APPLICATION EXAMPLE

AC500 V2 FLASH HANDLING

USE ALL BLOCKS OF FLASH FOR DATA STORAGE

Contents

1 Disclaimer

A. For customers domiciled outside Germany /

Für Kunden mit Sitz außerhalb Deutschlands

„Warranty, Liability:

The user shall be solely responsible for the use of this products described within this file. ABB shall be under no warranty whatsoever. ABB's liability in connection with application of the products or examples provided or the files included within this products, irrespective of the legal ground, shall be excluded. The exclusion of liability shall not apply in the case of intention or gross negligence. The present declaration shall be governed by and construed in accordance with the laws of Switzerland under exclusion of its conflict of laws rules and of the Vienna Convention on the International Sale of Goods (CISG)."

„Gewährleistung und Haftung:

Der Nutzer ist allein für die Verwendung des in diesem Dokument beschriebenen Produkte und beschriebenen Anwendungsbeispiele verantwortlich.

ABB unterliegt keiner Gewährleistung. Die Haftung von ABB im Zusammenhang mit diesem Anwendungsbeispiel oder den in dieser Datei enthaltenen Dateien - gleich aus welchem Rechtsgrund - ist ausgeschlossen. Dieser Ausschluss gilt nicht im Falle von Vorsatz oder grober Fahrlässigkeit. Diese Erklärung unterliegt Schweizer Recht unter Ausschluss der Verweisungsnormen und des UN-Kaufrechts (CISG)."

B. Nur für Kunden mit Sitz in Deutschland

„Gewährleistung und Haftung:

Die in diesem Dokument beschriebenen Anwendungsbeispiele oder enthaltenen Dateien beschreiben eine mögliche Anwendung der AC500 bzw. zeigen eine mögliche Einsatzart. Sie stellen nur Beispiele für Programmierungen dar, sind aber keine fertigen Lösungen. Eine Gewähr kann nicht übernommen werden.

Der Nutzer ist für die ordnungsgemäße, insbesondere vollständige und fehlerfreie Programmierung der Steuerungen selbst verantwortlich. Im Falle der teilweisen oder ganzen Übernahme der Programmierbeispiele können gegen ABB keine Ansprüche geltend gemacht werden.

Die Haftung von ABB, gleich aus welchem Rechtsgrund, im Zusammenhang mit den Anwendungsbeispielen oder den in dieser Datei enthaltenen Beschreibung wird ausgeschlossen. Der Haftungsausschluss gilt jedoch nicht in Fällen des Vorsatzes, der groben Fahrlässigkeit, bei Ansprüchen nach dem Produkthaftungsgesetz, im Falle der Verletzung des Lebens, des Körpers oder der Gesundheit oder bei schuldhafter Verletzung einer wesentlichen Vertragspflicht. Im Falle der Verletzung einer wesentlichen Vertragspflicht ist die Haftung jedoch auf den vertragstypischen, vorhersehbaren Schaden begrenzt, soweit nicht zugleich ein anderer der in Satz 2 dieses Unterabsatzes erwähnten Fälle gegeben ist. Eine Änderung der Beweislast zum Nachteil des Nutzers ist hiermit nicht verbunden.

Es gilt materielles deutsches Recht unter Ausschluss des UN-Kaufrechts."

2 Introduction

2.1 Scope of the document

The lifetime of the AC500 internal flash depends on the number of write and delete actions. Read actions will not influence the lifetime.

This application example program provides a way to use the internal Flash of AC500-V2 PLCs in a way, that the needed write and delete actions are minimized.
Assuming a use case where a set of data needs to be changed from time to time by the user. To avoid using the retain data area which needs a battery to keep the values in case of a power loss, the internal flash can be used to safely store the data.

This example provides a way to use all blocks of a segment to store the user data before a delete action of the whole segment is needed. This will increase the lifetime of the flash.

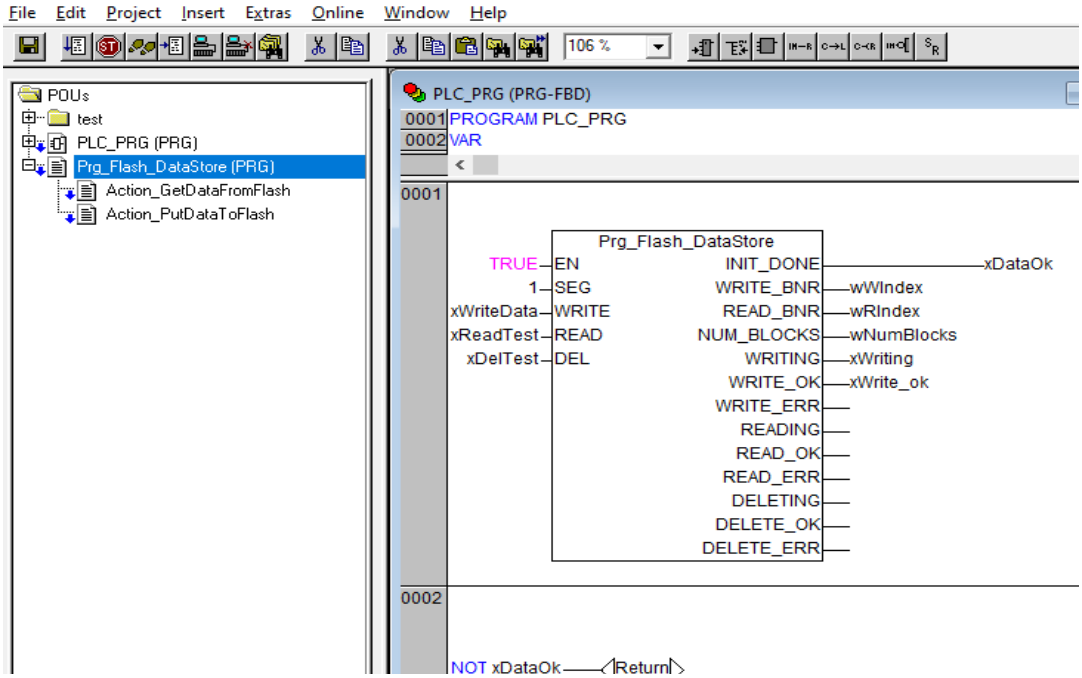
It contains a ready-made user program (with inputs and outputs) including the sequence to search the last written block at the startup of the PLC.

The program is provided as AC500-V2 project and as AC500-V2 export file to be included in already existing projects.

2.2 Compatibility

The application example explained in this document have been used with the below engineering system versions. They should also work with other versions, nevertheless some small adaptations may be necessary, for future versions.

- AC500 V2 PLC
- Automation Builder 2.4.0 or newer



3 General Flash handling

Find here some general info about the internal flash handling.

3.1 Docu in Automation Builder Help

Find here some official docu about the flash handling:

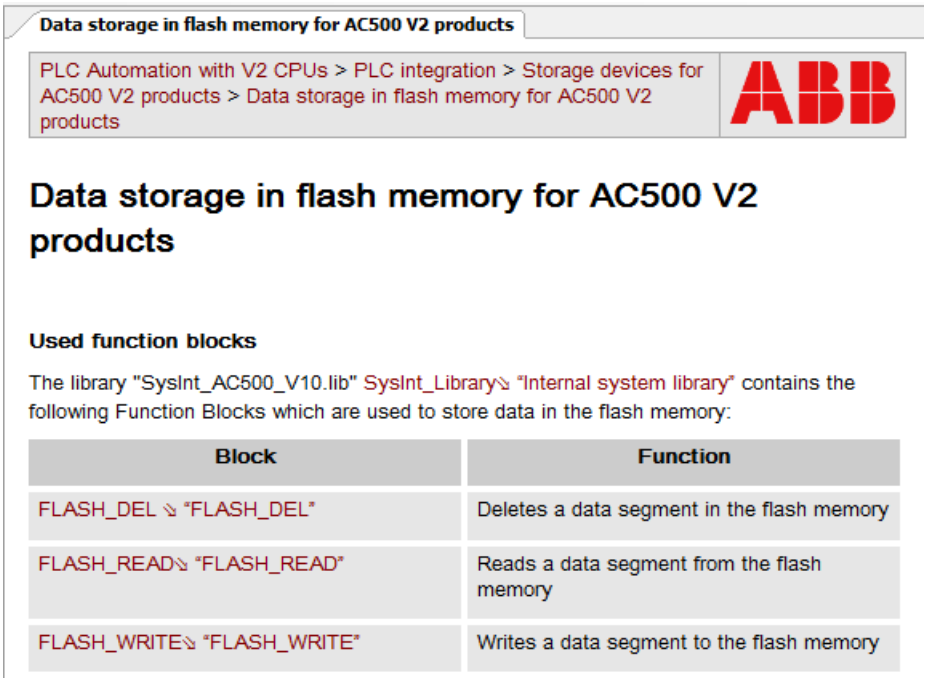
PLC Automation with V2 CPUs > PLC integration > Storage devices for AC500 V2 products > Introduction of AC500 storage devices for AC500 Products

3.1.1 Memory Size

PLC type	Flash memory
	User Data
PM554	128 kB
PM564	128 kB
PM554-ETH	128 kB
PM556-ETH	-
PM564-ETH	128 kB
PM572	128 kB
PM573-ETH	128 kB
PM582	128 kB
PM583-ETH	128 kB
PM585-ETH	-
PM590-ETH	128 kB
PM590-ARC	-
PM591-ETH	128 kB
PM591-2ETH	-
PM592-ETH	128 kB
PM595-4ETH-M	-
PM595-4ETH-F	-

3.1.2 Function blocks

PLC Automation with V2 CPUs > PLC integration > Storage devices for AC500 V2 products > Data storage in flash memory for AC500 V2 products



Data storage in flash memory for AC500 V2 products

PLC Automation with V2 CPUs > PLC integration > Storage devices for AC500 V2 products > Data storage in flash memory for AC500 V2 products

ABB

Data storage in flash memory for AC500 V2 products

Used function blocks

The library "SysInt_AC500_V10.lib" SysInt_Library "Internal system library" contains the following Function Blocks which are used to store data in the flash memory:

Block	Function
FLASH_DEL "FLASH_DEL"	Deletes a data segment in the flash memory
FLASH_READ "FLASH_READ"	Reads a data segment from the flash memory
FLASH_WRITE "FLASH_WRITE"	Writes a data segment to the flash memory

3.2 PM5xx: Access to the FLASH:

The access to the Flash is only possible with following FBs

- FLASH_DEL, FLASH_READ and FLASH_WRITE

3.2.1 Organization of the FLASH:

The Flash consist of 2 segments of each 64KB. These segments have the numbers 1 and 2.

Each segment is divided into 1927 blocks (0..1926) with 34 Byte per block.

The first 32 Byte of the block can be used for data storage. The remaining 2 Bytes are internally used for “CRC” and “Write identification” and therefore not accessible by the user.

3.2.2 Delete FLASH: (FLASH_DEL)

This FB is used to delete a segment (SEG) of the Flash. Only a whole segment with all its 1927 data blocks can be deleted. It’s not possible to delete single blocks inside the segment. After deleting a segment, the content of all (1927) blocks are cleared and enables an afterward write access to any block. The content of the deleted data cannot be seen, because a read to an unwritten block results in an error message and no data are “transferred” into the given destination address.

3.2.3 Read FLASH: (FLASH_READ)

This FB is used to read out single or multiple blocks from the designated segment. For doing so, the user has to specify the segment (SEG), the number of blocks (NB) to be read and the block number where the read starts (BNR). The required destination address, to which the read data is written, is determined by the input (SM) via FB ADR.

If a block is read which was not written to before, then the FB reports the error message 4101 at output ERNO. If none of the blocks were written to (after FLASH_DEL) and a read is performed, then the FB reports the error message 4131 at output ERNO.

In case of an error (4101, 4131), no data will be transferred to the given destination address. Therefore, the content is unchanged.

3.2.4 Write FLASH: (FLASH_WRITE)

This FB is used to write to single or multiple blocks in the designated segment. To perform a writing, the user has to specify the segment (SEG), the number of blocks (NB) to be written and the block number where the write starts (BNR). The input (SM) determine via the FB ADR the source address where the to be written bytes are located. If a block is written, then all available 32 Bytes inside the block are written at once, even if the length is shorter than 32 Bytes. In such a case, the data which are located directly behind the source data will be used to fill up the remaining bytes inside the block.

If a block is already written and a second write attempt to this block is carried out, then the error message 4101 occurs at output ERNO.

But the content is still overwritten with the new content!!!

3.2.5 Best practice to handle FLASH in PM5xx

Due to this behavior, it's possible to override a block inside the segment without a prior segment delete!

Therefore, one has to be carefully to avoid unintended overwriting of an already written block.

The detection whether the segment is deleted via "FLASH_DEL" can be done like this:

- Read a block of the segment with "FLASH_READ" FB

- Check the ERNO of the FB

If ERNO 4131 appears than the segment was deleted, and no block within this segment was written (all blocks are empty).

If ERNO 4101 appears than this block was not written but at least one other block within this segment was already written.

If no ERNO appears than this block was already written.

To avoid unintended rewrite of a block, first read the block with FB "FLASH_READ". If no ERNO appears after reading, then the block is already written (data available). Or with other words, if the FB reports ERNO 4101,4131 than the block is unused and can be written to.

This is used in the here described application example to use a storage method that reduces writing and deleting to the needed minimum. See next chapter

4 Application program to store new/changed data in next available blocks – Prg_Flash_DataStore

The application program "Prg_Flash_DataStore" can be called from within the main program or directly from the task configuration.

The output "INIT_DONE" should be used to enable the normal program, because only after this output is steady set the data are read from the flash after a restart of the PLC.

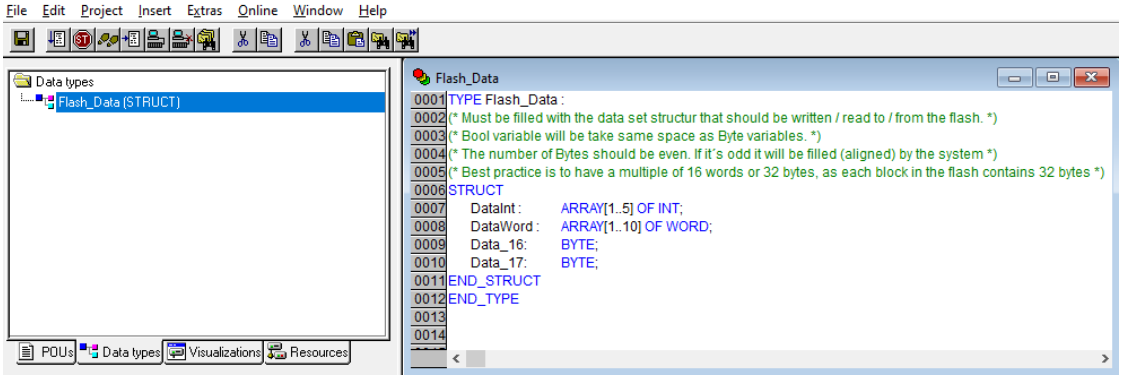
The following parts must be adapted to the user program:

4.1 Structur “Flash_Data”

As a kind of interface, the data is transferred from the user program to the flash via tow variables of the type “Flash_Data”.

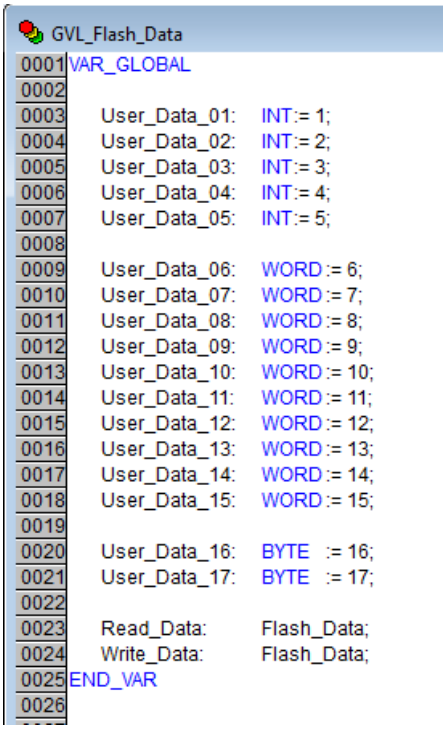
This structure has to be filled by the user and contains the data set structure that should be read/written from/to the flash. The structure declaration can be found in the tab “Data types”

In the example an array of 6 INT + an array of 10 WORD + 2 BYTE variables are considered as the user data set. (This occupies exactly 32 bytes = 1 block in the flash)



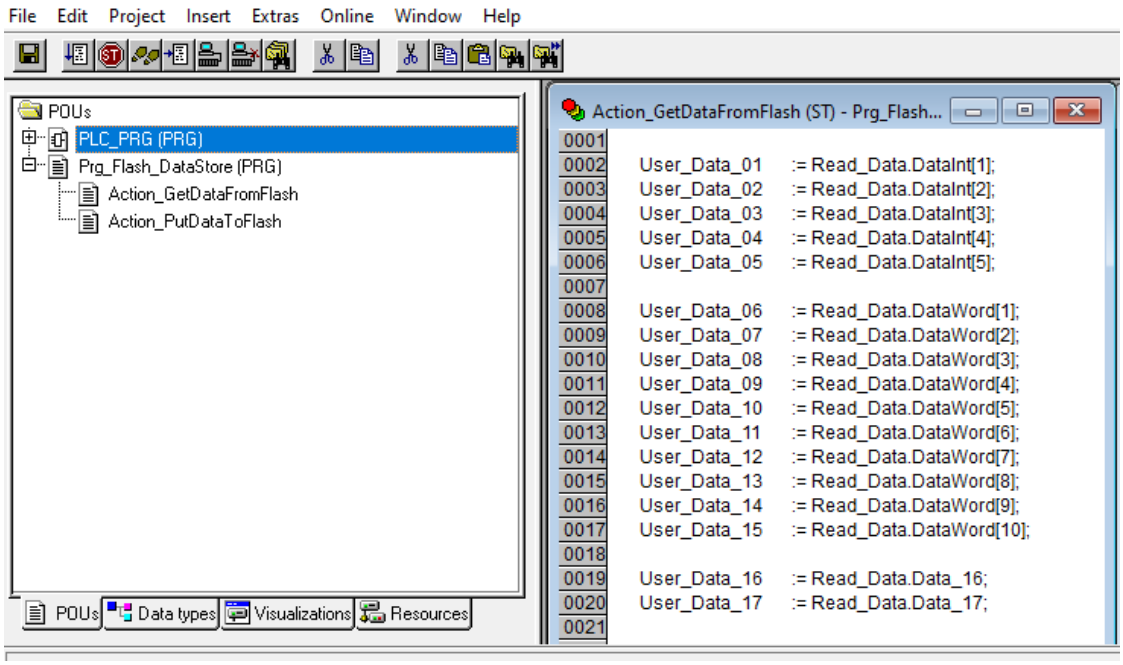
4.2 Global Variable list with user data and Flash_Data variables

In a global variable list the user data should be available and also the two variables of type “Flash_Data”. One to read the data from flash. The other to write the data to the flash. Both are used in the Prg_Flash_DataStore.



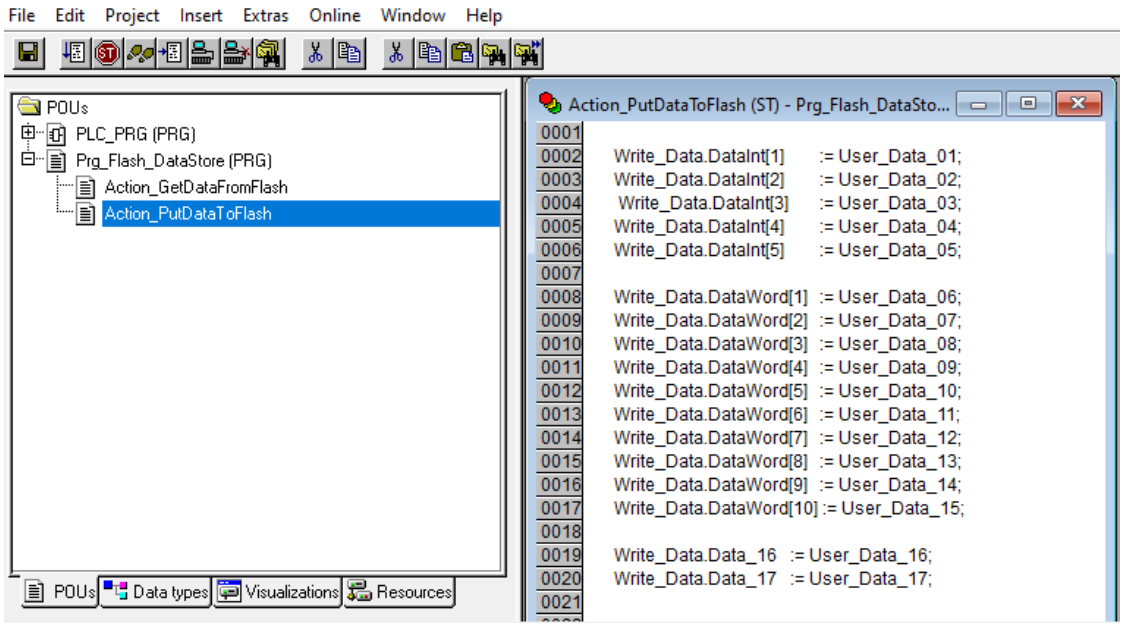
4.3 Action “Action_GetDataFromFlash”

This action needs to be filled by the user with the global variables that should get the data read from the flash.



4.4 Action “Action_PutDataToFlash”

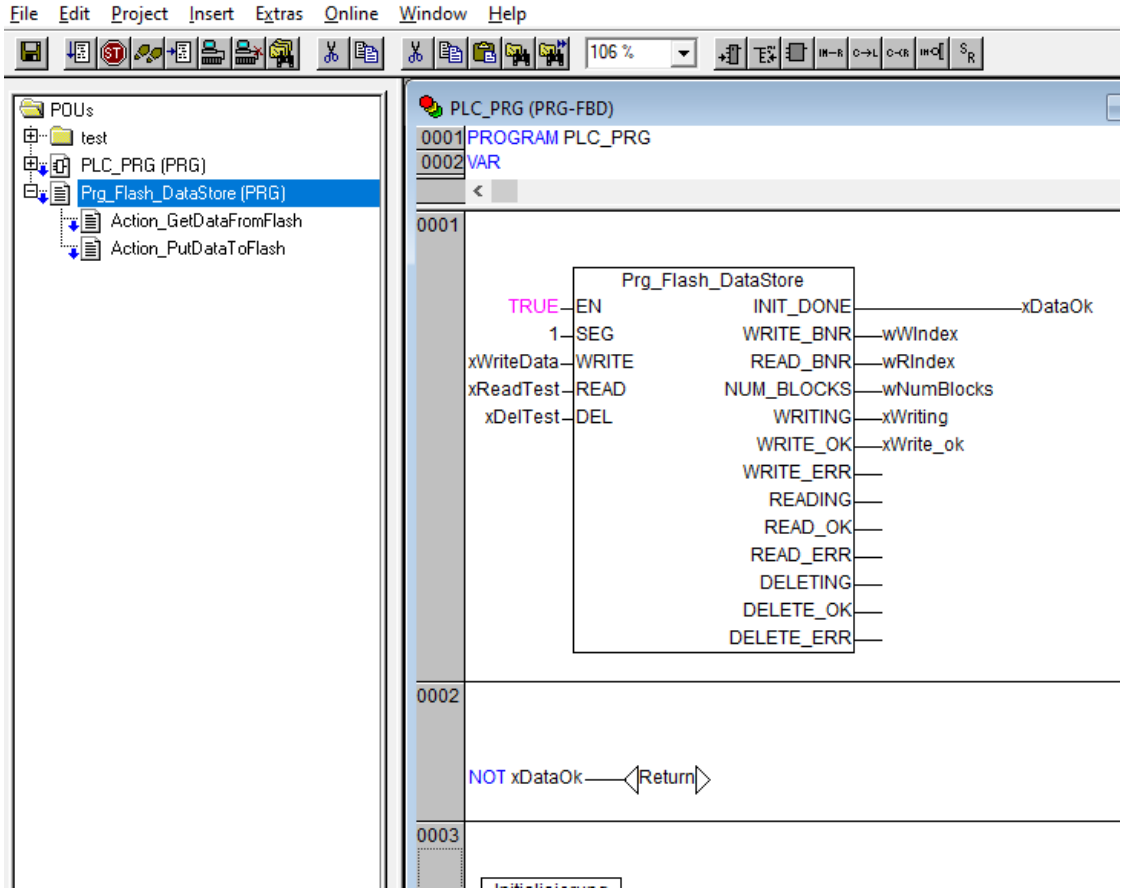
This action needs to be filled by the user with the global variables that should be stored to the flash.



4.5 Program “Prg_Flash_DataStore”

The program should be called at the first startup of the PLC and the output “INIT_DONE” should be used to release the other main programs. Because only after the INIT_DONE is set

the stored data was read from the flash. Depending on the number of already stored / changed data it can take up to 3 minutes (!) till the last written block is found.



4.5.1 Program Description

Handling of the flash to use the full range of one segment for maximum number of flash write accesses.

Input EN will enable the blocks processing. If the EN is set to FALSE the block will not process anymore, but all internal data will be kept and not be reset.

At the next time the EN input is set to TRUE again it will continue from the last execution.

The amount of data to be written and read is automatically calculated out of the struct "Flash_Data" under Data_types tab.

The number of blocks that will be read / written is shown on the output NUM_BLOCKS.

The first (minimum) block number and last (maximum) block number can be configured in the VAR_CONSTANT section of this declaration.

At startup of the PLC, an automatic search is performed to find the last written block(s). The data from this (these) blocks will be read out and written to the variables as programmed in the underlying Action: "Action_GetDataFromFlash".

If the last written block is already near the end of 1926 the search might need up to 3 minutes! Depending on the size of used data in the Flash_Data struct.

The block(s) is (are) read to find out, which is the first not already written block. As each Flash_Read might need up to 100ms this can take up to 1926 * 0,1s ~ 3 minutes.

If the search (initialization) is ended this is shown by a steady TRUE on the output INIT_DONE.

It´s highly recommended to use the output INIT_DONE to enable the other main PLC program. Otherwise (INIT_DONE = FALSE) the main PLC program might run with empty data!

At each rising edge on the input WRITE the underlying Action "Action_WriteDataToFlash" will be executed and the data defined there is written into the NEXT block(s) of the flash.

If the maximum number of blocks is reached, then the flash segment will be deleted, and the writing starts at the first block(s).

At each rising edge of the READ input the last written block(s) will be read out and the data will be written to the variables as programmed in the underlying Action: "Action_GetDataFromFlash".

It´s not possible to start several write / read of delete actions at the same time. The edge triggered inputs will be handled in the following order:

1. Initialization - no other action can be started before the INIT_DONE output is set to TRUE

2. WRITE has prio before READ and DEL

3. READ has prio before DEL

If two or more inputs of WRITE, READ, DEL are set at the same the time the not performed action needs to be retriggered again.

Check of the actions can be done by the outputs, which are set for one cycle mostly.

4.5.2 Inputs

Name	Type	Default value	Description
EN	BOOL	TRUE	Enable the block - level triggered - should be set to TRUE all the time
SEG	BYTE	1	Segment to be used: 1 or 2
WRITE	BOOL	FALSE	Write command as rising edge - triggers the PutDataToFlash Action, writes the data to the flash at actual WriteBlockNumber, set ReadBlockNumber to WriteBlockNumber and increases the WriteBlockNumber for next writting
READ	BOOL	FALSE	Read command as rising edge - reads the flash at actual ReadBlockNumber and if sucessfull triggers the GetDataFromFlash Action
DEL	BOOL	FALSE	Delete command as rising edge - deletes the whole flash segment

4.5.3 Outputs

Name	Type	Default value	Description
INIT_DONE	BOOL	FALSE	Initialisation done - search for last written block finished - see index - will last longer as higher the last written index was - up to 4 minutes at 15ms task
WRITE_BNR	WORD	0	Block number of the block that will be written by next writing - will be increase by wNumBlocks after the writing
READ_BNR	WORD	0	Block number which will be read at a reading - will be set to the last written block number after writing
NUM_BLOCKS	WORD	1	Number of blocks to read / write
WRITING	BOOL	FALSE	Writing block is ongoing
WRITE_OK	BOOL	FALSE	Write successfully done - true for one cycle
WRITE_ERR	BOOL	FALSE	Write finished with error - true for one cycle
READING	BOOL	FALSE	Reading block is ongoing
READ_OK	BOOL	FALSE	Read successfully done - true for one cycle
READ_ERR	BOOL	FALSE	Read finished with error - true for one cycle
DELETING	BOOL	FALSE	Deleting flash is ongoing
DELETE_OK	BOOL	FALSE	Deleting flash successfully done - true for one cycle
DELETE_ERR	BOOL	FALSE	Deleting flash finished with error - true for one cycle

5 Distribution

5.1 Application project

The application program is available as Automation Builder 2.0 project for a PM554-ETH PLC. (Created in Automation Builder 2.4.0 – Profile 2.0)

This can be upgraded to any newer Automation Builder and the PLC type can also be changed without any disadvantages.

5.2 Export file

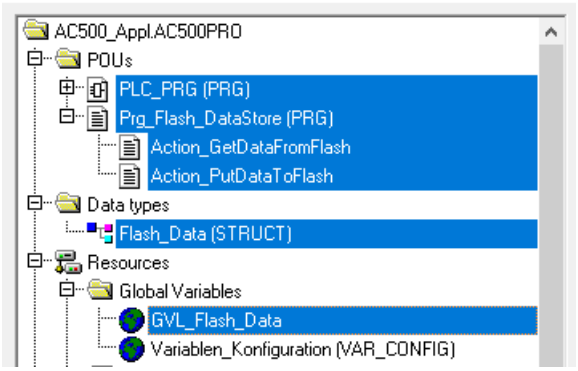
ApplExample_Prg_Flash_DataStore.exp for CODESYS 2.3.9 IEC61131 programming editor.

Use Project > Import function to integrate in your already existing project.

This includes the following parts:

- POUs
  - Program: Prg_Flash_DataStore including the
    - Action_GetDataFromFlash
    - Action_PutDataToFlash
  - Program: PLC_PRG (to see how the Prg_Flash_DataStore can be included)
- Data types
  - Flash_Data
- Global Varlist
  - GVL_Flash_Data (inlcuding a possible user data set and the two structure variables Read_Data and Write_Data of typ “Flash_Data”

Export Project



6 Special Hints

6.1 Caution

CAUTION!

Generally, the user in all applications is fully and alone responsible for checking all functions carefully, especially for safe and reliable operation.

6.2 Note

Note: The Function Blocks contained in the example can only be executed in RUN mode of the PLC, but not in simulation mode.

ABB Automation Products GmbH
Eppelheimer Straße 82
69123 Heidelberg, Germany
Phone: +49 62 21 701 1444
Fax: +49 62 21 701 1382
E-Mail: plc.support@de.abb.com
www.abb.com/plc