FAQ COMBIVIS Studio HMI

Recipes

Part	Version	Revision	Date	Status
en	5.1.1183.1	001	2021-05-07	Released

Content

Introduction

This document explains how to create and use recipes in an HMI project. Recipes are often used to change parameters within a production process. Through recipes, sets of parameters may be easily defined and switched between during runtime. A recipe consists of an index and temporary variable values. The corresponding actual variables are assigned the temporary values based on the recipe index that is appropriate for operation of the machine.

Define Recipe Resource

This section explains how to create recipes in an HMI project.

Create Recipe Variables

For this example, four numeric variables (Var1,...,Var4) are defined whose values will be managed by the recipe. Recipes also require an index variable, “IndexVar” in this case. The index variable identifies each parameter set and is used to switch between recipes. This index can either be an integer or a string. String indexes have the added benefit of being able to describe the recipe with text.

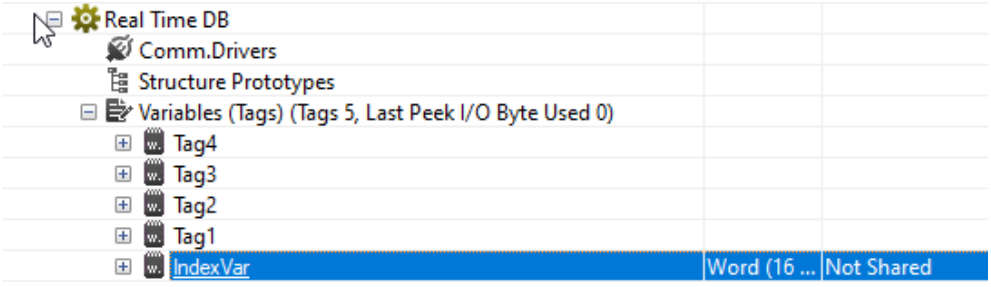


Fig. 1: Recipe Variables Defined

Create Recipe Resource

Add a recipe by right-clicking on *Data Loggers And Recipes* in the project resources and select *Add a new Recipe*.

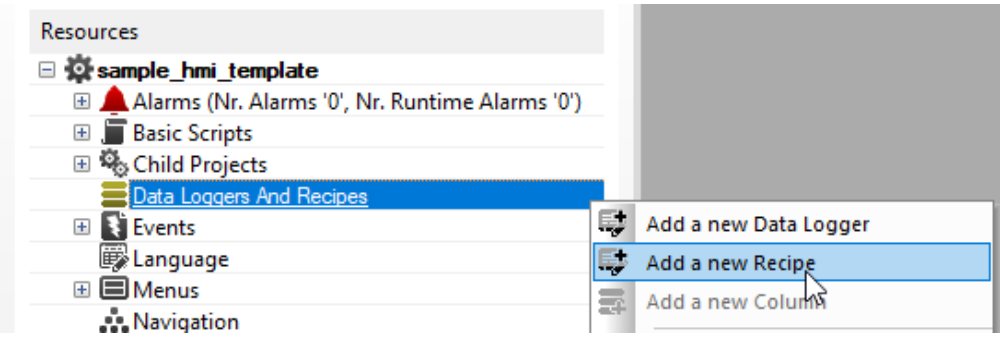


Fig. 2: Add Recipe

Change the recipe name if desired. To add variables to the recipe, select them in the Real Time DB and drag them to the newly created recipe.

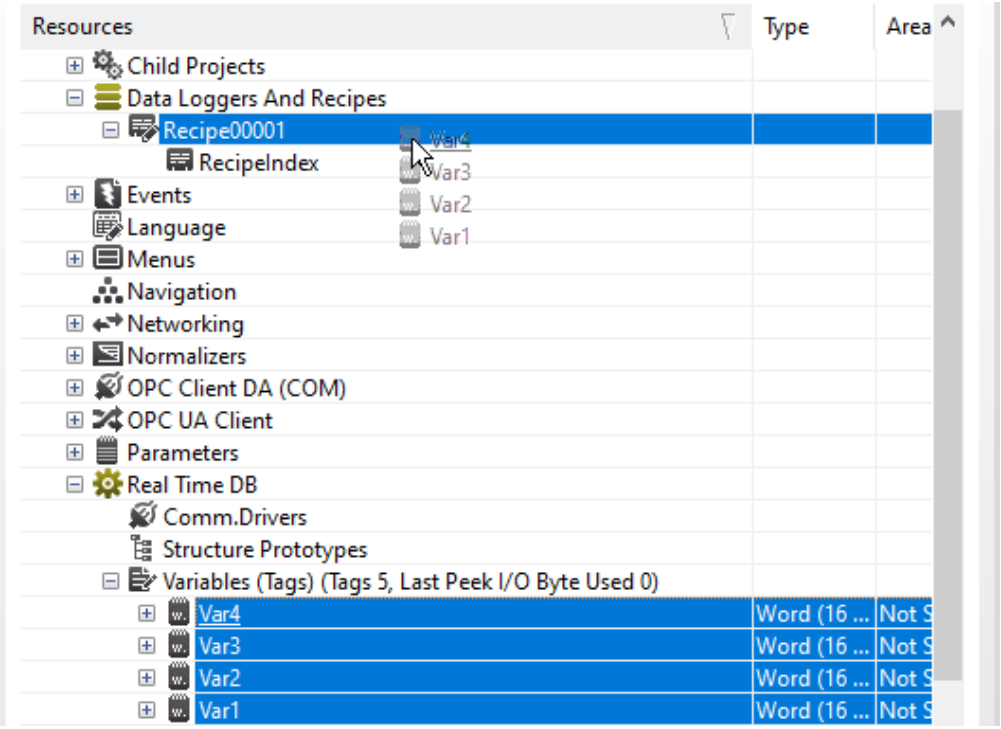


Fig. 3: Add Variables to Recipe

Next, select the *RecipeIndex* that was automatically added to the recipe resource upon creation. In the *Variable* field, browse for the index variable that was defined in Real Time DB (IndexVar in this example). Ensure that the *Recipe Index* property is enabled.

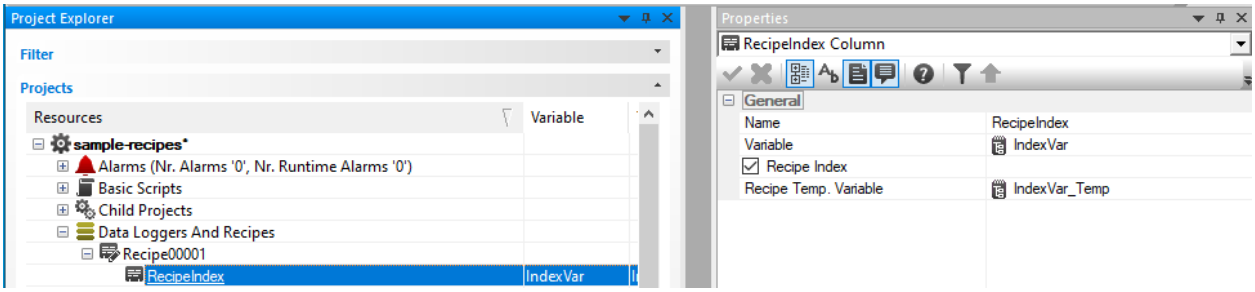


Fig. 4: Recipe Index Properties

For this example, two recipes are created: Recipe00001 (Var1,. . . ,Var4) and Recipe00002 (Var5,. . . ,Var8). The former will have an integer recipe index variable and will demonstrate the recipe screen that can be automatically generated and configured by Studio HMI. The latter will have a string recipe index variable and will be used to demonstrate the Recipe Manager window element in the following section.

Create Recipe Screen

This section explains how to create screens to manage recipes during runtime in an HMI project. This will be done in two ways: (1) automatically generating a recipe screen from the recipe resource and (2) adding and configuring a Resource Manager element from the Toolbox.

Create Recipe Screen

To generate a screen for managing a recipe that has been created, right-click on the recipe resource and select *Create Recipe Screen*.

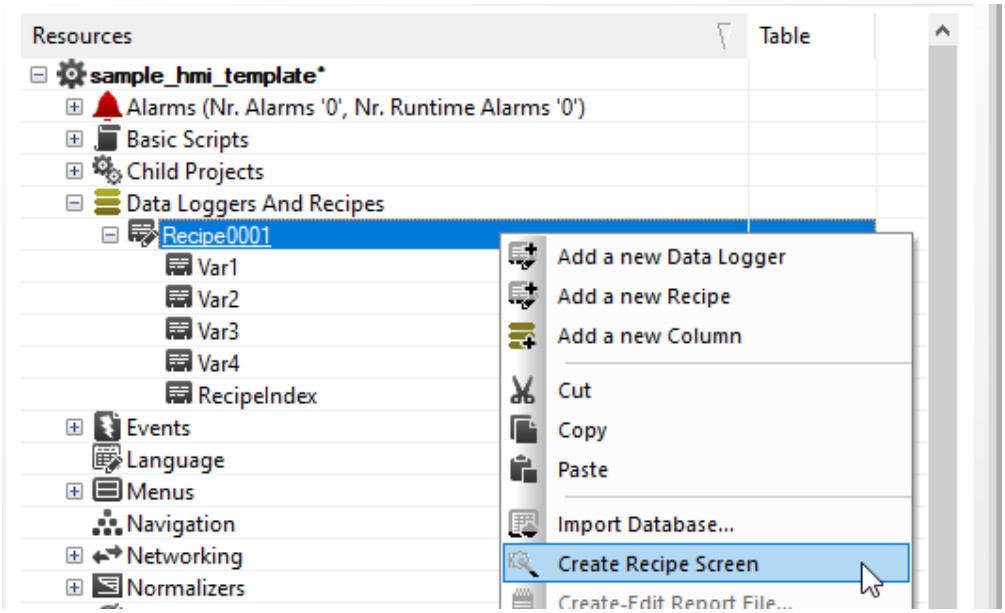


Fig. 5: Create Recipe Screen

A screen with the same name as the recipe will be created and will appear as shown below.

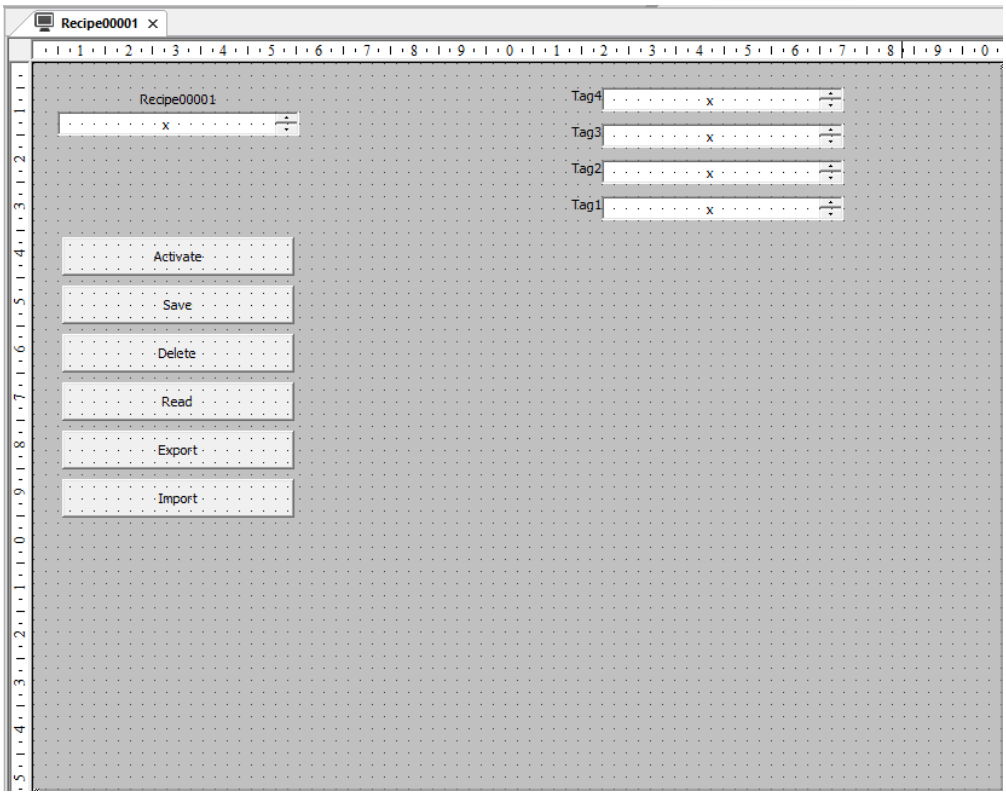


Fig. 6: Recipe Screen Generated

Temporary variables for each of the recipe variables (Var1,. . . ,Var4) are created in the Real Time DB with the suffix “_Temp” added to each. Similarly, a temporary variable for the index is also created. Finally a string variable with the name ” *RecipeName* _ListRecipes” is created. This string displays a list of the recipes that have been saved.

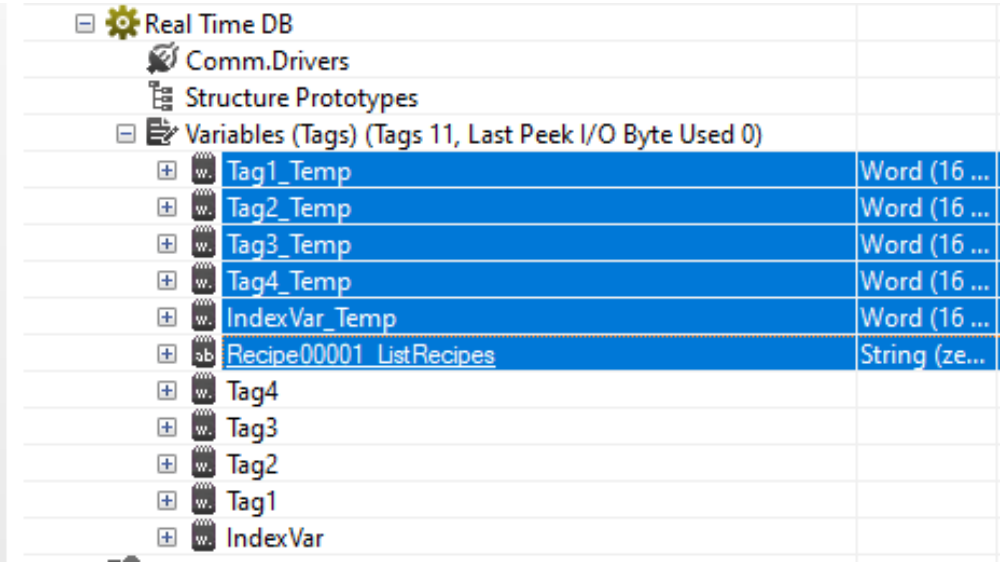


Fig. 7: Generated Recipe Variables

On the generated screen, each of the editbox elements that are labeled with the recipe variable names control and display their respective temporary variable. The editbox element that is labeled with the recipe name controls the index variable for the recipe. Note: if the index variable was defined as a string, this editbox element would be a combobox element. When this index is changed, the temporary variables will also change if a recipe is defined for the index. Each of the button elements correspond to a recipe command – see *Recipe Commands* for a description of each of these commands.

This recipe screen can then be reformatted. The example below labels the temporary index variables with “Temp” for reference. A new section is also created to display the actual recipe variable values and is labeled as such. Finally, a text element is added to display the string of recipes that have been saved for the operator’s reference.

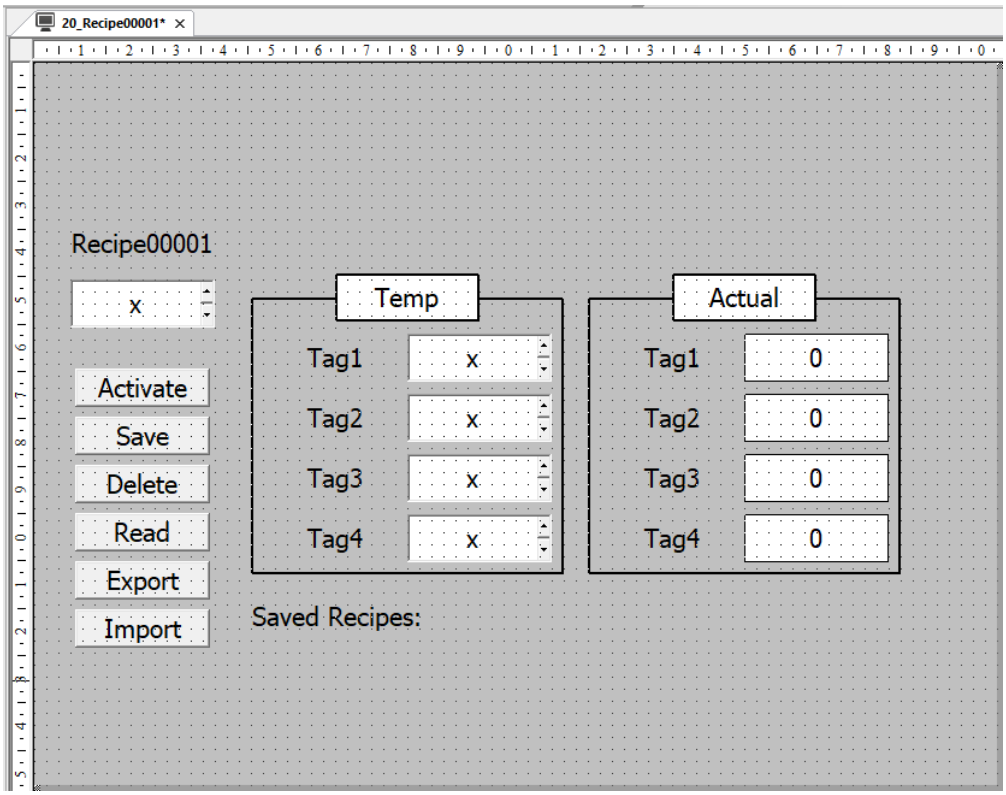


Fig. 8: Reformatted Recipe Screen

If variables are added to the recipe after the recipe screen has been created, the recipe screen should be recreated – the previously created screen will not automatically update.

Recipe Manager

Create a blank screen. Locate the *Recipe Manager* element in the Toolbox window and click-drag it onto the screen to add the element.

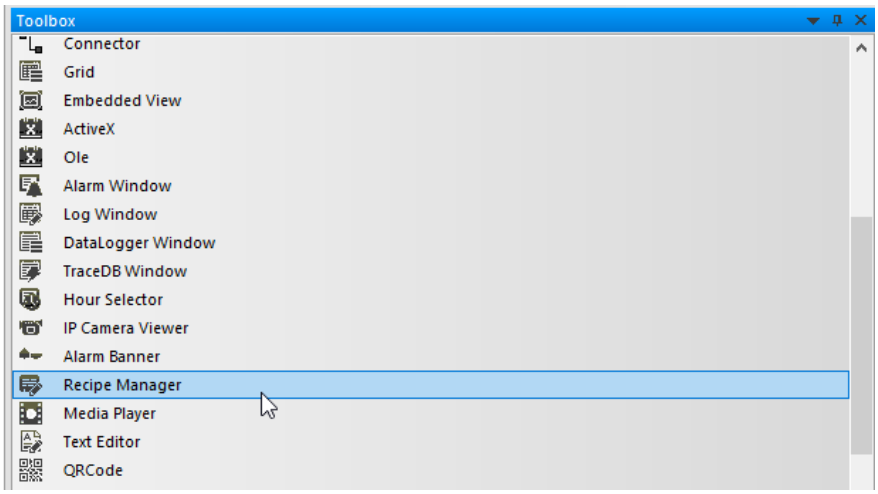


Fig. 9: Recipe Manager in Toolbox Window

The Recipe Manager window combines all of the text, editbox elements, and command buttons into a simplified UI. The recipe index is displayed with the combobox element at the top of the window. The temporary variable names and values are displayed in the data logger part of the window. Finally, the command buttons have been given symbols instead of text.

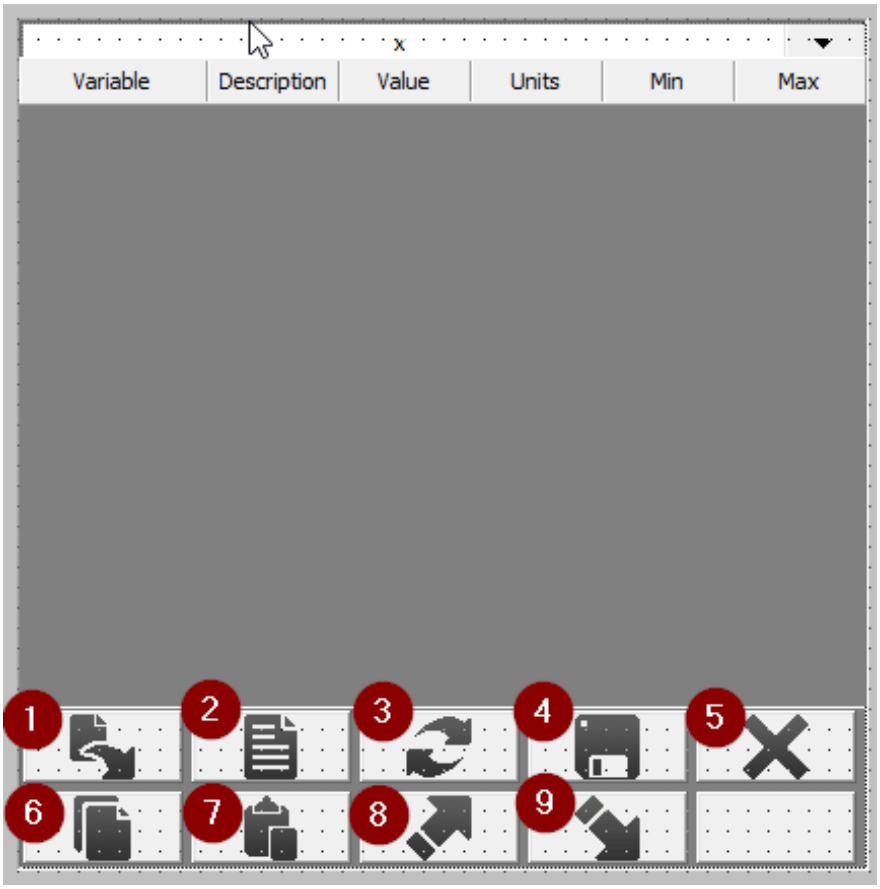


Fig. 10: Recipe Manager Window Layout

These graphic buttons correspond to (1) activate, (2) read, (3) refresh, (4) save, (5) delete, (6) copy, (7) paste, (8) import, and (9) export. If some of these buttons are not needed, they may be deactivated in the *Style* properties of the window. Also, text may be displayed for the buttons rather than symbols by deselecting the *Graphic Buttons* property in the *Style* properties of the window.

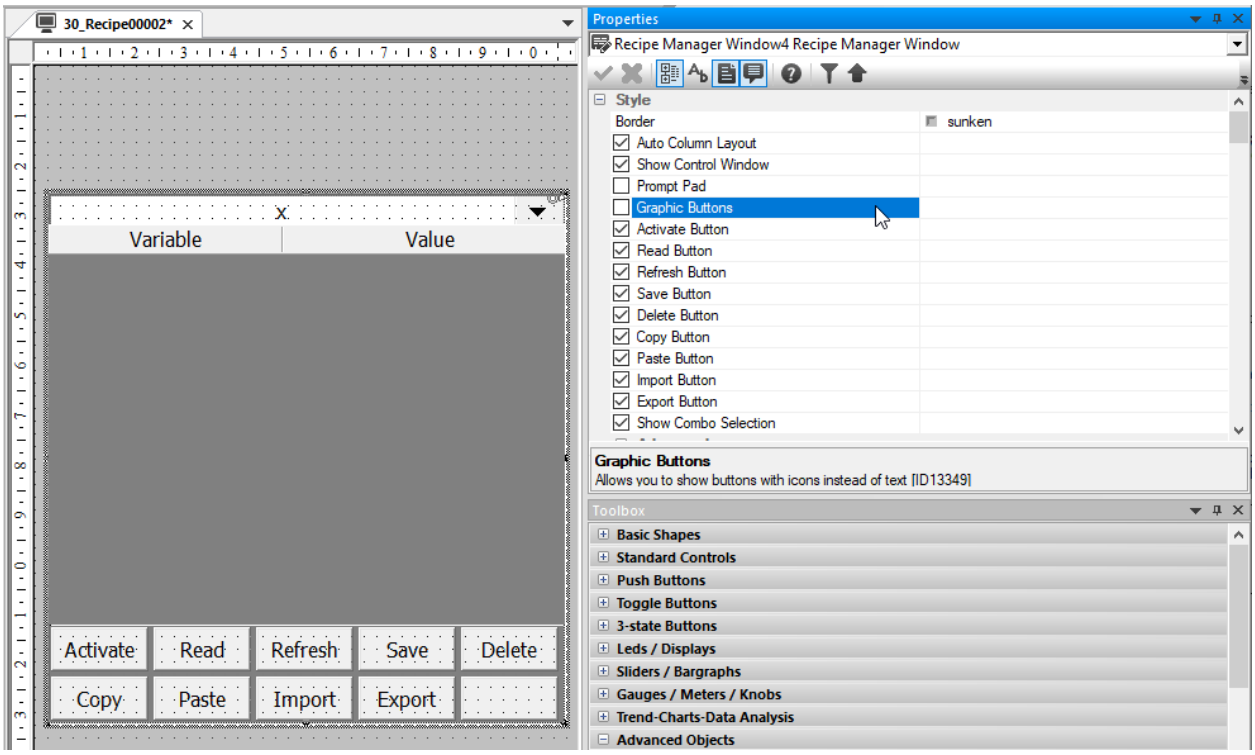


Fig. 11: Recipe Manager with Text Buttons

In the *Execution* properties of the Recipe Manager window, select the *Recipe* that is to be managed by the window.

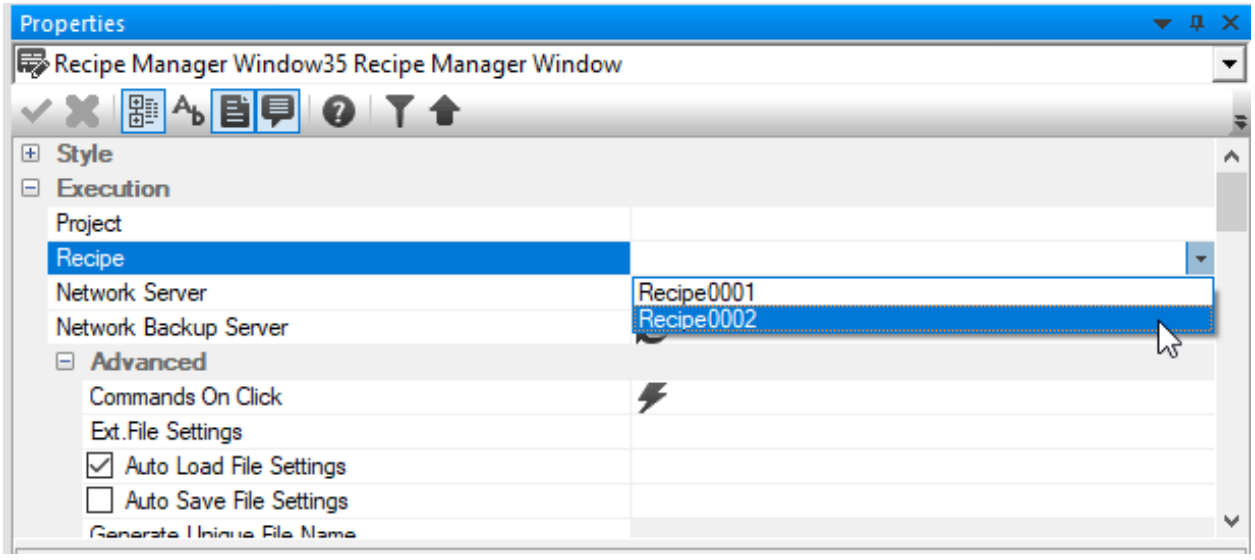


Fig. 12: Select Recipe Manager Window Recipe

There are many additional settings to format the window including displaying a confirmation message for commands, renaming columns, button size, font, etc.

The columns in the window may be removed by clicking on the column and dragging them into the *Field Choice* window that appears. To only show the variable name and value, remove the *Description*, *Units*, *Min*, and *Max* columns.

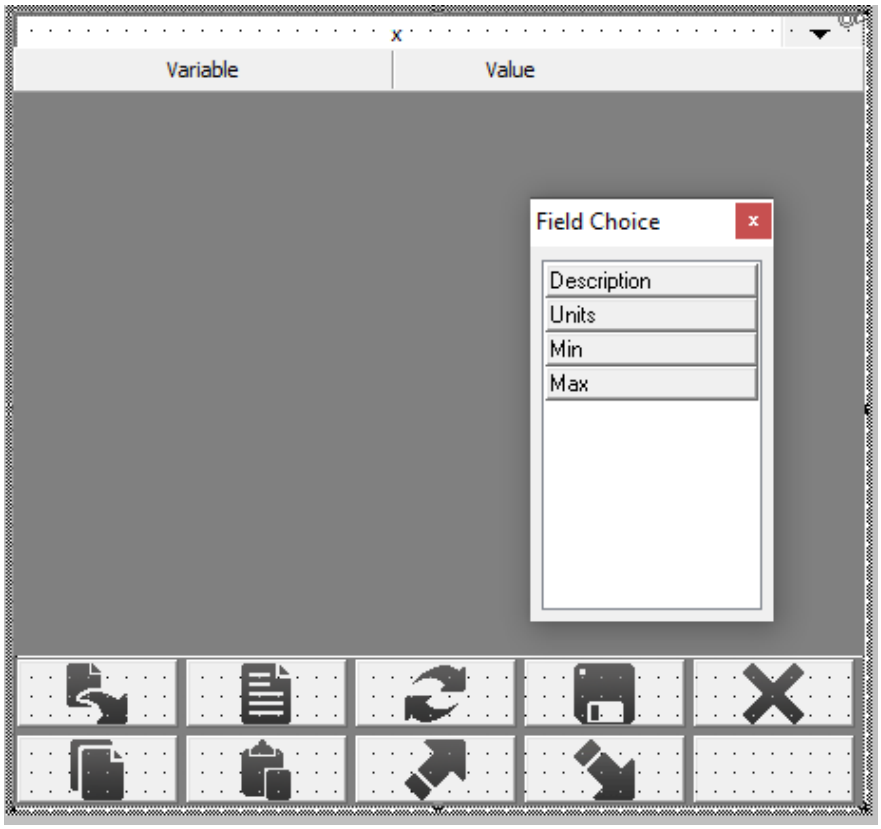


Fig. 13: Edit Recipe Manager Window Columns

Since a list recipes variable is not automatically generated in the Real Time DB for a Recipe Manager window, a string variable may be manually defined and linked to the *List Recipes Variable* in the *Execution* properties of the recipe object. As with the recipe screen in the previous section, elements are added to display the actual values of the recipe variables.

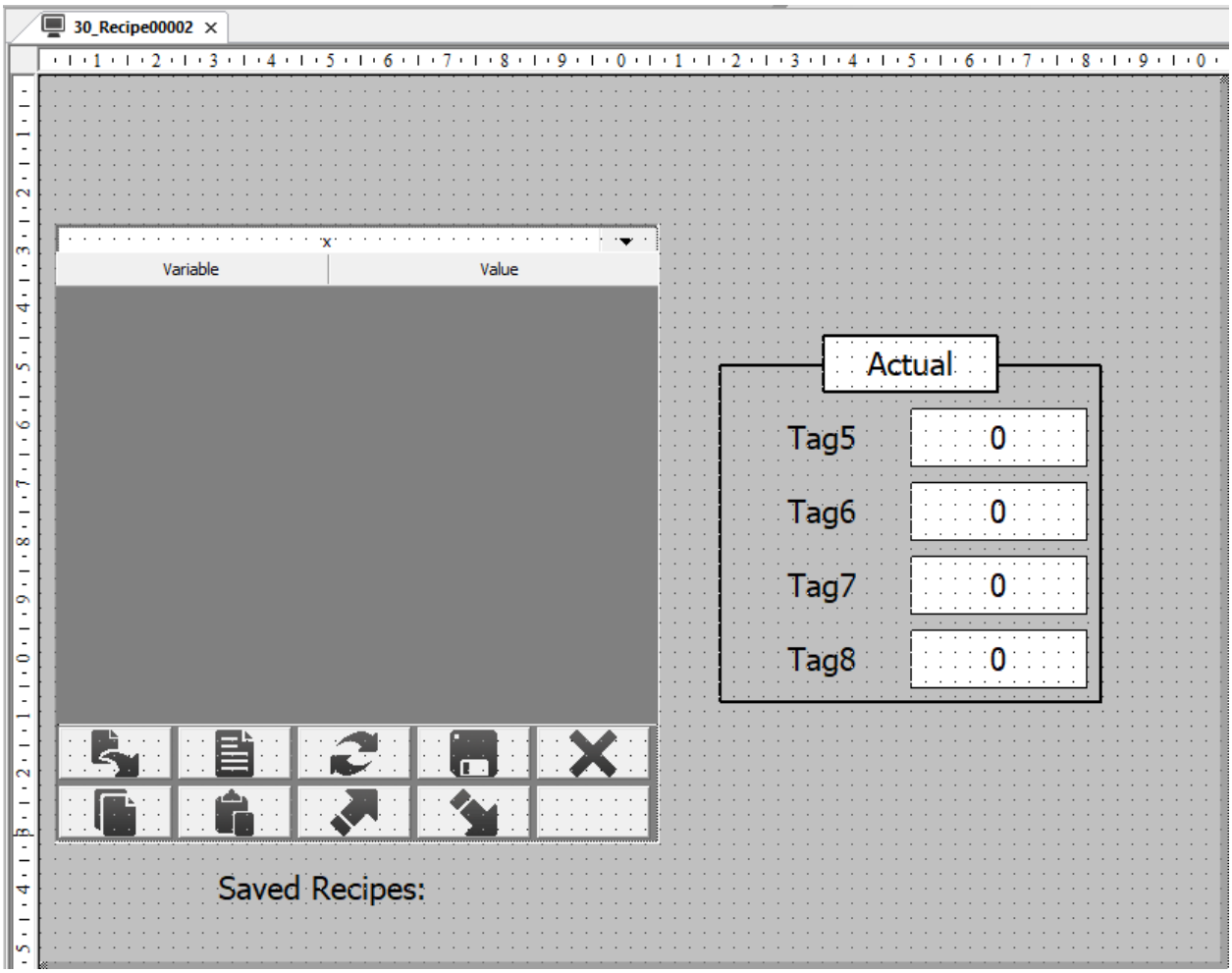


Fig. 14: Actual Values and List Recipes Added

If the recipe that is handled by the Recipe Manager is edited, the Recipe Manager will automatically update.

Recipe Execution

This section explains how recipes are managed during runtime. The first two sections describe how user inputs differ based on the methods chosen to manage the recipe (i.e. index variable type and Recipe Manager vs. Recipe Screen). The sections that follow demonstrate recipe commands with a recipe screen, however the same behavior is shared by the Recipe Manager window element.

Define and Change Recipe Index

For a recipe with an integer type recipe index, the active index is edited via the editbox element either with the spin buttons or by typing a value in the field. If a saved recipe exists at the selected index, the temporary variable values for the recipe will be displayed.

For a recipe with a string type index variable, a new index is added by typing the name of the recipe in the combobox element.

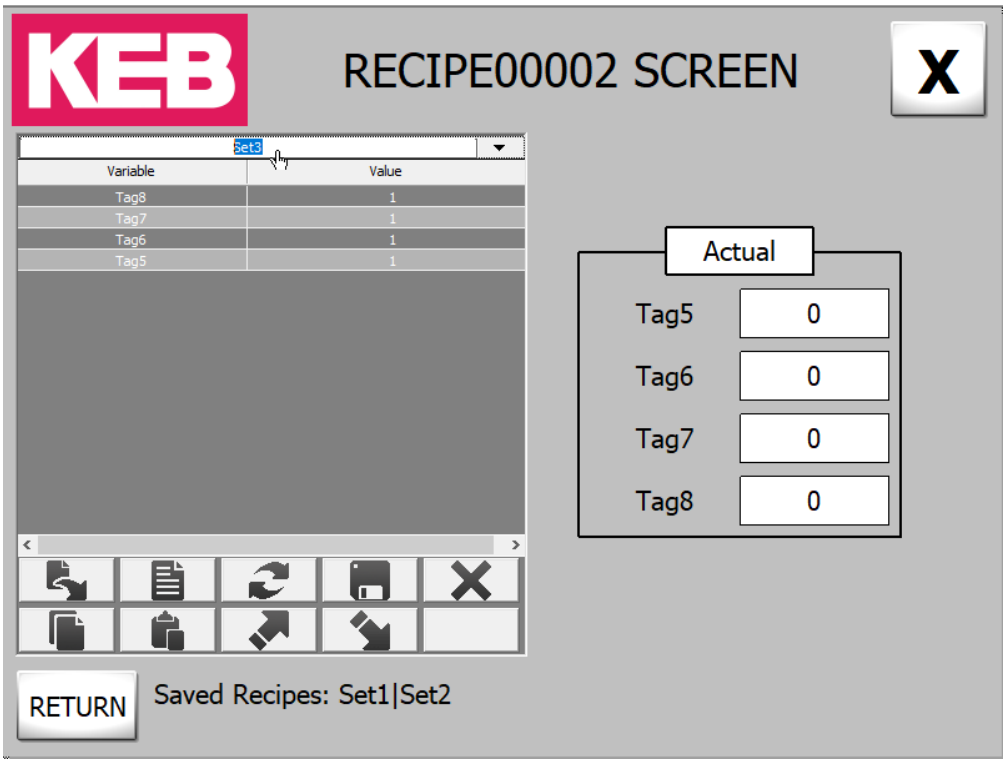


Fig. 15: Define New String Index in Recipe Manager

Selection of a saved recipe can be performed by clicking the down-arrow on the right-side of the combobox.

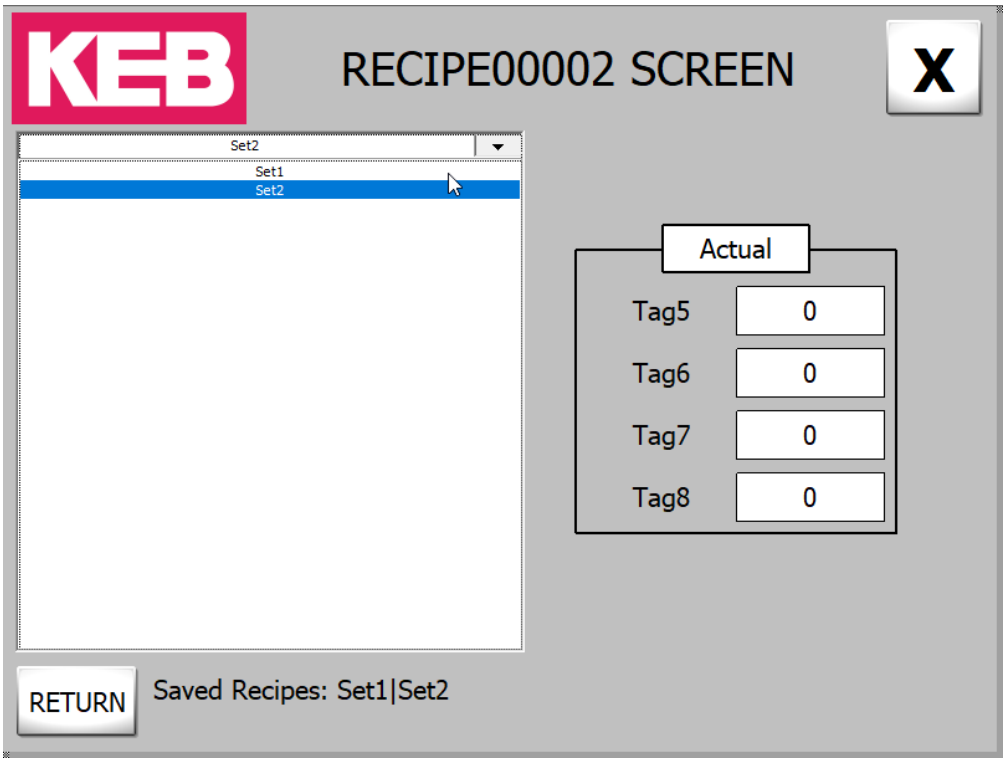


Fig. 16: Combo Box Selection of Recipe Index in Recipe Manager

Edit Recipe Variables

For a recipe managed by a recipe screen, temporary variable values are written via the editbox elements with the spin buttons or by typing a value in the editbox.

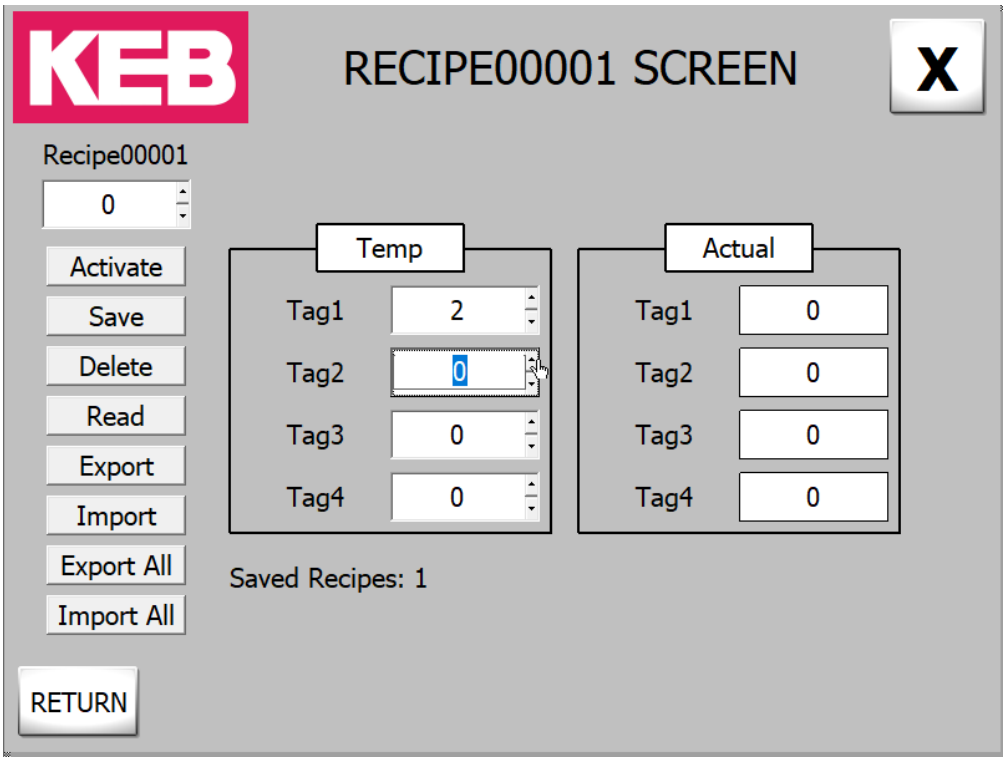


Fig. 17: Edit Temporary Value in Recipe Screen

For a Recipe Manager object, temporary variable values for an index are written by clicking on the value in the window and typing a new value.

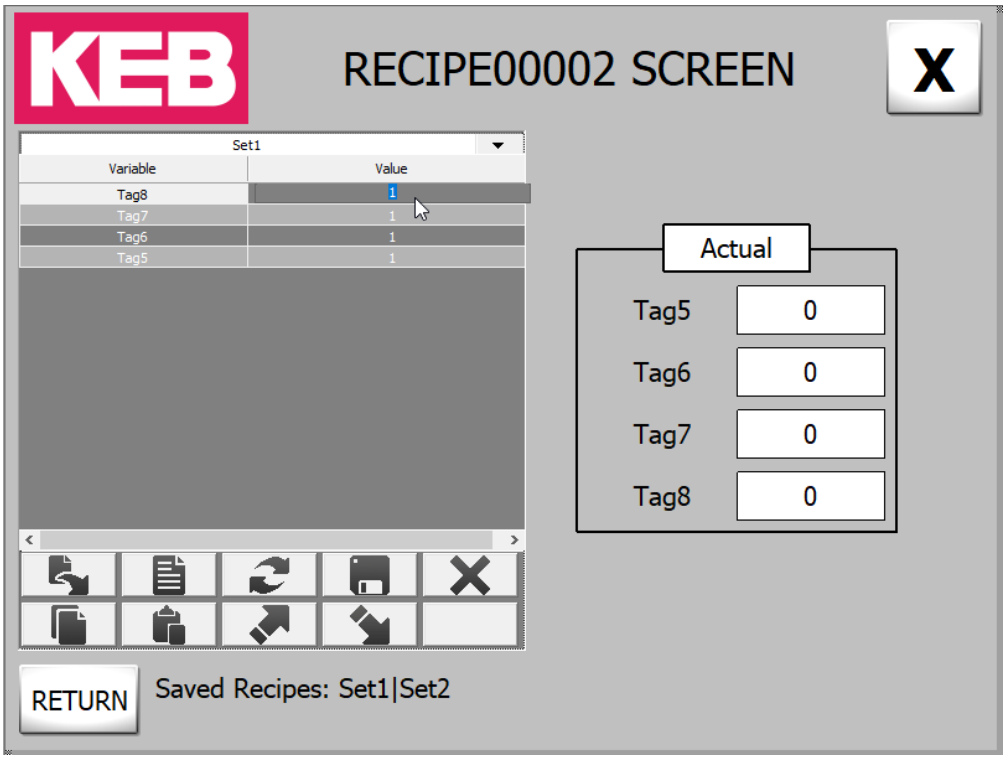


Fig. 18: Edit Temporary Value in Recipe Manager

Save Command

A *Save* recipe command saves the temporary recipe values at the active recipe index. For the example below, the recipe index is set to 1. The temporary variables are then assigned values.

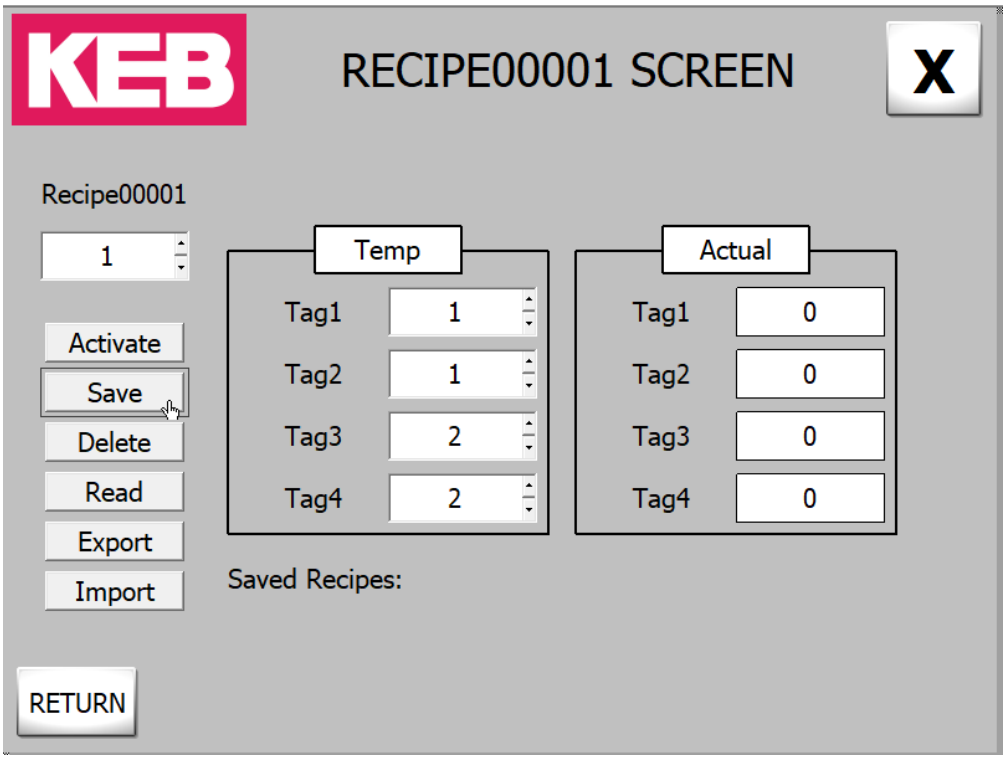


Fig. 19: Before Save Recipe Command

The save command is then executed.

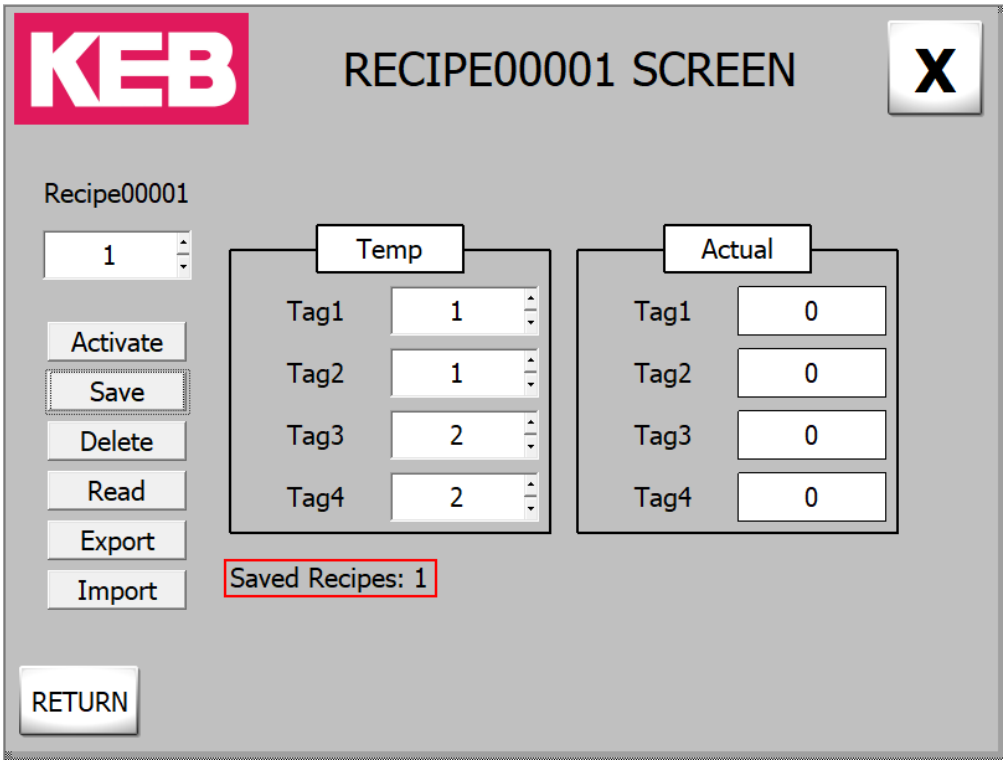


Fig. 20: After Save Recipe Command

The index may now be changed and other recipes may be defined. As long as the recipe at index 1 is not deleted, the saved values will appear when index = 1. Since recipes are stored as a database on the HMI device, all saved recipes will be retained through power cycles and opening/closing of the projects.

Activate Command

An *Activate* recipe command sets the actual value of each recipe variable to the value of its temporary variable defined at the active recipe index (i.e. Actual = Temp).

The previously saved recipe at index 1 is then activated.

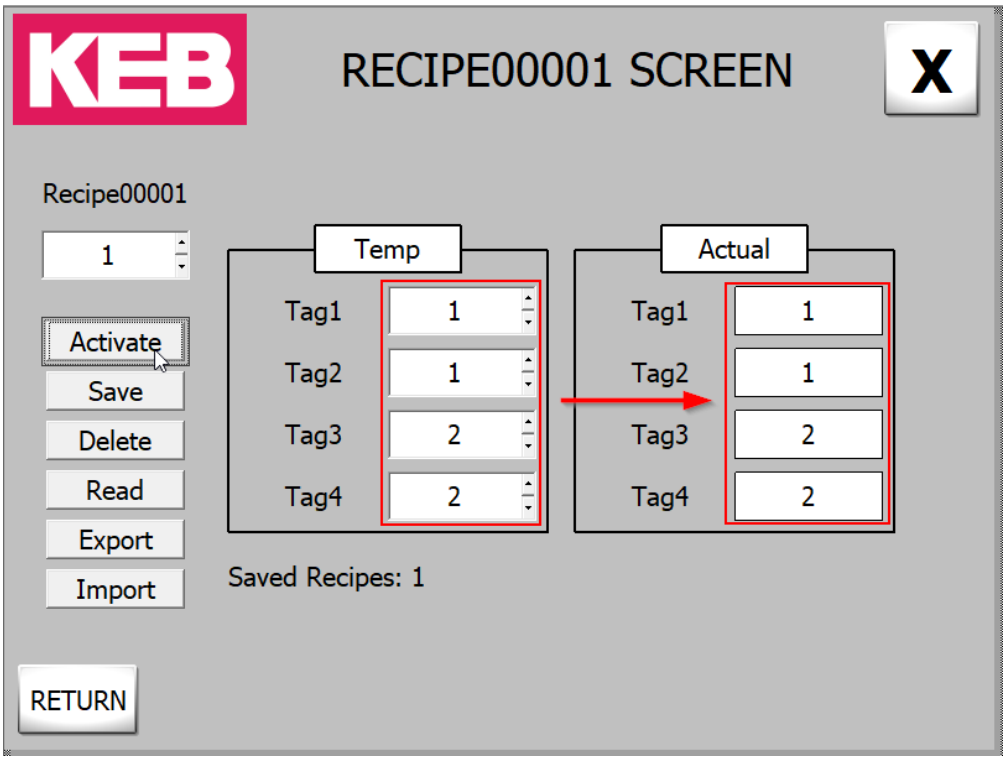


Fig. 21: After Recipe Activate Command

Delete Command

A *Delete* recipe command removes the temporary values for the active recipe index if the recipe has been previously saved.

In the example below, the previously saved and activated recipe is deleted. The recipe is no longer listed in the *Saved Recipes* field.

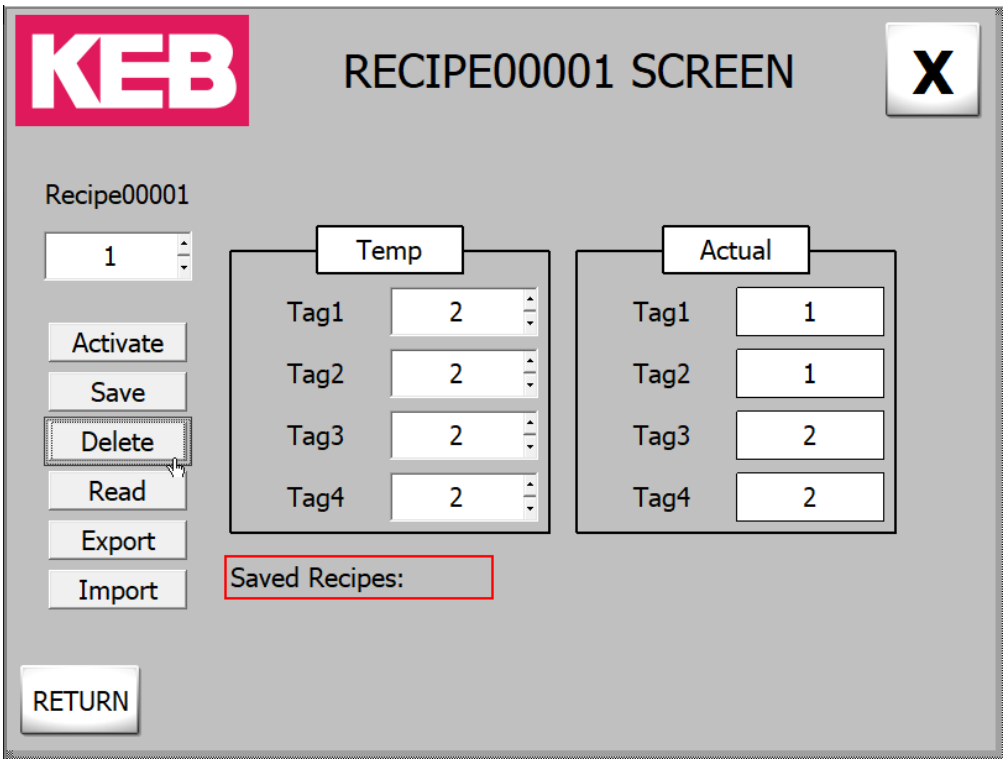


Fig. 22: After Recipe Delete Command

The value of the first two temporary variables are then changed to demonstrate the read command.

Read Command

A *Read* recipe command sets the temporary recipe variables at the active index equal to the actual recipe variables (i.e. Temp = Actual).

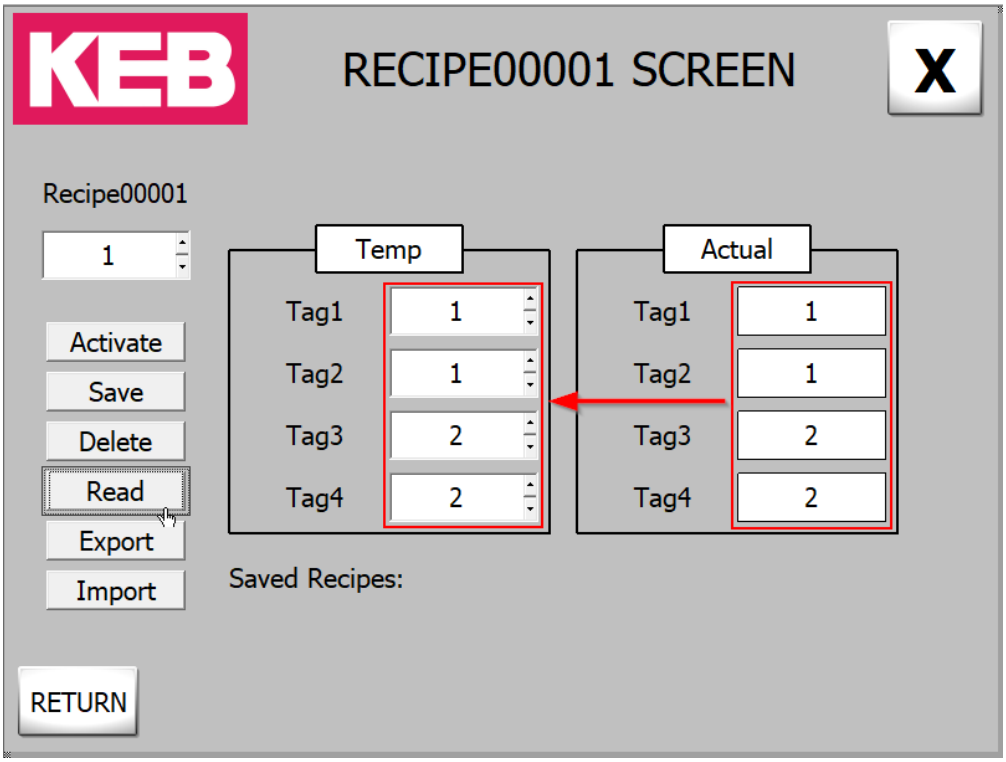


Fig. 23: After Recipe Read Command

NOTE: these read values must then be saved to change the recipe at the index.

Export Command

An *Export Recipe* command creates a .csv archive of the recipe at the active index. This command can be edited. A file path may be defined on the HMI device for the archive to be saved in (e.g. \MMCMemory\Recipes\). If the setting is defined, the directory must be created in the target device before recipes may be saved there – the location will not be automatically created.

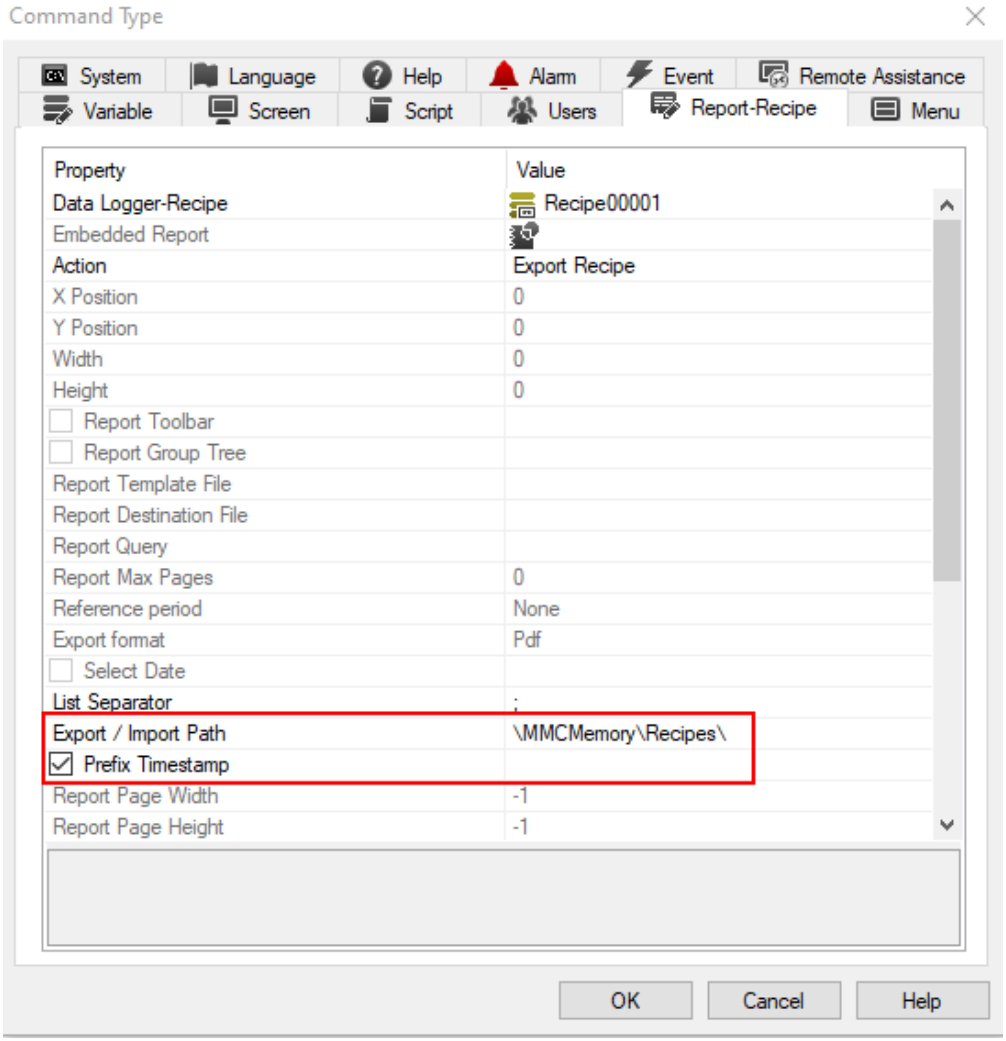


Fig. 24: Export Settings

Additionally, a timestamp prefix may be added automatically (formatted as YYYYMMDD_HHMMSS) to the .csv filename. The delimiter may also be altered with the *List Separator* field from the default of ';'.

Execute the export command by clicking the button. A file browser prompt will appear when the command is executed. The browser will be opened in the set file location.

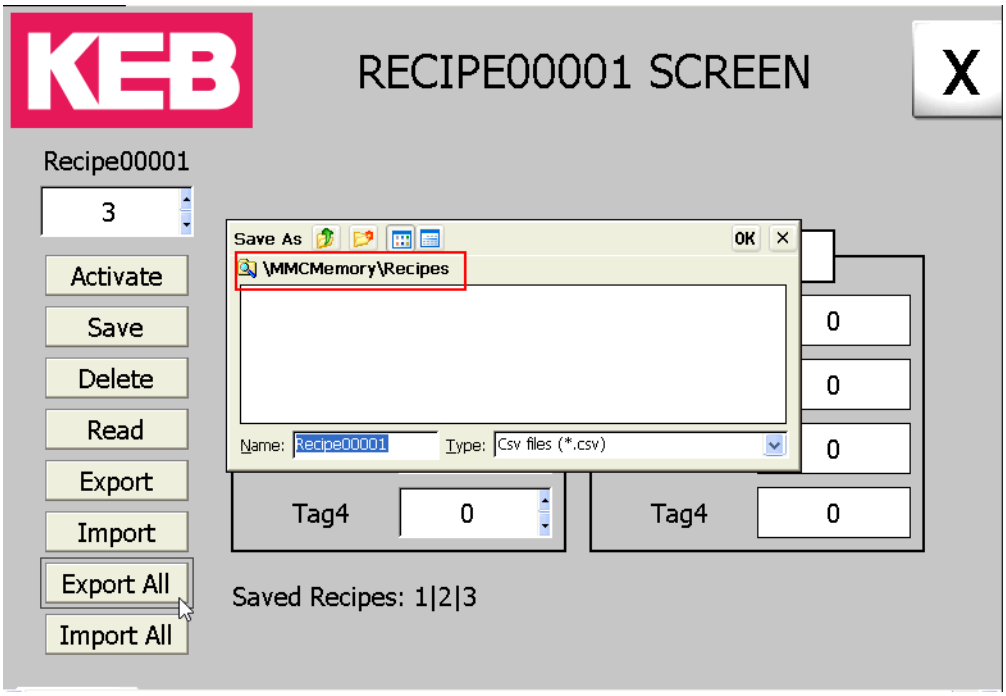


Fig. 25: Export with Defined Folder

The user must still define a name for the .csv. Alternatively, the file path may be set to include a .csv filename in (e.g. \MMCMemory\Recipes\Recipe00001.csv). If this is done, a file browser will not appear and the archive will be saved automatically.

The default file path for the archive is the *DATA* folder of the project on the device. This is the location that will be opened if the location is not set in the command settings.

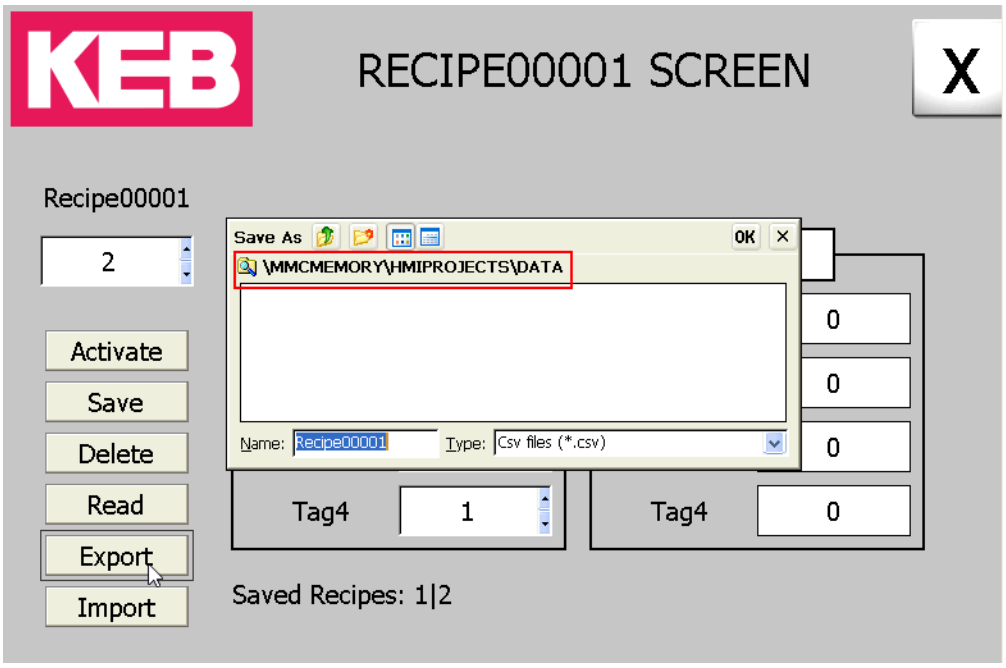


Fig. 26: Export Command Execution in Runtime with Default Folder

The timestamp prefix will not appear in *Name* field if the setting is enabled, but will be added to the filename when saved.

Define an alternate name if desired and select *OK*. The .csv file is now in the HMI device file system.

Fig. 27: Recipe Saved with Timestamp

Pictured below is the .csv file that was generated by the command.

	A
1	Variable;Description;Value;Units;
2	Tag4;;1;;
3	Tag3;;1;;
4	Tag2;;2;;
5	Tag1;;2;;

Fig. 28: Exported .csv File

Import Command

An *Import Recipe* command imports a .csv file into the active recipe index. Like the export command settings, the import command may also be edited to search for the .csv file in a file location that is not the default DATA directory.

To test the import recipe command, the previously exported .csv file can be edited and saved on the HMI device.

	A
1	Variable;Description;Value;Units;
2	Tag4;;2;;
3	Tag3;;2;;
4	Tag2;;2;;
5	Tag1;;2;;

Fig. 29: Edited .csv File

Execute the import command and locate the edited csv file. Select it in the file browser and hit *OK*.

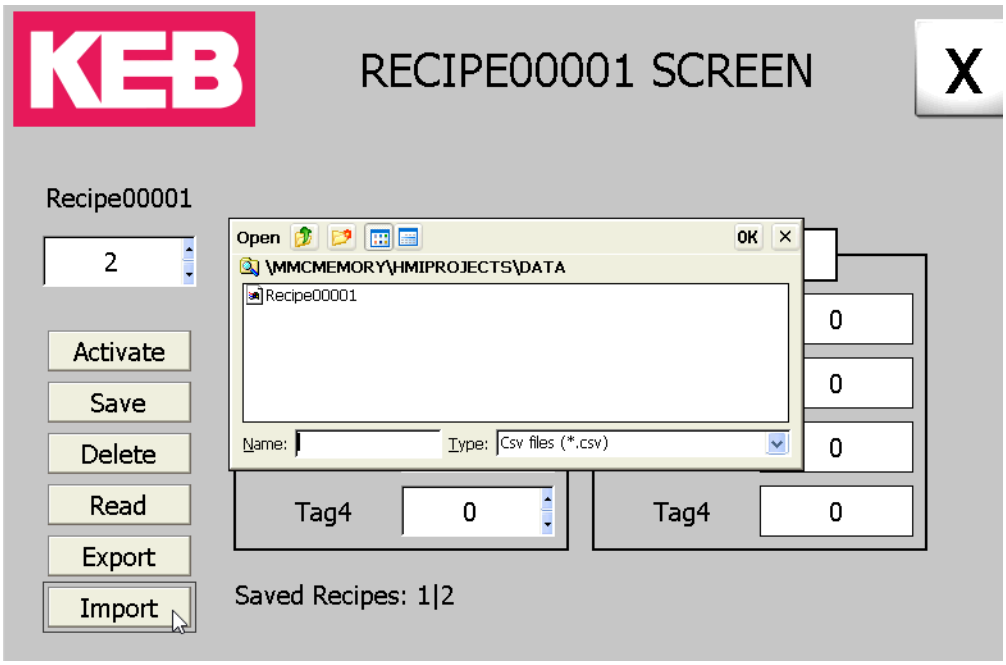


Fig. 30: Import Edited Recipe in Runtime

The temporary values at the active index are now set to the values defined in the edited .csv file.

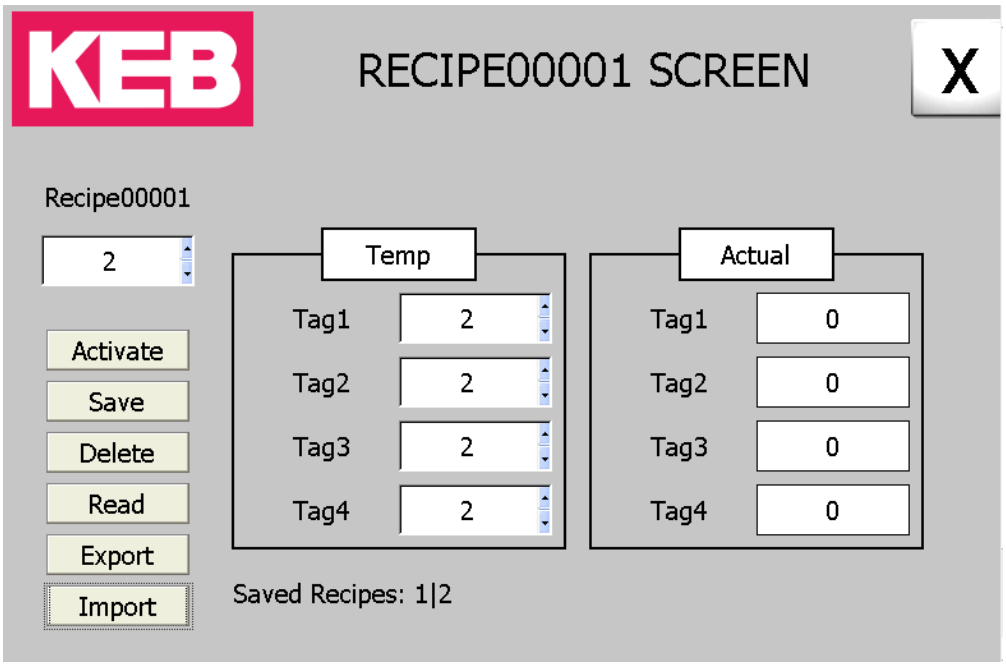


Fig. 31: Edited Recipe Imported

NOTE: as with the read command, a save command must be executed after the import command to change the recipe at the index.

Export All Command

The *Export All Recipe Records* command created a .csv archive of ALL saved recipes, rather than just that of the active recipe index. The same settings are available for this command as with the previous export command. Similar to the import and export commands that are automatically generated on the screen, *Export All Recipe Records* and *Import All Recipe Record* commands may be defined manually.

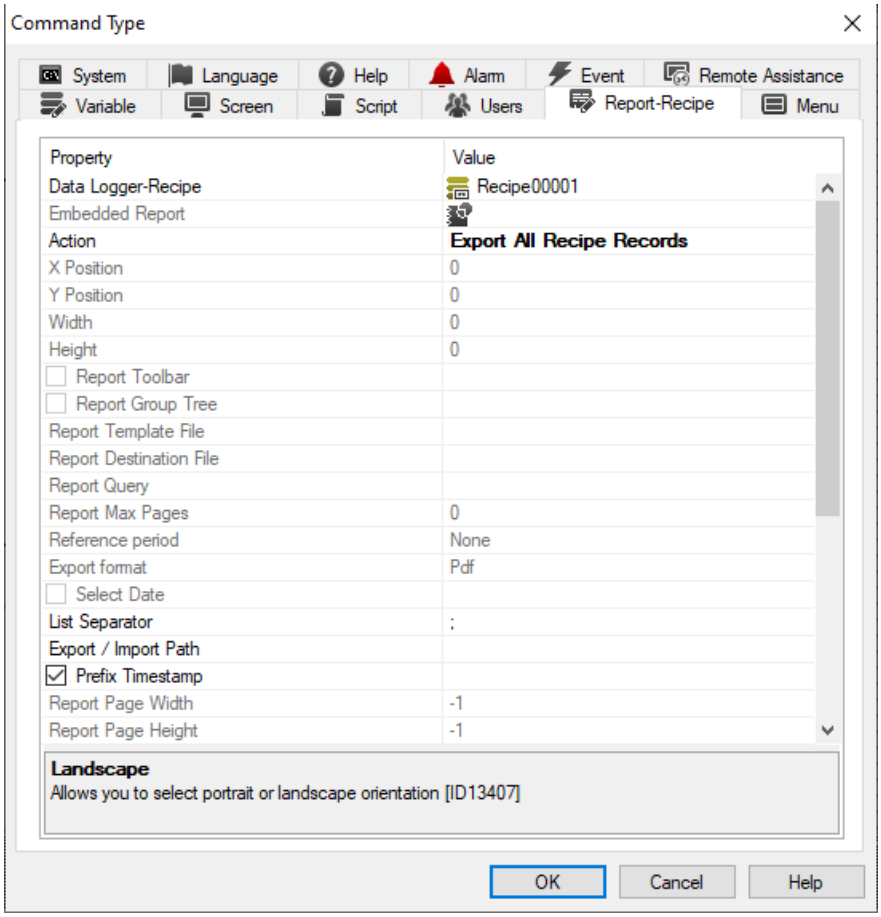


Fig. 32: Define Export all Records Command

These commands may be assigned to pushbuttons just as all of the automatically generated commands are.

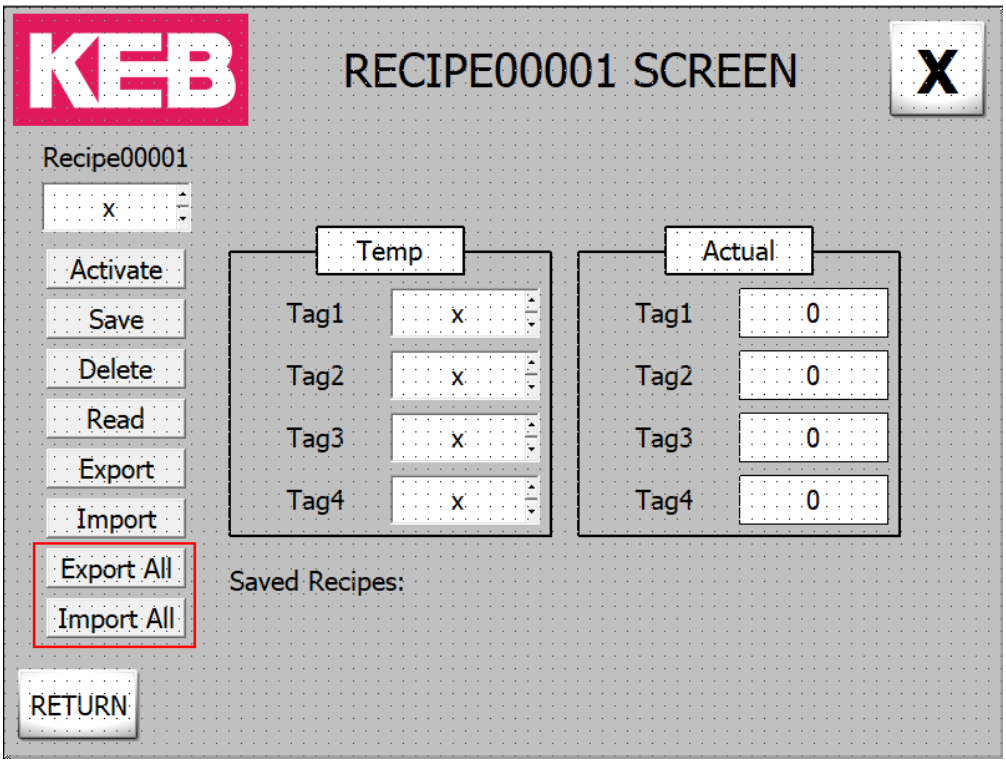


Fig. 33: Import/Export All Buttons Added

To test the export all command, define and save a few recipes in the runtime. Select the *Export All* button.

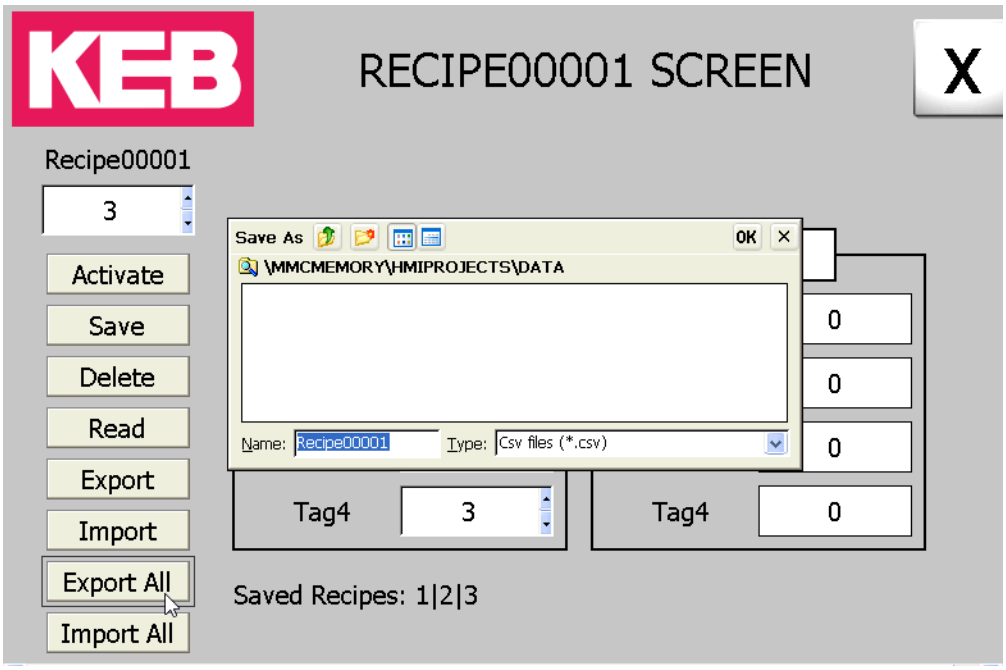
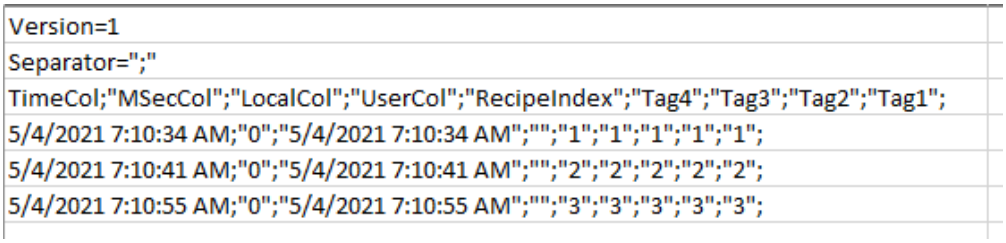


Fig. 34: Export All Command in Runtime

Navigate to the file destination and rename if desired. Select *OK*. The .csv file now contains all saved recipes as shown below.



```
Version=1
Separator=";"
TimeCol;"MSecCol";"LocalCol";"UserCol";"RecipeIndex";"Tag4";"Tag3";"Tag2";"Tag1";
5/4/2021 7:10:34 AM;"0";"5/4/2021 7:10:34 AM";"";"1";"1";"1";"1";"1";
5/4/2021 7:10:41 AM;"0";"5/4/2021 7:10:41 AM";"";"2";"2";"2";"2";"2";
5/4/2021 7:10:55 AM;"0";"5/4/2021 7:10:55 AM";"";"3";"3";"3";"3";"3";
```

Fig. 35: Export All Recipe Records Csv

Import All Command

The *Import All Recipes* command may be configured similar to the regular *Import* command. To test the import all command, delete all of the recipes that have been defined in the runtime.

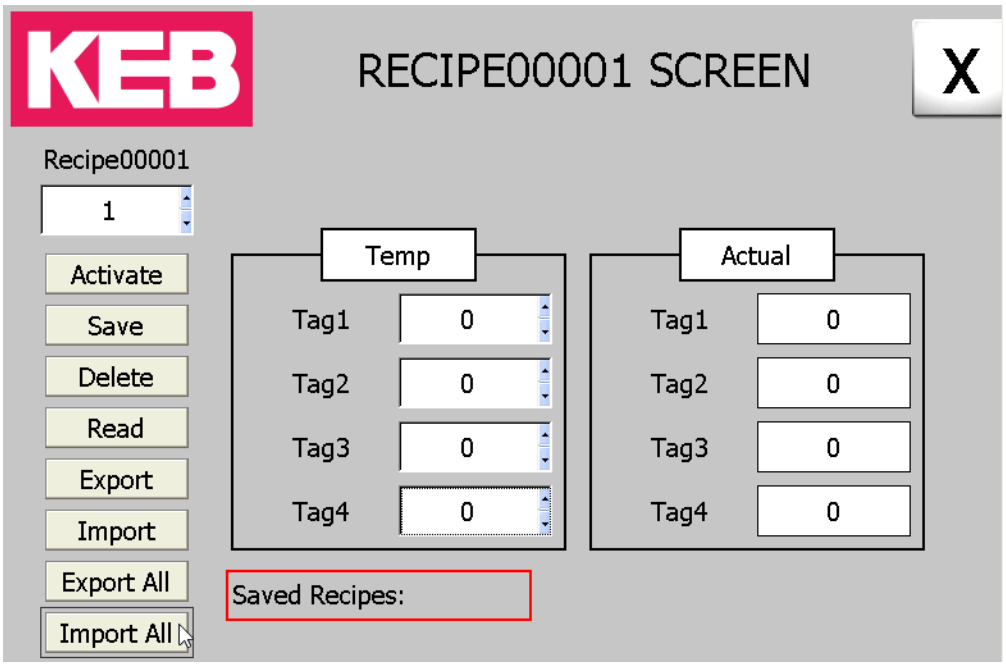


Fig. 36: All Recipes Deleted

Next, select the *Import All* button. Locate the previously saved archive in the file browser, select it, then select *OK*.

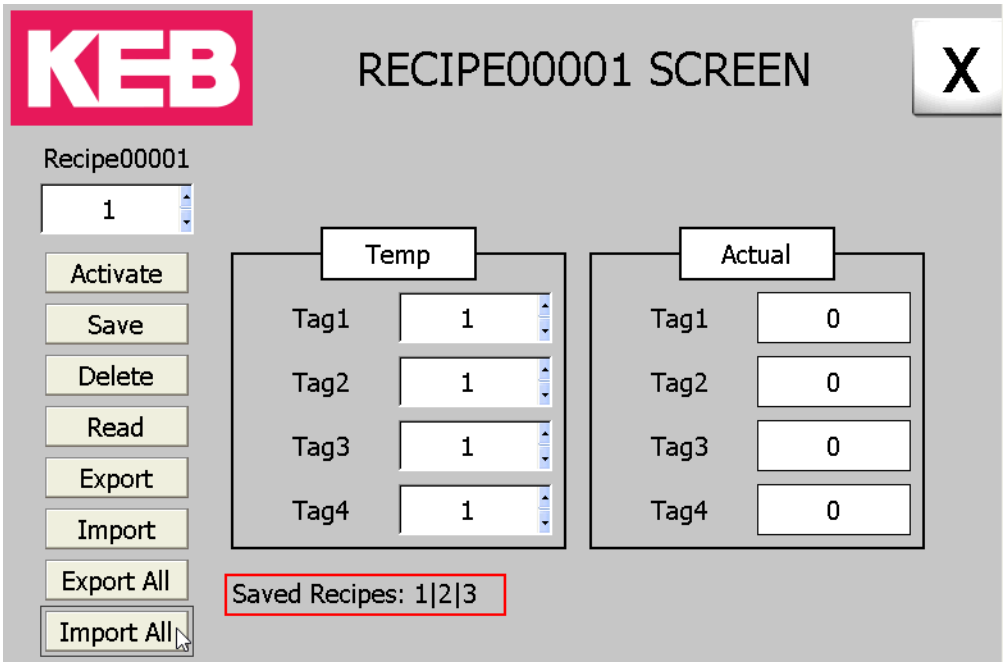


Fig. 37: All Recipes Imported

All of the recipes that were archived and deleted in the runtime are now again available for use after the import all command. Unlike the regular import command, these imported recipes are automatically saved.

Refresh Command (Recipe Manager Only)

The refresh button available in the Recipe Manager window resets the temporary values that are displayed if they have been altered from their saved value. NOTE: this does not apply if the index value has not been saved. This command is useful if the user is editing a recipe and would like to restart the editing process if an error is made.

Troubleshooting

This section explains common methods of troubleshooting for recipes that are not behaving as intended.

Create DB Table

Recipes are saved in the project directory as a database. When variables are added to or removed from the project, the database may need to be rebuilt if the recipe stops working correctly.

To force a rebuild of the recipe that is not working, navigate to the recipe's properties and go to *Database options* > *ODBC Manager* and select the '. . . ' icon next to *Create DB Table*.

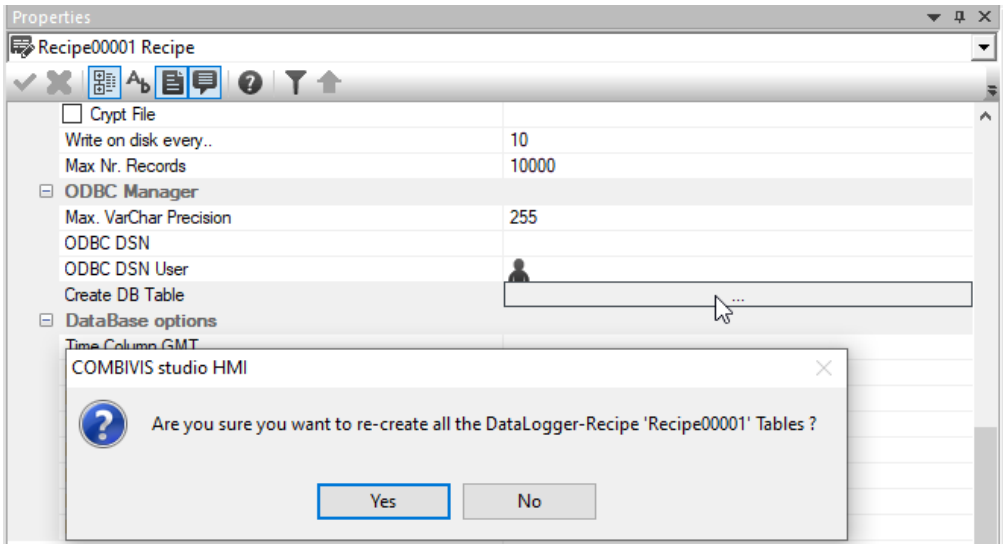


Fig. 38: Create DB Table in Recipe Properties

In the prompt that appears, select *Yes*. The database for the selected recipe has now been rebuilt. Save and rerun the project to test if the recipe is operating as expected.

Disclaimer

KEB America, Inc. reserves the right to change/adapt specifications and technical data without prior notification. The safety and warning reference specified in this manual is not exhaustive. Although the manual and the information contained in it is made with care, KEB does not accept responsibility for misprint or other errors or resulting damages. The marks and product names are trademarks or registered trademarks of the respective title owners.

The information contained in the technical documentation, as well as any user-specific advice in verbal or in written form are made to the best of our knowledge and information about the application. However, they are considered for information only without responsibility. This also applies to any violation of industrial property rights of a third-party.

Inspection of our units in view of their suitability for the intended use must be done generally by the user. Inspections are particular necessary, if changes are executed, which serve for the further development or adaption of our products to the applications (hardware, software or download lists). Inspections must be repeated completely, even if only parts of hardware, software or download lists are modified.

Application and use of our units in the target products is outside of our control and therefore lies exclusively in the area of responsibility of the user.

Americas:

KEB America, Inc.
5100 Valley Industrial Blvd South
Shakopee, MN 55379, USA
(+1) 952-224-1400
info@kebamerica.com

Headquarters:

KEB Automation KG
Suedstrasse 38
D - 32683 Barntrup, Germany
(+49) 5263 401-0
info@keb.de