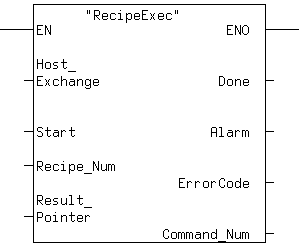
The function block ‘RecipeExec’ (recipe executor) is built to help a user to prepare the ‘Matrix’ reader configuration, using a PLC Siemens via FiendBus (Profibus or Profinet).

This function block process the ‘forms’ (Every command have a precompiled form stored in as many UDTs) read in a specific (see below) Data Block (DB) and compile the corresponding command strings to send at partner, using DAD\_DPD function block.

Every command string will stored in a buffer (the ‘DAD\_DPD’ function ‘Tx\_Buffer’), and the appropriate command will be set in order to command ‘DAD\_DPD’ function the send the string at partner.

After the partner answer (via ‘DAD\_DPD’, with string stored in ‘Rx\_Buffer’), ‘RecipeExec’ will analyze the received string and will set a Alarm if answer is different than correct one, or continue with next command until the recipe finish.

Below the description of correct Function block connection:



0

D

C

B

A

4

3

2

1**Errore. L'origine riferimento non è stata trovata.Errore. L'origine riferimento non è stata trovata.**

**Inputs description**

1. EN (BOOL)

This is the function block enable. At the moment this function block is strictly linked to ‘DAD\_DPD’ function block, is suggested to put TRUE this input only when:

* ‘DAD\_DPD’ function block ‘Enable’ is TRUE
* ‘DAD\_DPD’ function block ‘Alarm’ is FALSE

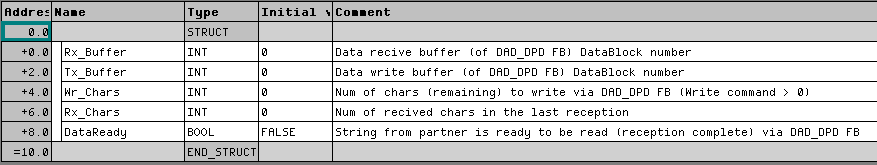
In any case, if ‘DAD\_DPD’ function block isn’t ready to send or receive messages, ‘RecipeExec’ will wait for ‘DAD\_DPD’ ready.

1. Host\_Exchange (POINTER)

Function block ‘RecipeExec’ need ‘DAD\_DPD’ function block to send (and receive) commands at partner. In order to link ‘RecipeExec’ with ‘DAD\_DPD’, a structured variable as been designed. This variable contain various signal to communicate order and read signals with ‘DAD\_DPD’.

Then size of this structured variable is 10bytes, so isn’t possible to connect it directly at the function block and for this reason it is linked via pointer. This pointer indicate the variable begin address and it MUST be specified otherwise the function recognize a bad address and the CPU will commutate in STOP. A default address is purposely omitted to avoid unintentional memory overwrite.

‘HostExchange’ structure is showed below:



1. Start (BOOL)

This Boolean variable commands ‘RecipeExec’ function block to execute specified command set. The command execution will continue while ‘Start’ is TRUE (until command set completed or a alarm occur), and it will abort if ‘Start’ is FALSE.

1. Recipe\_Num (INT)

This is the number of data block (DB) where the command set is stored. This variable value is check at the function activation (transition from FALSE to TRUE of ‘Start’), and if it don’t correspond at a valid and existing DB a alarm is set.

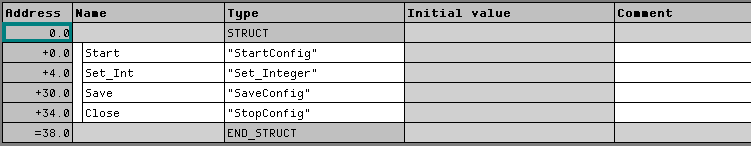
The indicated DB must be filled using precompiled forms, in order to declare the command list, and every command must be set with correct values (see specific documentation).

Possible (actually built) commands are listed below:

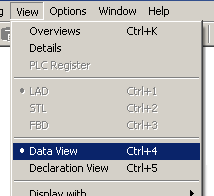
|  |  |  |
| --- | --- | --- |
| Command name | Command internal code | UDT number |
| StartConfiguration | 1 | 111 |
| RestoreDefault | 2 | 112 |
| Set\_Integer | 3 | 113 |
| Set\_Enumeration | 4 | 114 |
| Set\_String | 5 | 115 |
| Set\_BitField | 6 | 116 |
| Set\_AtLastOne | 7 | 117 |
| Toggle\_ButtonFunction 1 | 8 | 118 |
| Toggle\_ButtonFunction 2 | 9 | 119 |
| Toggle\_ButtonFunction 3 | A | 120 |
| Toggle\_ButtonFunction 4 | B | 121 |
| Set\_DisableAllSymbologies | C | 122 |
| SaveConfiguration | D | 123 |
| StopConfiguration | E | 124 |

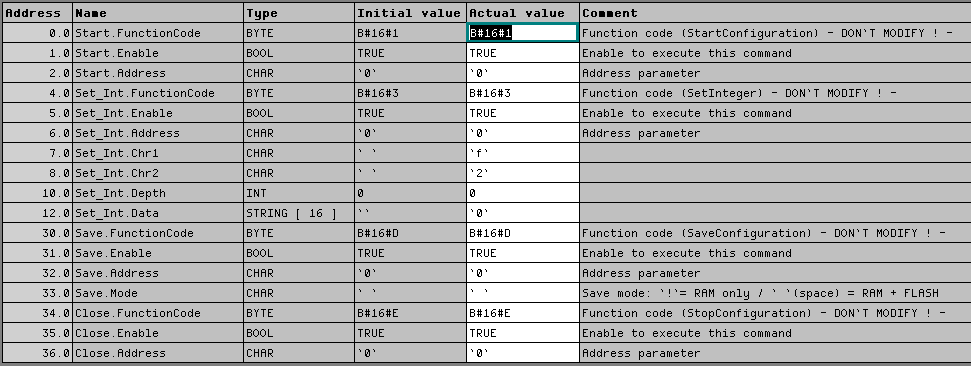
For example:

In a empty DB declare a variable list using UDTs listed above:



then assign correct value at every variable:





In this way, the command list (named ‘Recipe’) is ready to be executed by ‘RecipeExec’.

Function block ‘RecipeExec’ will process every command in this DB (‘Recipe’) if ‘Enable’ (variable present in every precompiled command form) is TRUE, otherwise ‘RecipeExec’ will skip it.

Number of Data block containing ‘Recipe’ (that command list) is the value to assign at the variable linked at input ‘Recipe\_Num’

‘RecipeExec’ match the value of variable ‘Code’ (first one of every precompiled command form) with a internal code who active the corresponding routine. If ‘Code’ is ‘out-of-range’, a alarm is activate.

1. Result\_Pointer (POINTER)

Every time a error occur during the recipe execution, ‘RecipeExec’ return a short description of alarm or, in case error is set from partner, it return first eight characters of partner answer. This input mast be connected at a valid address, and it represent address of first character of error description string. In MUST be connected otherwise the function recognize a bad address and the CPU will commutate in STOP. A default address is purposely omitted to avoid unintentional memory overwrite.

**Output description**

1. Done (BOOL)

At the end of recipe execution, if any error in occurred, ‘RecipeExec’ set output ‘Done’ to indicate the ‘Recipe’ as been successful executed. This signal remain TRUE while ‘Start’ is TRUE.

1. Alarm (BOOL)

When ‘Start’ is TRUE, ‘Done’ is FALSE, and ‘Alarm’ is FALSE, ‘RecipeExec’ is executing indicated Recipe. If a error occur, the function block set TRUE output ‘Alarm’ and abort recipe execution.

This signal will remain TRUE while ‘Start’ is TRUE.

1. ErrorCode (DWORD)

This output is a coded error description. List of possible errors is shoed at the end of this document. ErrorCode is coded in tree bytes with following meaning:

* 1st byte (left): Number of command in execution (sequence of recipe order)
* 2nd byte (center): Code of command in execution (first variable of every command form)
* 3st byte (right): Error description (see Alarm list)

1. Command\_Num (INT)

It is the number (in sequence of command in DB ‘Recipe’ order) of command in execution. If any arror occur during recipe execution, the output will scan number from 1 to last command number in recipe. If a error occur, it represent the command who caused error.

**Error table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Action | | Error code | Description | Result |
|  | |  |  |  |
| FB interface control | | FD | Recipe DB error: not present or 'UNLINKED' attribute | R\_Nfound |
|  | | | | |
| Error code interpretation | | | | |
| Working | Error code for recipe execution is made in 3 bytes. It meanings is: | | | Answer string from partner (8 characters max.) |
| 1st byte (left): | Number of command in execution (sequence of recipe order) | |
| 2nd byte (center): | Code of command in execution | |
| 3th byte (right): | 1 | First Answer string characters missmatch |
| 2 | Second answer string characters missmatch |
| A0 | Address' out of range | AdrWrong |
| A1 | Depth' out of range | DepthOut |
| A2 | Tx\_Buffer too little to contain 'Data' values | Tx\_Over |
| FE | Unknow function code | UnKnow |
| FF | Software error. (bad function index: overflow) | Sw\_ERROR |
|  | | | | |