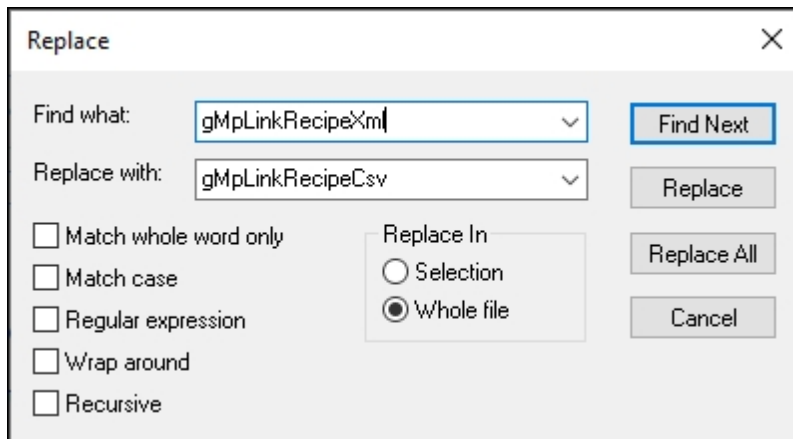
Topics in this section:

- [Change Recipe Format](#)

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 RecipeMgr.var:
 1. Change the datatype of variable **MpRecipe_0** to **MpRecipeCsv**
 2. Change the datatype of variable **Header** to **MpRecipeCsvHeaderType**
2. In RecipeMgr.st:
 1. Go to Edit → Find and Replace → Replace
 2. Replace all instances of **gMpLinkRecipeXml** with **gMpLinkRecipeCsv** in the whole file. There are 8 occurrences total.



3. Delete or move any existing .mcfg and .par files out of 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 → UserPartition → 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 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:

- [Features](#)
- [Task Overview](#)
- [Required Modifications](#)
- [Optional Modifications](#)

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 UserXMgr task contains all of the provided Framework code for mapp UserX. This task is located in the Logical View within the Infrastructure package. The chart below provides a description for the main task and each action file.

Filename	Type	Description
UserXMgr.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. If you did not import the mapp View front end, then connect the HmiUserX structure elements to your visualization accordingly (see [here](#) for more details).

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 UserXCfg.mpuserx configuration file as well
 - Note that users only need to be added to the UserXCfg.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 UserXCfg.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 “mappUserXFiles” file device to the desired storage medium. By default, this corresponds to the User partition (F:\UserX).
 - If you do, modify or delete lines 10-16 of the UserXMgr.st INIT program, which creates the directory F:\UserX if it does not already exist.